



1

2 **OASIS eXtensible Access Control**  
3 **Markup Language (XACML)**

4 **Committee Specification 1.0, 7 November**  
5 **2002**

6 Document identifier: cs-xacml-specification-1.0.doc

7 Location: <http://www.oasis-open.org/committees/xacml/docs/>

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

9 Editors:

10 Simon Godik, Overxeer ([simon.godik@overxeer.com](mailto:simon.godik@overxeer.com))  
11 Tim Moses, Entrust ([tim.moses@entrust.com](mailto:tim.moses@entrust.com))

12 Contributors:

13 Anne Anderson, Sun Microsystems  
14 Bill Parducci, Overxeer  
15 Carlisle Adams, Entrust  
16 Daniel Engovatov, CrossLogix  
17 Don Flinn, Quadrasis  
18 Ernesto Damiani, University of Milan  
19 James MacLean, Affinitex  
20 Hal Lockhart, Entegriy  
21 Ken Yagen, CrossLogix  
22 Konstantin Beznosov, Quadrasis  
23 Michiharu Kudo, IBM  
24 Pierangela Samarati, University of Milan  
25 Pirasenna Velandai Thiyagarajan, Sun Microsystems  
26 Polar Humenn, Syracuse University  
27 Sekhar Vajjhala, Sun Microsystems  
28 Seth Proctor, Sun Microsystems  
29 Steve Anderson, OpenNetworks  
30 Steve Crocker, Pervasive Security Systems  
31 Suresh Damodaran, Sterling Commerce  
32 Gerald Brose, Xtradyne

33 Abstract:

34 This specification defines an XML schema for an extensible access-control policy  
35 language.

36 Status:

cs-xacml-specification-1.0.doc

37 This version of the specification is a working draft of the committee. As such, it is expected  
38 to change prior to adoption as an OASIS standard.

39 If you are on the [xacml@lists.oasis-open.org](mailto:xacml@lists.oasis-open.org) list for committee members, send comments  
40 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  
41 and send comments there. To subscribe, send an email message to [xacml-comment-](mailto:xacml-comment-request@lists.oasis-open.org)  
42 [request@lists.oasis-open.org](mailto:xacml-comment-request@lists.oasis-open.org) with the word "subscribe" as the body of the message.

43

44 Copyright (C) OASIS Open 2002. All Rights Reserved.

45	<b>Table of contents</b>	
46	1. Introduction (non-normative)	9
47	1.1. Glossary	9
48	1.1.1 Preferred terms	9
49	1.1.2 Related terms	10
50	1.2. Notation	10
51	1.3. Schema organization and namespaces	11
52	2. Background (non-normative)	11
53	2.1. Requirements	12
54	2.2. Rule and policy combining	13
55	2.3. Combining algorithms	13
56	2.4. Multiple subjects	14
57	2.5. Policies based on subject and resource attributes	14
58	2.6. Multi-valued attributes	14
59	2.7. Policies based on resource contents	15
60	2.8. Operators	15
61	2.9. Policy distribution	16
62	2.10. Policy indexing	16
63	2.11. Abstraction layer	16
64	2.12. Actions performed in conjunction with enforcement	17
65	3. Models (non-normative)	17
66	3.1. Data-flow model	17
67	3.2. XACML context	19
68	3.3. Policy language model	19
69	3.3.1 Rule	20
70	3.3.2 Policy	22
71	3.3.3 Policy set	23
72	4. Examples (non-normative)	23
73	4.1. Example one	24
74	4.1.1 Example policy	24
75	4.1.2 Example request context	26
76	4.1.3 Example response context	27
77	4.2. Example two	28
78	4.2.1 Example medical record instance	28
79	4.2.2 Example request context	29
80	4.2.3 Example plain-language rules	31

81	4.2.4	Example XACML rule instances	31
82	5.	Policy syntax (normative, with the exception of the schema fragments)	45
83	5.1.	Element <PolicySet>	45
84	5.2.	Element <Description>	46
85	5.3.	Element <PolicySetDefaults>	46
86	5.4.	Element <XPathVersion>	47
87	5.5.	Element <Target>	47
88	5.6.	Element <Subjects>	48
89	5.7.	Element <Subject>	48
90	5.8.	Element <AnySubject>	48
91	5.9.	Element <SubjectMatch>	48
92	5.10.	Element <Resources>	49
93	5.11.	Element <Resource>	49
94	5.12.	Element <AnyResource>	50
95	5.13.	Element <ResourceMatch>	50
96	5.14.	Element <Actions>	51
97	5.15.	Element <Action>	51
98	5.16.	Element <AnyAction>	51
99	5.17.	Element <ActionMatch>	52
100	5.18.	Element <PolicySetIdReference>	52
101	5.19.	Element <PolicyIdReference>	52
102	5.20.	Element <Policy>	53
103	5.21.	Element <Rule>	54
104	5.22.	Simple type EffectType	55
105	5.23.	Element <Condition>	55
106	5.24.	Element <Apply>	55
107	5.25.	Element <Function>	56
108	5.26.	Complex type AttributeDesignatorType	56
109	5.27.	Element <ResourceAttributeDesignator>	57
110	5.28.	Element <ActionAttributeDesignator>	58
111	5.29.	Element <EnvironmentAttributeDesignator>	59
112	5.30.	Complex type SubjectAttributeDesignatorType	60
113	5.31.	Element <AttributeSelector>	61
114	5.32.	Element <AttributeValue>	62
115	5.33.	Element <Obligations>	62
116	5.34.	Element <Obligation>	63
117	5.35.	Element <AttributeAssignment>	63

118	6.	Context syntax (normative with the exception of the schema fragments)	64
119	6.1.	Element <Request>	64
120	6.2.	Element <Subject>	65
121	6.3.	Element <Resource>	65
122	6.4.	Element <ResourceContent>	66
123	6.5.	Element <Action>	66
124	6.6.	Element <Environment>	66
125	6.7.	Element <Attribute>	67
126	6.8.	Element <AttributeValue>	68
127	6.9.	Element <Response>	68
128	6.10.	Element <Result>	68
129	6.11.	Element <Decision>	69
130	6.12.	Element <Status>	69
131	6.13.	Element <StatusCode>	70
132	6.14.	Element <StatusMessage>	70
133	6.15.	Element <StatusDetail>	70
134	7.	Functional requirements (normative)	71
135	7.1.	Policy enforcement point	71
136	7.2.	Base policy	72
137	7.3.	Target evaluation	72
138	7.4.	Condition evaluation	72
139	7.5.	Rule evaluation	72
140	7.6.	Policy evaluation	73
141	7.7.	Policy Set evaluation	73
142	7.8.	Hierarchical resources	74
143	7.9.	Attributes	75
144	7.9.1.	Attribute Matching	75
145	7.9.2.	Attribute Retrieval	75
146	7.9.3.	Environment Attributes	76
147	7.9.4.	Subject Attributes	76
148	7.10.	Authorization decision	76
149	7.11.	Obligations	76
150	8.	XACML extensibility points (non-normative)	77
151	8.1.	Extensible XML attribute types	77
152	8.2.	Extensible XACML attribute types	77
153	8.3.	Structured attributes	78
154	9.	Security and privacy considerations (non-normative)	78

155	9.1.	Threat model	78
156	9.1.1.	Unauthorized disclosure	79
157	9.1.2.	Message replay	79
158	9.1.3.	Message insertion	79
159	9.1.4.	Message deletion	79
160	9.1.5.	Message modification	79
161	9.1.6.	Not-applicable results	79
162	9.1.7.	Negative rules	80
163	9.2.	Safeguards	81
164	9.2.1.	Authentication	81
165	9.2.2.	Policy administration	81
166	9.2.3.	Confidentiality	81
167	9.2.4.	Policy integrity	82
168	9.2.5.	Policy identifiers	82
169	9.2.6.	Trust model	83
170	9.2.7.	Privacy	83
171	10.	Conformance (normative)	83
172	10.1.	Introduction	83
173	10.2.	Attestation	84
174	10.3.	Conformance tables	84
175	10.3.1.	Schema elements	84
176	10.3.2.	Identifier Prefixes	85
177	10.3.3.	Algorithms	85
178	10.3.4.	Status Codes	85
179	10.3.5.	Attributes	86
180	10.3.6.	Identifiers	86
181	10.3.7.	Data Types	86
182	10.3.8.	Functions	87
183	11.	References	89
184	Appendix A.	Standard data types, functions and their semantics (normative)	91
185	A.1.	Introduction	91
186	A.2.	Primitive types	91
187	A.3.	Structured types	92
188	A.4.	Representations	92
189	A.5.	Bags	93
190	A.6.	Expressions	93
191	A.7.	Element <AttributeValue>	94

192	A.8.	Elements <AttributeDesignator> and <AttributeSelector>	94
193	A.9.	Element <Apply>	94
194	A.10.	Element <Condition>	94
195	A.11.	Element <Function>	95
196	A.12.	Matching elements	95
197	A.13.	Arithmetic evaluation	96
198	A.14.	XACML standard functions	97
199	A14.1	Equality predicates	97
200	A14.2	Arithmetic functions	99
201	A14.3	String conversion functions	99
202	A14.4	Numeric type conversion functions	100
203	A14.5	Logical functions	100
204	A14.6	Arithmetic comparison functions	101
205	A14.7	Date and time arithmetic functions	102
206	A14.8	Non-numeric comparison functions	103
207	A14.9	Bag functions	105
208	A14.10	Set functions	106
209	A14.11	Higher-order bag functions	106
210	A14.12	Special match functions	113
211	A14.13	XPath-based functions	114
212	A14.14	Extension functions and primitive types	114
213		Appendix B. XACML identifiers (normative)	115
214	B.1.	XACML namespaces	115
215	B.2.	Access subject categories	115
216	B.3.	XACML functions	115
217	B.4.	Data types	116
218	B.5.	Subject attributes	116
219	B.6.	Resource attributes	117
220	B.7.	Action attributes	118
221	B.8.	Environment attributes	118
222	B.9.	Status codes	118
223	B.10.	Combining algorithms	119
224		Appendix C. Combining algorithms (normative)	120
225	C.1.	Deny-overrides	120
226	C.2.	Permit-overrides	122
227	C.3.	First-applicable	124
228	C.4.	Only-one-applicable	125

229	Appendix D. Acknowledgments	127
230	Appendix E. Revision history	128
231	Appendix F. Notices	129
232		
233		

---

## 234 1. Introduction (non-normative)

### 235 1.1. Glossary

#### 236 1.1.1 Preferred terms

237 **Access** - Performing an *action*

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

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

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

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

244 **Authorization decision** - The result of evaluating *applicable policy*, returned by the *PDP* to the  
245 *PEP*. A function that evaluates to "Permit", "Deny", "Indeterminate" or "Not-applicable", and  
246 (optionally) a set of *obligations*

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

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

250 **Conjunctive sequence** - a sequence of elements combined using the logical 'AND' operation

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

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

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

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

257 **Disjunctive sequence** - a sequence of elements combined using the logical 'OR' 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 1.1.2 Related terms

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

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

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

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

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

## 290 1.2. Notation

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

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

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

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

302 Listings of XACML schemas appear like this.

303

304 Example code listings appear like this.

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

- 308 • The prefix `saml:` stands for the SAML assertion namespace [SAML].
- 309 • The prefix `ds:` stands for the W3C XML Signature namespace [DS].
- 310 • The prefix `xs:` stands for the W3C XML Schema namespace [XS].
- 311 • The prefix `xf:` stands for the XPath query and function specification namespace [XF].

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

### 315 1.3. Schema organization and namespaces

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

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

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

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

320 XACML data-types are defined in the following XML namespace:

321 `urn:oasis:names:tc:xacml:1.0:data-type`

322 The XML Signature `XMLSigXSD` is imported into the XACML schema and is associated with the  
323 following XML namespace:

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

---

## 325 2. Background (non-normative)

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

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

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

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

## 352 2.1. Requirements

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

- 354 • To provide a method for combining individual **rules** and **policies** into a single **policy set** that  
355 applies to a given action.
- 356 • To provide a method for flexible definition of the procedure by which **rules** and **policies** are  
357 combined.
- 358 • To provide a method for dealing with multiple **subjects** acting in different capacities.
- 359 • To provide a method for basing an **authorization decision** on **attributes** of the **subject** and  
360 **resource**.
- 361 • To provide a method for dealing with multi-valued **attributes**.
- 362 • To provide a method for basing an **authorization decision** on the contents of an information  
363 **resource**.
- 364 • To provide a set of logical and mathematical operators on **attributes** of the **subject**, **resource**  
365 and **environment**.
- 366 • To provide a method for handling a distributed set of **policy** components, while abstracting the  
367 method for locating, retrieving and authenticating the **policy** components.
- 368 • To provide a method for rapidly identifying the **policy** that applies to a given action, based upon  
369 the values of **attributes** of the **subjects**, **resource** and **action**.
- 370 • To provide an abstraction-layer that insulates the policy-writer from the details of the application  
371 environment.

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

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

## 377 2.2. Rule and policy combining

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

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

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

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

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

## 397 2.3. Combining algorithms

398 XACML defines a number of combining algorithms that can be identified by a  
399 RuleCombiningAlgId or PolicyCombiningAlgId attribute of the <Policy> or <PolicySet>  
400 elements, respectively. The *rule-combining algorithm* defines a procedure for arriving at an  
401 *authorization decision* given the individual results of evaluation of a set of *rules*. Similarly, the  
402 *policy-combining algorithm* defines a procedure for arriving at an *authorization decision* given  
403 the individual results of evaluation of a set of *policies*. Standard combining algorithms are defined  
404 for:

- 405 • Deny-overrides,
- 406 • Permit-overrides,
- 407 • First applicable and
- 408 • Only-one-applicable.

409 In the first case, if a single <Rule> or <Policy> element is encountered that evaluates to "Deny",  
410 then, regardless of the evaluation result of the other <Rule> or <Policy> elements in the  
411 *applicable policy*, the combined result is "Deny". Likewise, in the second case, if a single "Permit"  
412 result is encountered, then the combined result is "Permit". In the case of the "First-applicable"

413 combining algorithm, the combined result is the same as the result of evaluating the first <Rule>,  
414 <Policy> or <PolicySet> element in the list of **rules** whose **target** is applicable to the **decision**  
415 **request**. The "Only-one-applicable" **policy-combining algorithm** only applies to **policies**. The  
416 result of this combining algorithm ensures that one and only one **policy** or **policy set** is applicable  
417 by virtue of their **targets**. If no **policy** or **policy set** applies, then the result is "Not-applicable", but  
418 if more than one **policy** or **policy set** is applicable, then the result is "Indeterminate". When exactly  
419 one **policy** or **policy set** is applicable, the result of the combining algorithm is the result of  
420 evaluating the single **applicable policy** or **policy set**.

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

## 422 **2.4. Multiple subjects**

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

## 429 **2.5. Policies based on subject and resource attributes**

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

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

## 446 **2.6. Multi-valued attributes**

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

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

## 455 2.7. Policies based on resource contents

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

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

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

## 466 2.8. Operators

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

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

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

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

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

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

494 The XACML method of representing functions borrows from MathML [MathML] and from XPath  
495 Query and Functions [XF].

496

## 2.9. Policy distribution

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

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

## 2.10. Policy indexing

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

514 Two approaches are supported:

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

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

## 2.11. Abstraction layer

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

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

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

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

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

## 548 **2.12. Actions performed in conjunction with enforcement**

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

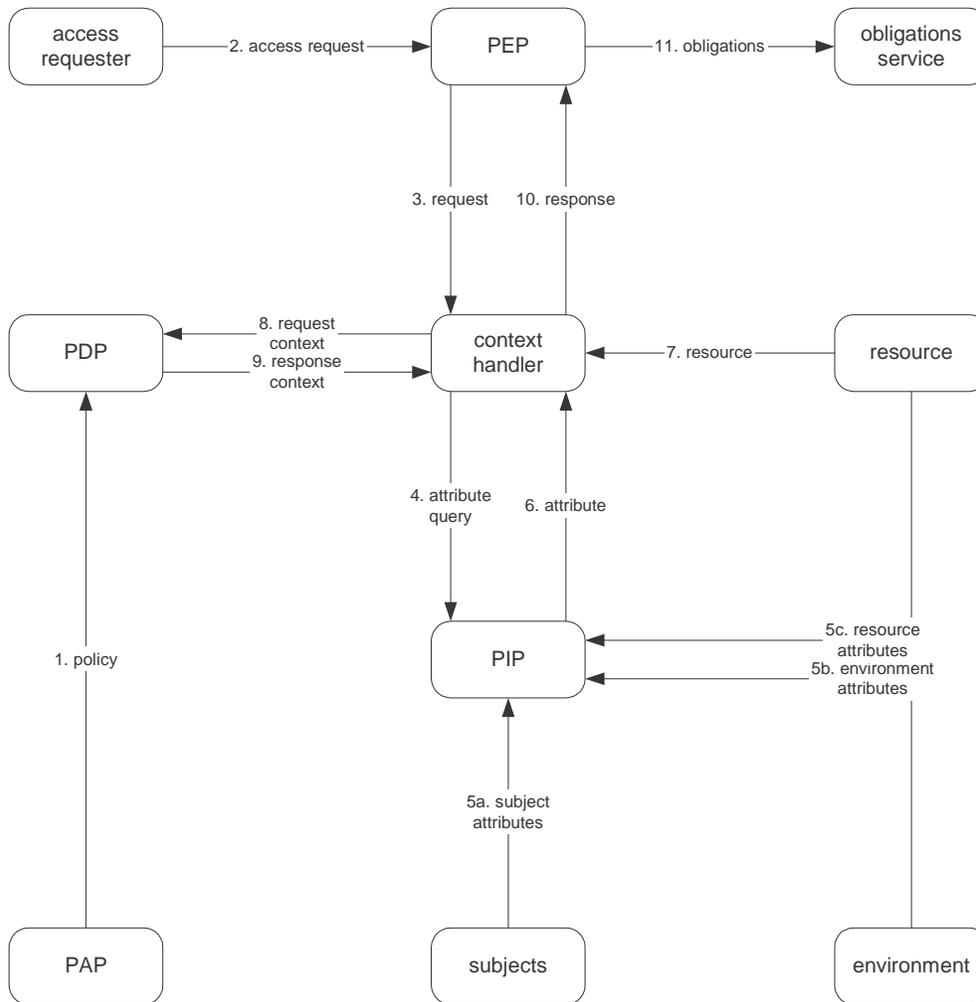
---

## 558 **3. Models (non-normative)**

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

### 560 **3.1. Data-flow model**

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



562

563

**Figure 1 - Data-flow diagram**

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

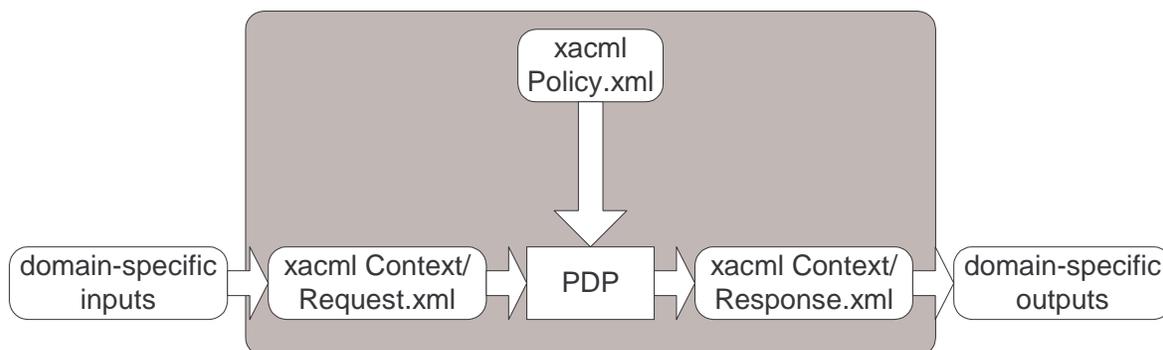
569 The model operates by the following steps.

- 570 1. **PAPs** write **policies** and make them available to the **PDP**. Its **policies** represent the complete  
 571 policy for a specified **target**.
- 572 2. The access requester sends a request for access to the **PEP**.
- 573 3. The **PEP** sends the request for **access** to the **context handler** in its native request format,  
 574 optionally including **attributes** of the **subjects**, **resource** and **action**. The **context handler**  
 575 constructs an XACML request **context** in accordance with steps 4,5,6 and 7.
- 576 4. **Subject**, **resource** and **environment attributes** may be requested from a **PIP**.
- 577 5. The **PIP** obtains the requested **attributes**.

- 578 6. The **PIP** returns the requested **attributes** to the **context handler**.
- 579 7. Optionally, the **context handler** includes the **resource** in the **context**.
- 580 8. The **context handler** makes information about the request **context** available to the **PDP**. The
- 581 **PDP** identifies the **policy** applicable to the request **context**. The **PDP** evaluates the **policy**.
- 582 9. The **PDP** returns the response **context** (including the **authorization decision**) to the **context**
- 583 **handler**.
- 584 10. The **context handler** translates the response **context** to the native response format of the
- 585 **PEP**. The **context handler** returns the response to the **PEP**.
- 586 11. The **PEP** fulfills the **obligations**.
- 587 12. (Not shown) If **access** is permitted, then the **PEP** permits **access** to the **resource**; otherwise, it
- 588 denies **access**.

### 589 3.2. XACML context

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



601

602

Figure 2 - XACML context

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

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

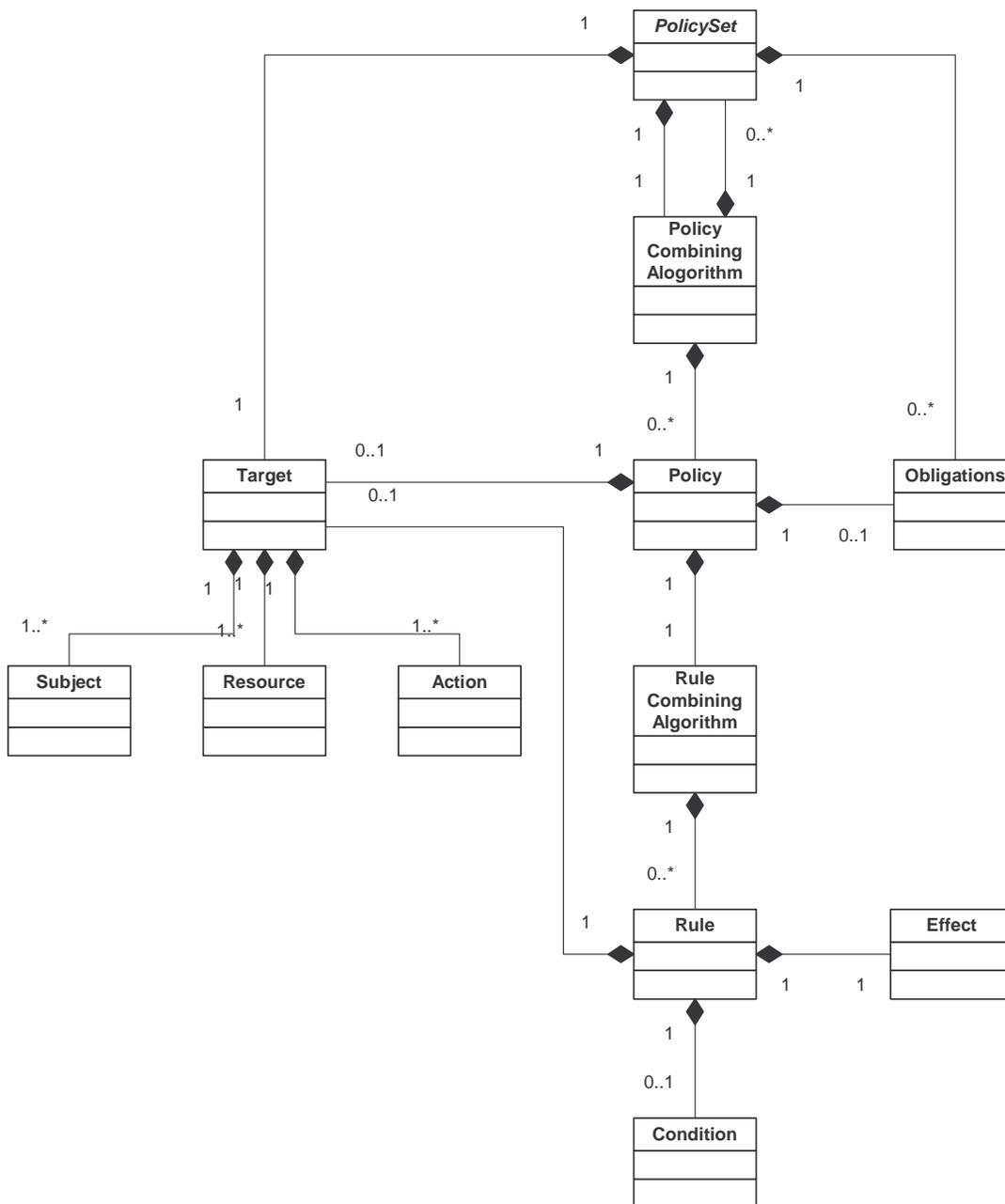
### 605 3.3. Policy language model

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

- 607 • **Rule**;
- 608 • **Policy**; and

609 • **Policy set.**

610 These are described in the following sub-sections.



611

612

**Figure 3 - Policy language model**

613

### 3.3.1 Rule

614 The main components of a **rule** are:

615 • a **target**,

- 616 • an *effect*; and  
617 • a *condition*.

618 These are discussed in the following sub-sections.

### 619 3.3.1.1. Rule target

620 The *target* defines the set of:

- 621 • *resources*;  
622 • *subjects*; and  
623 • *actions*

624 to which the *rule* is intended to apply. The <Condition> element may further refine the  
625 applicability established by the *target*. If the *rule* is intended to apply to all entities of a particular  
626 type, then an empty element named <AnySubject/>, <AnyResource/> or <AnyAction/> is  
627 used. An XACML *PDP* verifies that the *subjects*, *resource* and *action* identified in the request  
628 *context* are all present in the *target* of the *rules* that it uses to evaluate the *decision request*.  
629 *Target* definitions are discrete, in order that applicable *rules* may be efficiently identified by the  
630 *PDP*.

631 The <Target> element may be absent from a <Rule>. In this case, the <Rule> inherits its *target*  
632 from the parent <Policy> element.

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

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

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

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

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

- 654 1. the contents of the identified node only,  
655 2. the contents of the identified node and the contents of its immediate child nodes or  
656 3. the contents of the identified node and all its descendant nodes.

657 All three options are supported in XACML.

658

### 3.3.1.2. Effect

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

661

### 3.3.1.3. Condition

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

664

## 3.3.2 Policy

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

- 667 • a **target**;
- 668 • a **rule-combining algorithm**-identifier;
- 669 • a set of **rules**; and
- 670 • **obligations**.

671 **Rules** are described above. The remaining components are described in the following sub-  
672 sections.

673

### 3.3.2.1. Policy target

674 An XACML <PolicySet>, <Policy> or <Rule> element contains a <Target> element that  
675 specifies the set of **subjects**, **resources** and **actions** to which it applies. The <Target> of a  
676 <PolicySet> or <Policy> may be declared by the writer of the <PolicySet> or <Policy>, or  
677 it may be calculated from the <Target> elements of the <PolicySet>, <Policy> and <Rule>  
678 elements that it contains.

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

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

### 695 3.3.2.2. Rule-combining algorithm

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

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

### 701 3.3.2.3. Obligations

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

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

## 706 3.3.3 Policy set

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

- 708 • a **target**,
- 709 • a **policy-combining algorithm**-identifier
- 710 • a set of **policies**; and
- 711 • **obligations**.

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

### 714 3.3.3.1. Policy-combining algorithm

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

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

### 720 3.3.3.2. Obligations

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

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

---

## 725 4. Examples (non-normative)

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

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

## 730 4.1. Example one

### 731 4.1.1 Example policy

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

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

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

738 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">
```

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

741 Line [p02] introduces the XACML Policy itself.

742 Lines [p03-p05] are XML namespace declarations.

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

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

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

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

752 Lines [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>
```

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

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

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

760 Line [p23] specifies the identifier for this **rule**.

761 Line [p24] says what **effect** this **rule** has if the **rule** evaluates to "True". **Rules** can have an **effect**  
762 of either "Permit" or "Deny". In this case, the rule will evaluate to "Permit", meaning that, as far as  
763 this one **rule** is concerned, the requested **access** should be permitted. If a **rule** evaluates to  
764 "False", then it returns a result of "Not-applicable". If an error occurs when evaluating the **rule**, the  
765 **rule** returns a result of "Indeterminate". As mentioned above, the **rule-combining algorithm** for  
766 the **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>
```

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

```
[p29] <Target>
```

768 Line [p29] introduces the **target** of the **rule**. As described above for the **target** of a policy, the  
769 **target** of a **rule** describes the **decision requests** to which this **rule** applies. If the **subject**,  
770 **resource** and **action** in a **decision request** do not match the values specified in the **rule target**,  
771 then the remainder of the **rule** does not need to be evaluated, and a value of "Not-applicable" is  
772 returned to 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>
```

773 The **rule target** is similar to the **target** of the **policy** itself, but with one important difference. Lines  
774 [p32-p41] do not say `<AnySubject/>`, but instead spell out a specific value that the **subject** in the  
775 **decision request** must match. The `<SubjectMatch>` element specifies a matching function in  
776 the `MatchId` attribute, a pointer to a specific **subject attribute** in the request **context** by means of  
777 the `<SubjectAttributeDesignator>` element, and a literal value of "medico.com". The  
778 matching function will be used to compare the value of the **subject attribute** with the literal value.  
779 Only if the match returns "True" will this **rule** apply to a particular **decision request**. If the match  
780 returns "False", then this **rule** will return a value of "Not-applicable".

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

781 Line [p49] closes the **rule** we have been examining. In this **rule**, all the *work* is done in the  
782 `<Target>` element. In more complex **rules**, the `<Target>` may have been followed by a  
783 `<Condition>` (which could also be a set of **conditions** to be *AND*ed or *OR*ed together).

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

## 786 4.1.2 Example request context

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

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

792 In XACML, the information in the **decision request** is formatted into a **request context** statement  
793 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/sc-xacml-schema-context-01.xsd">
```

794 Lines [c01-c05] are the header for the **request context**, and are used the same way as the header  
795 for the **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>
```

796 The `<Subject>` element contains one or more **attributes** of the entity making the **access** request.  
797 There can be multiple **subjects**, and each **subject** can have multiple **attributes**. In this case, in  
798 lines [c06-c11], there is only one **subject**, and the **subject** has only one **attribute**: the **subject's**  
799 identity, 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>
```

800 The <Resource> element contains one or more **attributes** of the **resource** to which the **subject**  
801 (or **subjects**) has requested **access**. There can be only one <Resource> per **decision request**.  
802 Lines [c13-c16] contain the one **attribute** of the **resource** to which Bart Simpson has requested  
803 **access**: the **resource** unix file-system path-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>
```

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

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

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

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

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

### 818 4.1.3 Example response context

819 As a result, there is no **rule** in this **policy** that returns a "Permit" result for this request. The **rule-**  
820 **combining algorithm** for the **policy** specifies that, in this case, a result of "Not-applicable" should  
821 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/sc-xacml-schema-context-01.xsd">
```

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

```
[r05] <Result>  
[r06] <Decision>Not-applicable</Decision>  
[r07] </Result>
```

824 The <Result> element in lines [r05-r07] contains the result of evaluating the **decision request**  
825 against the **policy**. In this case, the result is `"Not-applicable"`. A **policy** can return `"Permit"`, `"Deny"`,  
826 `"Not-applicable"` or `"Indeterminate"`.

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

827 Line [r08] closes the response **context**.

828

## 4.2. Example two

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

832

### 4.2.1 Example medical record instance

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

```
836 <?xml version="1.0" encoding="UTF-8"?>
837 <record xmlns="http://www.medico.com/schemas/record.xsd "
838 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
839 <patient>
840 <patientName>
841 <first>Bartholomew</first>
842 <last>Simpson</last>
843 </patientName>
844 <patientContact>
845 <street>27 Shelbyville Road</street>
846 <city>Springfield</city>
847 <state>MA</state>
848 <zip>12345</zip>
849 <phone>555.123.4567</phone>
850 <fax/>
851 <email/>
852 </patientContact>
853 <patientDoB http://www.w3.org/2001/XMLSchema#type="date">1992-03-
854 21</patientDoB>
855 <patientGender
856 http://www.w3.org/2001/XMLSchema#type="string">male</patientGender>
857 <policyNumber
858 http://www.w3.org/2001/XMLSchema#type="string">555555</policyNumber>
859 </patient>
860 <parentGuardian>
861 <parentGuardianId>HS001</parentGuardianId>
862 <parentGuardianName>
863 <first>Homer</first>
864 <last>Simpson</last>
865 </parentGuardianName>
866 <parentGuardianContact>
867 <street>27 Shelbyville Road</street>
868 <city>Springfield</city>
869 <state>MA</state>
870 <zip>12345</zip>
871 <phone>555.123.4567</phone>
872 <fax/>
873 <email>homers@aol.com</email>
874 </parentGuardianContact>
875 </parentGuardian>
876 <primaryCarePhysician>
877 <physicianName>
878 <first>Julius</first>
879 <last>Hibbert</last>
880 </physicianName>
881 <physicianContact>
882 <street>1 First St</street>
883 <city>Springfield</city>
884 <state>MA</state>
```

```

885     <zin>12345</zin>
886     <phone>555.123.9012</phone>
887     <fax>555.123.9013</fax>
888     <email/>
889   </physicianContact>
890   <registrationID>ABC123</registrationID>
891 </primaryCarePhysician>
892 <insurer>
893   <name>Blue Cross</name>
894   <street>1234 Main St</street>
895   <city>Springfield</city>
896   <state>MA</state>
897   <zip>12345</zip>
898   <phone>555.123.5678</phone>
899   <fax>555.123.5679</fax>
900   <email/>
901 </insurer>
902 <medical>
903   <treatment>
904     <drug>
905       <name>methylphenidate hydrochloride</name>
906       <dailyDosage>30mgs</dailyDosage>
907       <startDate>1999-01-12</startDate>
908     </drug>
909     <comment>patient exhibits side-effects of skin coloration and carpal
910 degeneration</comment>
911   </treatment>
912   <result>
913     <test>blood pressure</test>
914     <value>120/80</value>
915     <date>2001-06-09</date>
916     <performedBy>Nurse Betty</performedBy>
917   </result>
918 </medical>
919 </record>

```

## 920 4.2.2 Example request context

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

```

924 [01] <?xml version="1.0" encoding="UTF-8"?>
925 [02] <Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
926 [03] xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy"
927 [04] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
928 [05] <Subject>
929 [06]   <Attribute AttributeId=
930 [07]     "urn:oasis:names:tc:xacml:1.0:subject:subject-category"
931 [08]     DataType="http://www.w3.org/2001/XMLSchema#string"
932 [09]     Issuer="www.medico.com"
933 [10]     IssueInstant="2001-12-17T09:30:47-05:00">
934 [11]   <AttributeValue>
935 [12]     urn:oasis:names:tc:xacml:1.0:subject:category:access-subject
936 [13]   </AttributeValue>
937 [14] </Attribute>
938 [15] <Attribute AttributeId=
939 [16]   "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
940 [17]   DataType=
941 [18]   "urn:oasis:names:tc:xacml:1.0:data-type:x500name"
942 [19]   Issuer="www.medico.com"
943 [20]   IssueInstant="2001-12-17T09:30:47-05:00">

```

```

944 [21] <AttributeValue>CNeTulius Hibbert</AttributeValue>
945 [22] </Attribute>
946 [23] <Attribute AttributeId=
947 [24]     "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
948 [25]     DataType="http://www.w3.org/2001/XMLSchema#string"
949 [26]     Issuer="www.medico.com"
950 [27]     IssueInstant="2001-12-17T09:30:47-05:00">
951 [28] <AttributeValue>physician</AttributeValue>
952 [29] </Attribute>
953 [30] <Attribute AttributeId=
954 [31]     "urn:oasis:names:tc:xacml:1.0:example:attribute:physician-id"
955 [32]     DataType="http://www.w3.org/2001/XMLSchema#string"
956 [33]     Issuer="www.medico.com"
957 [34]     IssueInstant="2001-12-17T09:30:47-05:00">
958 [35] <AttributeValue>jh1234</AttributeValue>
959 [36] </Attribute>
960 [37] </Subject>
961 [38] <Resource>
962 [39] <ResourceContent>
963 [40] <md:record
964 [41]     xmlns:md="//http:www.medico.com/schemas/record.xsd">
965 [42] <md:patient>
966 [43]     <md:patientDoB>1992-03-21</md:patientDoB>
967 [44] </md:patient>
968 [45] <!-- other fields -->
969 [46] </md:record>
970 [47] </ResourceContent>
971 [48] <Attribute AttributeId=
972 [49]     "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
973 [50]     DataType="http://www.w3.org/2001/XMLSchema#string">
974 [51] <AttributeValue>
975 [52]     //medico.com/records/bart-simpson.xml#
976 [53]     xmlns(md="//http:www.medico.com/schemas/record.xsd)
977 [54]     xpointer(/md:record/md:patient/md:patientDoB)
978 [55] </AttributeValue>
979 [56] </Attribute>
980 [57] <Attribute AttributeId=
981 [58]     "urn:oasis:names:tc:xacml:1.0:resource:xpath"
982 [59]     DataType="http://www.w3.org/2001/XMLSchema#string">
983 [60] <AttributeValue>
984 [61]     xmlns(md=http:www.medico.com/schemas/record.xsd)
985 [62]     xpointer(/md:record/md:patient/md:patientDoB)
986 [63] </AttributeValue>
987 [64] </Attribute>
988 [65] <Attribute AttributeId=
989 [66]     "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
990 [67]     DataType="http://www.w3.org/2001/XMLSchema#string">
991 [68] <AttributeValue>
992 [69]     http://www.medico.com/schemas/record.xsd
993 [70] </AttributeValue>
994 [71] </Attribute>
995 [72] </Resource>
996 [73] <Action>
997 [74] <Attribute AttributeId=
998 [75]     "urn:oasis:names:tc:xacml:1.0:action:action-id"
999 [76]     DataType="http://www.w3.org/2001/XMLSchema#string">
1000 [77] <AttributeValue>read</AttributeValue>
1001 [78] </Attribute>
1002 [79] </Action>
1003 [80] </Request>

```

1004 [02]-[04] Standard namespace declarations.

1005 [05]-[37] **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 [06]-[14] Each `Subject` section must have one and only one subject-category **attribute**. The  
 1008 value of this **attribute** describes the role that the **subject** plays in making the **decision request**.  
 1009 The value of “`access-subject`” denotes the identity for which the request was issued.

1010 [15]-[22] **Subject** `subject-id` **attribute**.

1011 [23]-[29] **Subject** `role` **attribute**.

1012 [30]-[36] **Subject** `physician-id` **attribute**.

1013 [38]-[69] **Resource** attributes are placed in the `Resource` section of the `Request`. Each **attribute**  
 1014 consists of **attribute** meta-data and an **attribute** value.

1015 [39]-[47] **Resource** content. The XML document that is being requested is placed here.

1016 [48]-[60] **Resource** identifier.

1017 [56]-[58] 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 [61]-[68] The XPath location path in the “`resource-id`” attribute is extracted and placed in the  
 1021 `xpath` attribute.

1022 [69]-[75] **Resource** `target-namespace` **attribute**.

1023 [77]-[84] **Action** **attributes** are placed in the `Action` section of the `Request`.

1024 [78]-[82] **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 may read any record for which he or she is the designated patient.

1028 Rule 2: A person may read any record for which he or she is the designated parent or  
 1029 guardian, and for which the patient is under 16 years of age.

1030 Rule 3: A physician may write to any medical element for which he or she is the designated  
 1031 primary care physician, provided an email is sent to the patient.

1032 Rule 4: An administrator shall not be permitted to read or write to medical elements of a  
 1033 patient record.

1034 These **rules** may be written by different **PAPs** operating independently, or by a single **PAP**.

### 1035 4.2.4 Example XACML rule instances

#### 1036 4.2.4.1. Rule 1

1037 Rule 1 illustrates a simple **rule** with a single `<Condition>` element. The following XACML  
 1038 `<Rule>` instance expresses Rule 1:

```
1039 [01] <?xml version="1.0" encoding="UTF-8"?>
1040 [02] <Rule
```

```

1041 [03] xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1042 [04] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1043 [05] xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1044 [06] xmlns:md="http://www.medico.com/schemas/record.xsd"
1045 [07] RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"
1046 [08] Effect="Permit">
1047 [09] <Description>
1048 [10] A person may read any medical record in the
1049 [11] http://www.medico.com/schemas/record.xsd namespace
1050 [12] for which he or she is a designated patient
1051 [13] </Description>
1052 [14] <Target>
1053 [15] <Subjects>
1054 [16] <AnySubject/>
1055 [17] </Subjects>
1056 [18] <Resources>
1057 [20] <Resource>
1058 [21] <!-- match document target namespace -->
1059 [22] <ResourceMatch
1060 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1061 [23] <ResourceAttributeDesignator AttributeId=
1062 [24] "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1063 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1064 [25] <AttributeValue
1065 DataType="http://www.w3.org/2001/XMLSchema#string">
1066 [26] http://www.medico.com/schemas/record.xsd
1067 [27] </AttributeValue>
1068 [28] </ResourceMatch>
1069 [29] <!-- match requested xml element -->
1070 [30] <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-
1071 node-match">
1072 [31] <ResourceAttributeDesignator AttributeId=
1073 [32] "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1074 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1075 [33] <AttributeValue
1076 DataType="http://www.w3.org/2001/XMLSchema#string">/md:record</AttributeValue>
1077 [34] </ResourceMatch>
1078 [35] </Resource>
1079 [36] </Resources>
1080 [37] <Actions>
1081 [38] <Action>
1082 [39] <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1083 equal">
1084 [40] <ActionAttributeDesignator AttributeId=
1085 [41] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1086 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1087 [42] <AttributeValue
1088 DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1089 [43] </ActionMatch>
1090 [44] </Action>
1091 [45] </Actions>
1092 [46] </Target>
1093 [47] <!-- compare policy number in the document with
1094 [48] policy-number attribute -->
1095 [49] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1096 [50] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-
1097 only">
1098 [51] <!-- policy-number attribute -->
1099 [52] <SubjectAttributeDesignator AttributeId=
1100 [53] "urn:oasis:names:tc:xacml:1.0:examples:attribute:policy-number"
1101 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1102 [54] </Apply>

```

```

1103 [55] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-
1104 only">
1105 [56] <!-- policy number in the document -->
1106 [57] <AttributeSelector RequestContextPath=
1107 [58] " //md:record/md:patient/md:policyNumber"
1108 DataType="http://www.w3.org/2001/XMLSchema#string">
1109 [59] </AttributeSelector>
1110 [60] </Apply>
1111 [61] </Condition>
1112 [62] </Rule>

```

1113 [02]-[06]. XML namespace declarations.

1114 [07] **Rule** identifier.

1115 [08]. When a **rule** evaluates to 'True' it emits the value of the `Effect` attribute. This value is  
1116 combined with the `Effect` values of other rules according to the **rule-combining algorithm**.

1117 [09]-[13] Free form description of the **rule**.

1118 [14]-[46]. A **rule target** defines a set of **decision requests** that are applicable to the **rule**. A  
1119 decision request, such that the value of the  
1120 "urn:oasis:names:tc:xacml:1.0:resource:target-namespace" **resource attribute** is  
1121 equal to "http://www.medico.com/schema/records.xsd" and the value of the  
1122 "urn:oasis:names:tc:xacml:1.0:resource:xpath" **resource attribute** matches the XPath  
1123 expression /md:record and the value of the  
1124 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** is equal to "read",  
1125 matches the **target** of this **rule**.

1126 [15]-[17]. The `Subjects` element may contain either a **disjunctive sequence** of `Subject`  
1127 elements or `AnySubject` element.

1128 [16] The `AnySubject` element is a special element that matches any **subject** in the request  
1129 **context**.

1130 [18]-[36]. The `Resources` element may contain either a **disjunctive sequence** of `Resource`  
1131 elements or `AnyResource` element.

1132 [20]-[35] The `Resource` element encloses the **conjunctive sequence** of `ResourceMatch`  
1133 elements.

1134 [22]-[28] The `ResourceMatch` element compares its first and second child elements according to  
1135 the matching function. A match is positive if any of the values selected by the first argument match  
1136 the explicit value of the second argument. This match compares the target namespace of the  
1137 requested document with the value of "http://www.medico.com/schema.records.xsd".

1138 [22] The `MatchId` attribute names the matching function.

1139 [23]-[24] The `ResourceAttributeDesignator` element selects the **resource attribute** values  
1140 from the request **context**. The **attribute** name is specified by the `AttributeId`. The selection  
1141 result is a **bag** of values.

1142 [25]-[27] Literal attribute value to match.

1143 [30]-[34] The `ResourceMatch`. This match compares the results of two XPath expressions. The  
1144 first XPath expression is the location path to the requested xml element and the second XPath  
1145 expression is /md:record. The "xpath-node-match" function evaluates to "True" if the requested  
1146 XML element is below the /md:record element.

1147 [30] `MatchId` attribute names the matching function.

1148 [31]-[32] The ResourceAttributeDesignator selects the **bag** of values for the  
1149 “urn:oasis:names:tc:xacml:1.0:xpath” **resource attribute**. Here, there is just one  
1150 element in the **bag**, which is the location path for the requested XML element.

1151 [33] The literal XPath expression to match. The md prefix is resolved using a standard namespace  
1152 declaration.

1153 [37]-[45] The Actions element may contain either a **disjunctive sequence** of Action elements  
1154 or an AnyAction element.

1155 [38]-[44] The Action element contains a **conjunctive sequence** of ActionMatch elements.

1156 [39]-[43] The ActionMatch element compares its first and second child elements according to the  
1157 matching function. Match is positive, if any of the values selected by the first argument match  
1158 explicit value of the second argument. In this case, the value of the action-id action attribute in  
1159 the request **context** is compared with the value “read”.

1160 [39] The MatchId attribute names the matching function.

1161 [40]-[41] The ActionAttributeDesignator selects **action attribute** values from the request  
1162 **context**. The **attribute** name is specified by the AttributeId. The selection result is a **bag** of  
1163 values. “urn:oasis:names:tc:xacml:1.0:action:action-id” is the predefined name for  
1164 the action identifier.

1165 [42] The **Attribute** value to match. This is an **action** name.

1166 [49]-[61] The Condition element. A **condition** must evaluate to “True” for the **rule** to be  
1167 applicable. This condition evaluates the truth of the statement: the policy-number **subject**  
1168 **attribute** is equal to the policy number in the XML document.

1169 [49] The FunctionId attribute of the Condition element names the function to be used for  
1170 comparison. In this case, comparison is done with function:string-equal; this function takes  
1171 two arguments of the “http://www.w3.org/2001/XMLSchema#string” type.

1172 [50] The first argument to the function:string-equal in the Condition. Functions can take  
1173 other functions as arguments. The Apply element encodes the function call with the FunctionId  
1174 attribute naming the function. Since function:string-equal takes arguments of the  
1175 “http://www.w3.org/2001/XMLSchema#string” type and  
1176 SubjectAttributeDesignator selects a **bag** of  
1177 “http://www.w3.org/2001/XMLSchema#string” values, “function:string-one-and-  
1178 only” is used. This function guarantees that its argument evaluates to a **bag** containing one and  
1179 only one “http://www.w3.org/2001/XMLSchema#string” element.

1180 [52]-[53] The SubjectAttributeDesignator selects a **bag** of values for the policy-number  
1181 **subject attribute** in the request **context**.

1182 [55] The second argument to the “function:string-equal” in the Condition. Functions can  
1183 take other functions as arguments. The Apply element encodes function call with the  
1184 FunctionId attribute naming the function. Since “function:string-equal” takes arguments  
1185 of the “http://www.w3.org/2001/XMLSchema#string” type and the AttributeSelector  
1186 selects a **bag** of “http://www.w3.org/2001/XMLSchema#string” values,  
1187 “function:string-one-and-only” is used. This function guarantees that its argument  
1188 evaluates to a **bag** containing one and only one  
1189 “http://www.w3.org/2001/XMLSchema#string” element.

1190 [57] The AttributeSelector element selects a **bag** of values from the request **context**. The  
1191 AttributeSelector is a free-form XPath pointing device into the request **context**. The

1192 RequestContextPath attribute specifies an XPath expression over the content of the requested  
1193 XML document, selecting the policy number. Note that the namespace prefixes in the XPath  
1194 expression are resolved with the standard XML namespace declarations.

#### 1195 4.2.4.2. Rule 2

1196 Rule 2 illustrates the use of a mathematical function, i.e. the <Apply> element with functionId  
1197 "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" to calculate date. It also  
1198 illustrates the use of **predicate** expressions, with the functionId  
1199 "urn:oasis:names:tc:xacml:1.0:function:and".

```
1200 [01] <?xml version="1.0" encoding="UTF-8"?>
1201 [02] <Rule
1202 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1203 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1204 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1205 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1206 [07]   RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
1207 [08]   Effect="Permit">
1208 [09]   <Description>
1209 [10]     A person may read any medical record in the
1210 [11]     http://www.medico.com/records.xsd namespace
1211 [12]     for which he or she is the designated parent or guardian,
1212 [13]     and for which the patient is under 16 years of age
1213 [14]   </Description>
1214 [15]   <Target>
1215 [16]     <Subjects>
1216 [17]       <AnySubject/>
1217 [18]     </Subjects>
1218 [19]     <Resources>
1219 [20]       <Resource>
1220 [21]         <!-- match document target namespace -->
1221 [22]         <ResourceMatch
1222 [23]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1223 [24]             <ResourceAttributeDesignator AttributeId=
1224 [25]               "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1225 [26]               DataType="http://www.w3.org/2001/XMLSchema#string"/>
1226 [27]             <AttributeValue
1227 [28]               DataType="http://www.w3.org/2001/XMLSchema#string">
1228 [29]                 http://www.medico.com/schemas/record.xsd
1229 [30]             </AttributeValue>
1230 [31]           </ResourceMatch>
1231 [32]           <!-- match requested xml element -->
1232 [33]           <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-
1233 [34]             node-match">
1234 [35]             <ResourceAttributeDesignator AttributeId=
1235 [36]               "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1236 [37]               DataType="http://www.w3.org/2001/XMLSchema#string"/>
1237 [38]             <AttributeValue
1238 [39]               DataType="http://www.w3.org/2001/XMLSchema#string">/md:record</AttributeValue>
1239 [40]             </ResourceMatch>
1240 [41]           </Resource>
1241 [42]         </Resources>
1242 [43]       <Actions>
1243 [44]         <Action>
1244 [45]           <!-- match 'read' action -->
1245 [46]           <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1246 [47]             equal">
1247 [48]             <ActionAttributeDesignator AttributeId=
1248 [49]               "urn:oasis:names:tc:xacml:1.0:action:action-id"
1249 [50]               DataType="http://www.w3.org/2001/XMLSchema#string"/>
```

```

1250 [43]         <AttributeValue
1251 DataTypeId="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1252 [44]         </ActionMatch>
1253 [45]     </Action>
1254 [46] </Actions>
1255 [47] </Target>
1256 [48] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
1257 [49]     <!-- compare parent-guardian-id subject attribute with
1258 [50]         the value in the document -->
1259 [51]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1260 [52]         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1261 and-only">
1262 [53]             <!-- parent-guardian-id subject attribute -->
1263 [54]             <SubjectAttributeDesignator AttributeId=
1264 [55]                 "urn:oasis:names:tc:xacml:1.0:examples:attribute:
1265 [56]                 parent-guardian-id"
1266 DataTypeId="http://www.w3.org/2001/XMLSchema#string"/>
1267 [57]             </Apply>
1268 [58]         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1269 and-only">
1270 [59]             <!-- parent-guardian-id element in the document -->
1271 [60]             <AttributeSelector RequestContextPath=
1272 [61]                 "//md:record/md:parentGuardian/md:parentGuardianId"
1273 [62]                 DataTypeId="http://www.w3.org/2001/XMLSchema#string">
1274 [63]             </AttributeSelector>
1275 [64]             </Apply>
1276 [65]         </Apply>
1277 [66]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-or-
1278 equal">
1279 [67]         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-
1280 only">
1281 [68]             <EnvironmentAttributeDesignator AttributeId=
1282 [69]                 "urn:oasis:names:tc:xacml:1.0:environment:current-date"
1283 DataTypeId="http://www.w3.org/2001/XMLSchema#date"/>
1284 [70]             </Apply>
1285 [71]         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-
1286 yearMonthDuration">
1287 [73]             <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-
1288 and-only">
1289 [74]                 <!-- patient dob recorded in the document -->
1290 [75]                 <AttributeSelector RequestContextPath=
1291 [76]                     "//md:record/md:patient/md:patientDoB"
1292 DataTypeId="http://www.w3.org/2001/XMLSchema#date">
1293 [77]                 </AttributeSelector>
1294 [78]                 </Apply>
1295 [79]                 <AttributeValue DataTypeId="xf:yearMonthDuration">
1296 [80]                     P16Y
1297 [81]                 </AttributeValue>
1298 [82]                 </Apply>
1299 [83]             </Apply>
1300 [84]         </Condition>
1301 [85] </Rule>

```

1302 [02]-[47] **Rule** declaration and **rule target**. See Rule 1 in section 4.2.4.1 for the detailed  
1303 explanation of these elements.

1304 [48]-[82] The **Condition** element. **Condition** must evaluate to “True” for the **rule** to be applicable.  
1305 This **condition** evaluates the truth of the statement: the requestor is the designated parent or  
1306 guardian and the patient is under 16 years of age.

1307 [48] The **Condition** is using the “function:and” function. This is a boolean function that takes  
1308 one or more boolean arguments (2 in this case) and performs the logical “AND” operation to  
1309 compute the truth value of the expression.

1310 [51]-[65] The truth of the first part of the condition is evaluated: The requestor is the designated  
1311 parent or guardian. The `Apply` element contains a function invocation. The function name is  
1312 contained in the `FunctionId` attribute. The comparison is done with “`function:string-`  
1313 `equal`” that takes 2 arguments of “`http://www.w3.org/2001/XMLSchema#string`” type.

1314 [52] Since “`function:string-equal`” takes arguments of the  
1315 “`http://www.w3.org/2001/XMLSchema#string`” type, “`function:string-one-and-`  
1316 `only`” is used to ensure that the **subject attribute**  
1317 “`urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id`” in the request **context**  
1318 contains one and only one value. “`Function:string-equal`” takes an argument expression  
1319 that evaluates to a **bag** of “`http://www.w3.org/2001/XMLSchema#string`” values.

1320 [54] Value of the **subject attribute**  
1321 “`urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id`” is  
1322 selected from the request **context** with the `SubjectAttributeDesignator` element. This  
1323 expression evaluates to a bag of “`http://www.w3.org/2001/XMLSchema#string`” values.

1324 [58] “`function:string-one-and-only`” is used to ensure that the **bag** of values selected by  
1325 it’s argument contains one and only one value of type  
1326 “`http://www.w3.org/2001/XMLSchema#string`”.

1327 [60] The value of the `md:parentGuardianId` element is selected from the **resource** content with  
1328 the `AttributeSelector` element. `AttributeSelector` is a free-form XPath expression,  
1329 pointing into the request **context**. The `RequestContextPath` XML attribute contains an XPath  
1330 expression over the request **context**. Note that all namespace prefixes in the XPath expression  
1331 are resolved with standard namespace declarations. The `AttributeSelector` evaluates to the  
1332 **bag** of values of type “`http://www.w3.org/2001/XMLSchema#string`”.

1333 [66]-[83] The expression: “the patient is under 16 years of age” is evaluated. The patient is under  
1334 16 years of age if the current date is less than the date computed by adding 16 to the patient’s date  
1335 of birth.

1336 [66] “`function:date-less-or-equal`” is used to compute the difference of two dates.

1337 [67] “`function:date-one-and-only`” is used to ensure that the **bag** of values selected by its  
1338 argument contains one and only one value of type  
1339 “`http://www.w3.org/2001/XMLSchema#date`”.

1340 [68]-[69] Current date is evaluated by selecting the  
1341 “`urn:oasis:names:tc:xacml:1.0:environment:current-date`” **environment attribute**.

1342 [71] “`function:date-add-yearMonthDuration`” is used to compute the date by adding 16 to  
1343 the patient’s date of birth. The first argument is a  
1344 “`http://www.w3.org/2001/XMLSchema#date`”, and the second argument is an  
1345 “`xf:yearMonthDuration`”.

1346 [73] “`function:date-one-and-only`” is used to ensure that the **bag** of values selected by it’s  
1347 argument contains one and only one value of type  
1348 “`http://www.w3.org/2001/XMLSchema#date`”.

1349 [75]-[76] The `<AttributeSelector>` element selects the patient’s date of birth by taking the  
1350 XPath expression over the document content.

1351 [79]-[81] Year Month Duration of 16 years.

1352

### 4.2.4.3. Rule 3

1353 Rule 3 illustrates the use of an **obligation**. The XACML <Rule> element syntax does not include  
 1354 an element suitable for carrying an **obligation**, therefore Rule 3 has to be formatted as a  
 1355 <Policy> element.

```

1356 [01] <?xml version="1.0" encoding="UTF-8"?>
1357 [02] <Policy
1358 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1359 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1360 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1361 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1362 [07]   PolicyId="urn:oasis:names:tc:xacml:examples:policyid:3"
1363 [08]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
1364 [09]     rule-combining-algorithm:deny-overrides">
1365 [10]   <Description>
1366 [11]     Policy for any medical record in the
1367 [12]     http://www.medico.com/schemas/record.xsd namespace
1368 [13]   </Description>
1369 [14]   <Target>
1370 [15]     <Subjects>
1371 [16]       <AnySubject/>
1372 [17]     </Subjects>
1373 [18]     <Resources>
1374 [19]       <Resource>
1375 [20]         <!-- match document target namespace -->
1376 [21]         <ResourceMatch
1377 [22]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1378 [23]             <ResourceAttributeDesignator AttributeId=
1379 [24]               "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1380 [25]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1381 [26]             <AttributeValue
1382 [27]               DataType="http://www.w3.org/2001/XMLSchema#string">
1383 [28]               http://www.medico.com/schemas/record.xsd
1384 [29]             </AttributeValue>
1385 [30]           </ResourceMatch>
1386 [31]         </Resource>
1387 [32]       </Resources>
1388 [33]     <Actions>
1389 [34]       <AnyAction/>
1390 [35]     </Actions>
1391 [36]   </Target>
1392 [37] <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:3"
1393 [38]   Effect="Permit">
1394 [39]   <Description>
1395 [40]     A physician may write any medical element in a record
1396 [41]     for which he or she is the designated primary care
1397 [42]     physician, provided an email is sent to the patient
1398 [43]   </Description>
1399 [44]   <Target>
1400 [45]   <Subjects>
1401 [46]     <Subject>
1402 [47]       <!-- match subject group attribute -->
1403 [48]       <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1404 [49]         equal">
1405 [50]         <SubjectAttributeDesignator AttributeId=
1406 [51]           "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
1407 [52]         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1408 [53]         <AttributeValue
1409 [54]           DataType="http://www.w3.org/2001/XMLSchema#string">physician</AttributeValue>
1410 [55]         </SubjectMatch>
1411 [56]       </Subject>

```

```

1412 [ 51] </Subjects>
1413 [ 52] <Resources>
1414 [ 53]   <Resource>
1415 [ 54]     <!-- match requested xml element -->
1416 [ 55]     <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-
1417 node-match">
1418 [ 56]       <ResourceAttributeDesignator AttributeId=
1419 [ 57]         "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1420 [ 58]       DataType="http://www.w3.org/2001/XMLSchema#string"/>
1421 [ 59]       <AttributeValue
1422 [ 60]         DataType="http://www.w3.org/2001/XMLSchema#string">
1423 [ 61]         /md:record/md:medical
1424 [ 62]       </AttributeValue>
1425 [ 63]     </ResourceMatch>
1426 [ 64]   </Resource>
1427 [ 65] </Resources>
1428 [ 66] <Actions>
1429 [ 67]   <Action>
1430 [ 68]     <!-- match action -->
1431 [ 69]     <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1432 equal">
1433 [ 70]       <ActionAttributeDesignator AttributeId=
1434 [ 71]         "urn:oasis:names:tc:xacml:1.0:action:action-id"
1435 [ 72]       DataType="http://www.w3.org/2001/XMLSchema#string"/>
1436 [ 73]       <AttributeValue
1437 [ 74]         DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
1438 [ 75]       </ActionMatch>
1439 [ 76]     </Action>
1440 [ 77]   </Actions>
1441 [ 78] </Target>
1442 [ 79] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1443 equal">
1444 [ 80]   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1445 and-only">
1446 [ 81]     <!-- physician-id subject attribute -->
1447 [ 82]     <SubjectAttributeDesignator AttributeId=
1448 [ 83]       "urn:oasis:names:tc:xacml:1.0:example:
1449 [ 84]       attribute:physician-id"
1450 [ 85]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1451 [ 86]     </Apply>
1452 [ 87]   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1453 and-only">
1454 [ 88]     <AttributeSelector RequestContextPath=
1455 [ 89]       "/md:record/md:primaryCarePhysician/md:registrationID"
1456 [ 90]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1457 [ 91]     </Apply>
1458 [ 92]   </Condition>
1459 [ 93] </Rule>
1460 [ 94] <Obligations>
1461 [ 95]   <!-- send e-mail message to the document owner -->
1462 [ 96]   <Obligation ObligationId=
1463 [ 97]     "urn:oasis:names:tc:xacml:example:obligation:email"
1464 [ 98]   FulfillOn="Permit">
1465 [ 99]     <AttributeAssignment AttributeId=
1466 [ 100]       "urn:oasis:names:tc:xacml:1.0:example:attribute:mailto"
1467 [ 101]     DataType="http://www.w3.org/2001/XMLSchema#string">
1468 [ 102]     <AttributeSelector RequestContextPath=
1469 [ 103]       "/md:/record/md:patient/md:patientContact/md:email"
1470 [ 104]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1471 [ 105]     </AttributeAssignment>
1472 [ 106]     <AttributeAssignment AttributeId=
1473 [ 107]       "urn:oasis:names:tc:xacml:1.0:example:attribute:text"
1474 [ 108]     DataType="http://www.w3.org/2001/XMLSchema#string">

```

```

1475 [105] <AttributeValue>
1476 [106]     Your medical record has been accessed by:
1477 [107] </AttributeValue>
1478 [108] </AttributeAssignment>
1479 [109] <AttributeAssignment AttributeId=
1480 [110]     "urn:oasis:names:tc:xacml:example:attribute:text"
1481 [111]     DataType="http://www.w3.org/2001/XMLSchema#string">
1482 [112]     <SubjectAttributeDesignator AttributeId=
1483 [113]     "urn:osasis:names:tc:xacml:1.0:subject:subject-id"
1484 [114]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1485 [114]     </AttributeAssignment>
1486 [115] </Obligation>
1487 [116] </Obligations>
1488 [117] </Policy>

```

1489 [01]-[09] The `Policy` element includes standard namespace declarations as well as policy specific  
1490 parameters, such as `PolicyId` and `RuleCombiningAlgId`.

1491 [07] **Policy** identifier. This parameter is used for the inclusion of the `Policy` in the `PolicySet`  
1492 element.

1493 [08]-[09] **Rule combining algorithm** identifier. This parameter is used to compute the combined  
1494 outcome of **rule effects** for **rules** that are applicable to the **decision request**.

1495 [10-13] Free-form description of the **policy**.

1496 [14]-[33] **Policy target**. The **policy target** defines a set of applicable decision requests. The  
1497 structure of the `Target` element in the `Policy` is identical to the structure of the `Target` element  
1498 in the `Rule`. In this case, the **policy target** is a set of all XML documents conforming to the  
1499 "<http://www.medico.com/schemas/record.xsd>" target namespace. For the detailed description of  
1500 the `Target` element see Rule 1, section 4.2.4.1.

1501 [34]-[89] The only `Rule` element included in this `Policy`. Two parameters are specified in the **rule**  
1502 header: `RuleId` and `Effect`. For the detailed description of the `Rule` structure see Rule 1,  
1503 section 4.2.4.1.

1504 [41]-[74] A **rule target** narrows down a **policy target**. **Decision requests** with the value of  
1505 "urn:oasis:names:tc:xacml:1.0:example:attribute:role" **subject attribute** equal to  
1506 "physician" [42]-[51], and that access elements of the medical record that "xpath-node-match"  
1507 the "/md:record/md:medical" XPath expression [52]-[63], and that have the value of the  
1508 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** equal to "read".

1509 [65]-[73] match the **target** of this **rule**. For a detailed description of the rule target see example 1,  
1510 section 4.2.4.1.

1511 [75]-[87] The `Condition` element. For the **rule** to be applicable to the authorization request,  
1512 **condition** must evaluate to True. This **rule condition** compares the value of the  
1513 "urn:oasis:names:tc:xacml:1.0:examples:attribute:physician-id" **subject**  
1514 **attribute** with the value of the `physician id` element in the medical record that is being  
1515 accessed. For a detailed explanation of rule condition see Rule 1, section 4.2.4.1.

1516 [90]-[116] The `Obligations` element. **Obligations** are a set of operations that must be  
1517 performed by the **PEP** in conjunction with an **authorization decision**. An **obligation** may be  
1518 associated with a positive or negative **authorization decision**.

1519 [92]-[115] The `Obligation` element consists of the `ObligationId`, the authorization decision  
1520 value for which it must fulfill, and a set of attribute assignments.

1521 [92]-[93] `ObligationId` identifies an **obligation**. **Obligation** names are not interpreted by the  
1522 **PDP**.

1523 [94] FulfillOn attribute defines an **authorization decision** value for which this **obligation** must  
1524 be fulfilled.

1525 [95]-[101] **Obligation** may have one or more parameters. The **obligation** parameter  
1526 "urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto" is assigned the value  
1527 from the content of the xml document.

1528 [95-96] AttributeId declares  
1529 "urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto" **obligation** parameter.

1530 [97] The **obligation** parameter data type is defined.

1531 [98]-[100] The **obligation** parameter value is selected from the content of the XML document that is  
1532 being accessed with the XPath expression over request **context**.

1533 [102]-[108] The **obligation** parameter  
1534 "urn:oasis:names:tc:xacml:1.0:examples:attribute:text" of type  
1535 "http://www.w3.org/2001/XMLSchema#string" is assigned the literal value "Your  
1536 medical record has been accessed by:"

1537 [109]-[114] The **obligation** parameter  
1538 "urn:oasis:names:tc:xacml:1.0:examples:attribute:text" of the  
1539 "http://www.w3.org/2001/XMLSchema#string" data type is assigned the value of the  
1540 "urn:oasis:names:tc:xacml:1.0:subject:subject-id" **subject attribute**.

#### 4.2.4.4. Rule 4

1541 Rule 4 illustrates the use of the "Deny" Effect value, and a Rule with no Condition element.

```

1543 [01] <?xml version="1.0" encoding="UTF-8"?>
1544 [02] <Rule
1545 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1546 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1547 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1548 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1549 [07]   RuleId="urn:oasis:names:tc:xacml:example:ruleid:4"
1550 [08]   Effect="Deny">
1551 [09]   <Description>
1552 [10]     An Administrator shall not be permitted to read or write
1553 [11]     medical elements of a patient record in the
1554 [12]     http://www.medico.com/records.xsd namespace.
1555 [13]   </Description>
1556 [14]   <Target>
1557 [15]     <Subjects>
1558 [16]       <Subject>
1559 [17]         <!-- match role subject attribute -->
1560 [18]         <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1561 [19] equal">
1562 [20]           <SubjectAttributeDesignator AttributeId=
1563 [21] "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
1564 [22] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1565 [23]           <AttributeValue
1566 [24] DataType="http://www.w3.org/2001/XMLSchema#string">administrator</AttributeValue>
1567 [25]         </SubjectMatch>
1568 [26]       </Subject>
1569 [27]     </Subjects>
1570 [28]   <Resources>
1571 [29]     <Resource>
1572 [30]       <!-- match document target namespace -->

```

```

1573 [28] <ResourceMatch
1574 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1575 [29] <ResourceAttributeDesignator AttributeId=
1576 "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1577 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1578 [31] <AttributeValue
1579 DataType="http://www.w3.org/2001/XMLSchema#string">
1580 [32] http://www.medico.com/schemas/record.xsd
1581 [33] </AttributeValue>
1582 [34] </ResourceMatch>
1583 [35] <!-- match requested xml element -->
1584 [36] <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-
1585 node-match">
1586 [37] <ResourceAttributeDesignator AttributeId=
1587 [38] "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1588 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1589 [39] <AttributeValue
1590 DataType="http://www.w3.org/2001/XMLSchema#string">
1591 [40] /md:record/md:medical
1592 [41] </AttributeValue>
1593 [42] </ResourceMatch>
1594 [43] </Resource>
1595 [44] </Resources>
1596 [45] <Actions>
1597 [46] <Action>
1598 [47] <!-- match 'read' action -->
1599 [48] <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1600 equal">
1601 [49] <ActionAttributeDesignator AttributeId=
1602 [50] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1603 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1604 [51] <AttributeValue
1605 DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1606 [52] </ActionMatch>
1607 [53] </Action>
1608 [54] <Action>
1609 [55] <!-- match 'write' action -->
1610 [56] <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1611 equal">
1612 [57] <ActionAttributeDesignator AttributeId=
1613 [58] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1614 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1615 [59] <AttributeValue
1616 DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
1617 [60] </ActionMatch>
1618 [61] </Action>
1619 [62] </Actions>
1620 [63] </Target>
1621 [64] </Rule>

```

1622 [01]-[08] The `Rule` element declaration. The most important parameter here is `Effect`. See Rule  
1623 1, section 4.2.4.1 for a detailed explanation of the `Rule` structure.

1624 [08] **Rule Effect**. Every *rule* that evaluates to “True” emits *rule effect* as its value that will be  
1625 combined later on with other *rule effects* according to the *rule combining algorithm*. This *rule*  
1626 `Effect` is “Deny” meaning that according to this rule, access must be denied.

1627 [09]-[13] Free form description of the *rule*.

1628 [14]-[63] **Rule target**. The *Rule target* defines a set of *decision requests* that are applicable to  
1629 the *rule*. This *rule* is matched by:

- 1630 • a **decision request** with **subject attribute**
- 1631 "urn:oasis:names:tc:xacml:1.0:examples:attribute:role" equal to
- 1632 "administrator";
- 1633 • the value of **resource attribute**
- 1634 "urn:oasis:names:tc:xacml:1.0:resource:target-namespace" is equal to
- 1635 "http://www.medico.com/schemas/record.xsd"
- 1636 • the value of the requested XML element matches the XPath expression
- 1637 "/md:record/md:medical";
- 1638 • the value of **action attribute** "urn:oasis:names:tc:xacml:1.0:action:action-id" is equal to
- 1639 "read"

1640 See Rule 1, section 4.2.4.1 for the detailed explanation of the Target element.

1641 This **rule** does not have a Condition element.

#### 1642 4.2.4.5. Example PolicySet

1643 This section uses the examples of the previous sections to illustrate the process of combining

1644 **policies**. The policy governing read access to medical elements of a record is formed from each of

1645 the four **rules described in Section 4.2.3**. In plain language, the combined rule is:

- 1646 • Either the requestor is the patient; or
- 1647 • the requestor is the parent or guardian and the patient is under 16; or
- 1648 • the requestor is the primary care physician and a notification is sent to the patient; and
- 1649 • the requestor is not an administrator.

1650 The following XACML <PolicySet> illustrates the combined **policies**. **Policy 3** is included by

1651 reference and **policy 2** is explicitly included.

```

1652 [01] <?xml version="1.0" encoding="UTF-8"?>
1653 [02] <PolicySet
1654 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1655 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1656 [05]   PolicySetId=
1657 [06]     "urn:oasis:names:tc:xacml:1.0:examples:policysetid:1"
1658 [07]   PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
1659 [071]     policy-combining-algorithm:deny-overrides"/>
1660 [08] <Description>
1661 [09]   Example policy set.
1662 [10] </Description>
1663 [11] <Target>
1664 [12]   <Subjects>
1665 [13]     <Subject>
1666 [14]       <!-- any subject -->
1667 [15]       <AnySubject/>
1668 [16]     </Subject>
1669 [17]   </Subjects>
1670 [18]   <Resources>
1671 [19]     <Resource>
1672 [20]       <!-- any resource in the target namespace -->
1673 [21]       <ResourceMatch
1674 [22] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1675 [22]   <ResourceAttributeDesignator AttributeId=
1676 [23]     "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1677 [23]   DataType="http://www.w3.org/2001/XMLSchema#string"/>

```

```

1678 [24]         <AttributeValue
1679 DataTyped="http://www.w3.org/2001/XMLSchema#string">
1680 [25]         http://www.medico.com/records.xsd
1681 [26]         </AttributeValue>
1682 [27]         </ResourceMatch>
1683 [28]         </Resource>
1684 [29]     </Resources>
1685 [30]     <Actions>
1686 [31]         <Action>
1687 [32]             <!-- any action -->
1688 [33]             <AnyAction/>
1689 [34]         </Action>
1690 [35]     </Actions>
1691 [36] </Target>
1692 [37] <!-- include policy from the example 3 by reference -->
1693 [38] <PolicyIdReference>
1694 [39]     urn:oasis:names:tc:xacml:1.0:examples:policyid:3
1695 [40] </PolicyIdReference>
1696 [41]     <!-- policy 2 combines rules from the examples 1, 2,
1697 [42]     and 4 is included by value. -->
1698 [43] <Policy
1699 [44]     PolicyId="urn:oasis:names:tc:xacml:examples:policyid:2"
1700 [45]     RuleCombiningAlgId=
1701 [46] "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
1702 [47]     <Description>
1703 [48]         Policy for any medical record in the
1704 [49]         http://www.medico.com/schemas/record.xsd namespace
1705 [50]     </Description>
1706 [51]     <Target> ... </Target>
1707 [52]     <Rule
1708 [53]         RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"
1709 [54]         Effect="Permit"> ... </Rule>
1710 [55]     <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
1711 [56]         Effect="Permit"> ... </Rule>
1712 [57]     <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:4"
1713 [58]         Effect="Deny"> ... </Rule>
1714 [59]     <Obligations> ... </Obligations>
1715 [60] </Policy>
1716 [61] </PolicySet>

```

1717

1718 [02]-[07] PolicySet declaration. Standard XML namespace declarations are included as well as  
1719 PolicySetId, and **policy combining algorithm** identifier.

1720 [05]-[06] PolicySetId is used for identifying this **policy set** and for possible inclusion of this  
1721 **policy set** into another **policy set**.

1722 [07] **Policy combining algorithm** identifier. Policies in the **policy set** are combined according to  
1723 the specified **policy combining algorithm** identifier when the **authorization decision** is  
1724 computed.

1725 [08]-[10] Free form description of the **policy set**.

1726 [11]-[36] PolicySet Target element defines a set of **decision requests** that are applicable to  
1727 this PolicySet.

1728 [38]-[40] PolicyIdReference includes **policy** by id.

1729 [43]-[60] **Policy 2** is explicitly included in this **policy set**.

---

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

### 5.1. Element <PolicySet>

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

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

**Policies** included in the <PolicySet> element MUST be combined by the algorithm specified by the PolicyCombiningAlgId attribute.

The <Target> element defines the applicability of the <PolicySet> to *decision requests*. If there is a match between the <Target> element within <PolicySet> and the *request context*, then the <PolicySet> element MAY be used by the *PDP* in making its *authorization decision*.

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

```
<xs:element name="PolicySet" type="xacml:PolicySetType"/>
<xs:complexType name="PolicySetType">
  <xs:sequence>
    <xs:element ref="xacml:Description" minOccurs="0"/>
    <xs:element ref="xacml:PolicySetDefaults" minOccurs="0"/>
    <xs:element ref="xacml:Target"/>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="xacml:PolicySet"/>
      <xs:element ref="xacml:Policy"/>
      <xs:element ref="xacml:PolicySetIdReference"/>
      <xs:element ref="xacml:PolicyIdReference"/>
    </xs:choice>
    <xs:element ref="xacml:Obligations" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="PolicySetId"
type="http://www.w3.org/2001/XMLSchema#anyURI" use="required"/>
  <xs:attribute name="PolicyCombiningAlgId"
type="http://www.w3.org/2001/XMLSchema#anyURI" use="required"/>
</xs:complexType>
```

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

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

PolicySetId [Required]

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

- 1775 PolicyCombiningAlgId [Required]
- 1776 The identifier of the **policy-combining algorithm** by which the <PolicySet>  
 1777 components MUST be combined. Standard **policy-combining algorithms** are listed in  
 1778 Appendix C. Standard **policy-combining algorithm** identifiers are listed in Section B.10.
- 1779 <Description> [Optional]
- 1780 A free-form description of the <PolicySet>.
- 1781 <PolicySetDefaults> [Optional]
- 1782 A set of default values applicable to the <PolicySet>. The scope of the  
 1783 <PolicyDefaults> element SHALL be the enclosing policy.
- 1784 <Target> [Required]
- 1785 The <Target> element defines the applicability of a <PolicySet> to **decision requests**.
- 1786 The <Target> element MAY be declared by the creator of the <PolicySet> or it MAY be  
 1787 computed from the <Target> elements of the referenced <Policy> elements, either as an  
 1788 intersection or as a union.
- 1789 <PolicySet> [Any Number]
- 1790 A **policy set** component that is included in this **policy set**.
- 1791 <Policy> [Any Number]
- 1792 A **policy** component that is included in this **policy set**.
- 1793 <PolicySetIdReference> [Any Number]
- 1794 A reference to a <PolicySet> component that MUST be included in this **policy set**. If  
 1795 <PolicySetIdReference> is a URL, then it MAY be resolvable.
- 1796 <PolicyIdReference> [Any Number]
- 1797 A reference to a <Policy> component that MUST be included in this **policy set**. If the  
 1798 <PolicyIdReference> is a URL, then it MAY be resolvable.
- 1799 <Obligations> [Optional]
- 1800 Contains the set of <Obligation> elements. See Section 7.11 for a description of how  
 1801 the set of **obligations** to be returned by the **PDP** shall be determined.

## 1802 5.2. Element <Description>

- 1803 The <Description> element is used for a free-form description of the <PolicySet> element  
 1804 and <Policy> element. The <Description> element is of **xs:string** simple type.

1805 

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

## 1806 5.3. Element <PolicySetDefaults>

- 1807 The <PolicySetDefaults> element SHALL specify default values that apply to the  
 1808 <PolicySet> element.

1809 

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

```

1810 <xs:complexType name="DefaultsType">
1811 <xs:sequence>
1812 <xs:choice>
1813 <xs:element ref="xacml:XPathVersion" minOccurs="0"/>
1814 </xs:choice>
1815 </xs:sequence>
1816 </xs:complexType>

```

1817 <PolicySetDefaults> element is of **DefaultsType** complex type.

1818 <XPathVersion> [Optional]

1819 Default XPath version.

## 1820 5.4. Element <XPathVersion>

1821 The <XPathVersion> element SHALL specify the version of the XPath specification to be used by  
 1822 <AttributeSelector> elements.

```

1823 <xs:element name="XPathVersion" type="xs:anyURI"/>

```

1824 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 policy contains  
 1826 <AttributeSelector> elements.

## 1827 5.5. Element <Target>

1828 The <Target> element identifies the set of **decision requests** that the parent element is intended  
 1829 to evaluate. The <Target> element SHALL appear as a child of <PolicySet>, <Policy> and  
 1830 <Rule> elements. It contains definitions for **subjects**, **resources** and **actions**.

1831 The <Target> element SHALL contain a **conjunctive sequence** of <Subjects>, <Resources>  
 1832 and <Actions> elements. For the parent of the <Target> element to be applicable to the  
 1833 **decision request**, there MUST be at least one positive match between each section of the  
 1834 <Target> element and the corresponding section of the <xacml-context:Request> element.

```

1835 <xs:element name="Target" type="xacml:TargetType"/>
1836 <xs:complexType name="TargetType">
1837 <xs:sequence>
1838 <xs:element ref="xacml:Subjects"/>
1839 <xs:element ref="xacml:Resources"/>
1840 <xs:element ref="xacml:Actions"/>
1841 </xs:sequence>
1842 </xs:complexType>

```

1843 The <Target> element is of **TargetType** complex type.

1844 <Subjects> [Required]

1845 Matching specification for the **subject attributes** in the **context**.

1846 <Resources> [Required]

1847 Matching specification for the **resource attributes** in the **context**.

1848 <Actions> [Required]

1849 Matching specification for the **action attributes** in the **context**.

## 1850 **5.6. Element <Subjects>**

1851 The <Subjects> element SHALL contain a *disjunctive sequence* of <Subject> elements.

```
1852 <xs:element name="Subjects" type="xacml:SubjectsType"/>
1853 <xs:complexType name="SubjectsType">
1854   <xs:choice>
1855     <xs:element ref="xacml:Subject" maxOccurs="unbounded"/>
1856     <xs:element ref="xacml:AnySubject"/>
1857   </xs:choice>
1858 </xs:complexType>
```

1859 The <Subjects> element is of **SubjectsType** complex type.

1860 <Subject> [One To Many, Required Choice]

1861 See section 5.7.

1862 <AnySubject> [Required Choice]

1863 See section 5.8.

## 1864 **5.7. Element <Subject>**

1865 The <Subject> element SHALL contain a *conjunctive sequence* of <SubjectMatch>  
1866 elements.

```
1867 <xs:element name="Subject" type="xacml:SubjectType"/>
1868 <xs:complexType name="SubjectType">
1869   <xs:sequence>
1870     <xs:element ref="xacml:SubjectMatch" maxOccurs="unbounded"/>
1871   </xs:sequence>
1872 </xs:complexType>
```

1873 The <Subject> element is of **SubjectType** complex type.

1874 <Subject> element contains the following elements:

1875 <SubjectMatch> [One to Many]

1876 A *conjunctive sequence* of individual matches of the *subject attributes* in the *context*  
1877 and the embedded *attribute* values.

## 1878 **5.8. Element <AnySubject>**

1879 The <AnySubject> element SHALL match any *subject attribute* in the *context*.

```
1880 <xs:element name="AnySubject"/>
```

## 1881 **5.9. Element <SubjectMatch>**

1882 The <SubjectMatch> element SHALL identify a set of *subject*-related entities by matching  
1883 *attribute* values in the <xacml-context:Subject> element of the *context* with the embedded  
1884 *attribute* value.

```
1885 <xs:element name="SubjectMatch" type="xacml:SubjectMatchType"/>
1886 <xs:complexType name="SubjectMatchType">
1887   <xs:sequence>
1888     <xs:choice>
```

```

1889     <xs:element ref="xacml:SubjectAttributeDesignator" />
1890     <xs:element ref="xacml:AttributeSelector" />
1891   </xs:choice>
1892   <xs:element ref="xacml:AttributeValue" />
1893 </xs:sequence>
1894 <xs:attribute name="MatchId" type="xs:anyURI" use="required" />
1895 </xs:complexType>

```

1896 The <SubjectMatch> element is of **SubjectMatchType** complex type.

1897 The <SubjectMatch> element contains the following attributes and elements:

1898 MatchId [Required]

1899 Specifies a matching function. The value of this attribute MUST be of type xs:anyURI with  
1900 legal values documented in Appendix A.

1901 <SubjectAttributeDesignator> [Required choice]

1902 Identifies one or more **attribute** values in the <xacml-context:Subject> child of the  
1903 <xacml-context:Request> element.

1904 <AttributeSelector> [Required choice]

1905 MAY be used to identify one or more **attribute** values in the <xacml-context:Subject>  
1906 child of the <xacml-context:Request> element.

1907 <AttributeValue> [Required]

1908 Embedded **attribute** value.

## 1909 **5.10. Element <Resources>**

1910 The <Resources> element SHALL contain a **disjunctive sequence** of <Resource> elements.

```

1911 <xs:element name="Resources" type="xacml:ResourcesType" />
1912 <xs:complexType name="ResourcesType">
1913   <xs:choice>
1914     <xs:element ref="xacml:Resource" maxOccurs="unbounded" />
1915     <xs:element ref="xacml:AnyResource" />
1916   </xs:choice>
1917 </xs:complexType>

```

1918 The <Resources> element is of **ResourcesType** complex type.

1919 The <Resources> element consists of the following elements:

1920 <Resource> [One To Many, Required Choice]

1921 See section 5.11.

1922 <AnyResource> [Required Choice]

1923 See section 5.12.

## 1924 **5.11. Element <Resource>**

1925 The <Resource> element SHALL contain a **conjunctive sequence** of <ResourceMatch>  
1926 elements.

```

1927 <xs:element name="Resource" type="xacml:ResourceType" />

```

1928  
1929  
1930  
1931  
1932

```
<xs:complexType name="ResourceType">
  <xs:sequence>
    <xs:element ref="xacml:ResourceMatch" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
```

1933 The <Resource> element is of **ResourceType** complex type.

1934 The <Resource> element contains the following elements:

1935 <ResourceMatch> [One to Many]

1936 A **conjunctive sequence** of individual matches of the **resource attributes** in the **context**  
1937 and the embedded **attribute** values.

## 1938 5.12. Element <AnyResource>

1939 The <AnyResource> element SHALL match any **resource attribute** in the **context**.

```
1940 <xs:element name="AnyResource" />
```

## 1941 5.13. Element <ResourceMatch>

1942 The <ResourceMatch> element SHALL identify a set of **resource**-related entities by matching  
1943 **attribute** values in the <xacml-context:Resource> element of the **context** with the embedded  
1944 **attribute** value.

```
1945 <xs:element name="ResourceMatch" type="xacml:ResourceMatchType" />
1946 <xs:complexType name="ResourceMatchType">
1947   <xs:sequence>
1948     <xs:choice>
1949       <xs:element ref="xacml:ResourceAttributeDesignator" />
1950       <xs:element ref="xacml:AttributeSelector" />
1951     </xs:choice>
1952     <xs:element ref="xacml:AttributeValue" />
1953   </xs:sequence>
1954   <xs:attribute name="MatchId" type="xs:anyMatch" use="required" />
1955 </xs:complexType>
```

1956 The <ResourceMatch> element is of **ResourceMatchType** complex type.

1957 The <ResourceMatch> element contains the following attributes and elements:

1958 MatchId [Required]

1959 Specifies a matching function. Values of this attribute MUST be of type xs:anyURI, with  
1960 legal values documented in Appendix A.

1961 <ResourceAttributeDesignator> [Required Choice]

1962 Identifies one or more **attribute** values in the <xacml-context:Resource> child of the  
1963 <xacml-context:Request> element.

1964 <AttributeSelector> [Required Choice]

1965 MAY be used to identify one or more **attribute** values in the <xacml-  
1966 context:Resource> child of the <xacml-context:Request> element.

1967 <AttributeValue> [Required]

1968 Embedded *attribute* value.

## 1969 **5.14. Element <Actions>**

1970 The <Actions> element SHALL contain a *disjunctive sequence* of <Action> elements.

```
1971 <xs:element name="Actions" type="xacml:ActionTypes" />
1972 <xs:complexType name="ActionTypes">
1973   <xs:choice>
1974     <xs:element ref="xacml:Action" maxOccurs="unbounded" />
1975     <xs:element ref="xacml:AnyAction" />
1976   </xs:choice>
1977 </xs:complexType>
```

1978 The <Actions> element is of **ActionTypes** complex type.

1979 The <Actions> element contains the following elements:

1980 <Action> [One To Many, Required Choice]

1981 See section 5.15.

1982 <AnyAction> [Required Choice]

1983 See section 5.16.

## 1984 **5.15. Element <Action>**

1985 The <Action> element SHALL contain a *conjunctive sequence* of <ActionMatch> elements.

```
1986 <xs:element name="Action" type="xacml:ActionType" />
1987 <xs:complexType name="ActionType">
1988   <xs:sequence>
1989     <xs:element ref="xacml:ActionMatch" maxOccurs="unbounded" />
1990   </xs:sequence>
1991 </xs:complexType>
```

1992 The <Action> element is of **ActionType** complex type.

1993 The <Action> element contains the following elements:

1994 <ActionMatch> [One to Many]

1995 A *conjunctive sequence* of individual matches of the *action* attributes in the *context* and  
1996 the embedded *attribute* values.

## 1997 **5.16. Element <AnyAction>**

1998 The <AnyAction> element SHALL match any *action attribute* in the *context*.

```
1999 <xs:element name="AnyAction" />
```

2000

2001

## 5.17. Element <ActionMatch>

2002 The <ActionMatch> element SHALL identify a set of **action**-related entities by matching **attribute**  
2003 values in the <xacml-context:Action> element of the **context** with the embedded **attribute**  
2004 value.

```

2005 <xs:element name="ActionMatch" type="xacml:ActionMatchType"/>
2006 <xs:complexType name="ActionMatchType">
2007   <xs:sequence>
2008     <xs:choice>
2009       <xs:element ref="xacml:ActionAttributeDesignator"/>
2010       <xs:element ref="xacml:AttributeSelector"/>
2011     </xs:choice>
2012     <xs:element ref="xacml:AttributeValue"/>
2013   </xs:sequence>
2014   <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
2015 </xs:complexType>

```

2016 The <ActionMatch> element is of **ActionMatchType** complex type.

2017 The <ActionMatch> element contains the following attributes and elements:

2018 MatchId [Required]

2019 Specifies a matching function. The value of this attribute MUST be of type xs:anyURI, with  
2020 legal values documented in Appendix A.

2021 <ActionAttributeDesignator> [Required Choice]

2022 Identifies one or more **attribute** values in the <xacml-context:Action> child of the  
2023 <xacml-context:Request> element.

2024 <AttributeSelector> [Required Choice]

2025 MAY be used to identify one or more **attribute** values in the <xacml-context:Action>  
2026 child of the <xacml-context:Request> element.

2027 <AttributeValue> [Required]

2028 Embedded **attribute** value.

## 2029 5.18. Element <PolicySetIdReference>

2030 The <PolicySetIdReference> element SHALL be used to reference a <PolicySet> element  
2031 by id. If <PolicySetIdReference> is a URL, then it MAY be resolvable to the <PolicySet>.  
2032 The mechanism for resolving a **policy set** reference to the corresponding **policy set** is  
2033 implementation dependent.

```

2034 <xs:element name="PolicySetIdReference" type="xs:anyURI"/>

```

2035 Element <PolicySetIdReference> is of **xs:anyURI** simple type.

## 2036 5.19. Element <PolicyIdReference>

2037 The <xacml:PolicyIdReference> element SHALL be used to reference a <Policy> element  
2038 by id. If <PolicyIdReference> is a URL, then it MAY be resolvable to the <Policy>. The  
2039 mechanism for resolving a **policy** reference to the corresponding **policy** is implementation  
2040 dependent.

2041 `<xs:element name="PolicyIdReference" type="xs:anyURI"/>`

2042 Element `<PolicyIdReference>` is of **xs:anyURI** simple type.

## 2043 **5.20. Element `<Policy>`**

2044 The `<Policy>` element is the smallest entity that SHALL be presented to the **PDP** for evaluation.

2045 The main components of this element are the `<Target>`, `<Rule>` and `<Obligations>` elements  
2046 and the `RuleCombiningAlgId` attribute.

2047 The `<Target>` element SHALL define `<Policy>` applicability to **decision requests**. A sequence  
2048 of `<Rule>` elements SHALL specify authorizations that MUST be combined according to the  
2049 `RuleCombiningAlgId` attribute. The `<Obligations>` element SHALL contain a set of  
2050 **obligations** that MUST be discharged by the **PDP** in conjunction with the **authorization decision**.

```
2051 <xs:element name="Policy" type="xacml:PolicyType"/>
2052 <xs:complexType name="PolicyType">
2053   <xs:sequence>
2054     <xs:element ref="xacml:Description" minOccurs="0"/>
2055     <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>
2056     <xs:element ref="xacml:Target"/>
2057     <xs:element ref="xacml:Rule" minOccurs="0" maxOccurs="unbounded"/>
2058     <xs:element ref="xacml:Obligations" minOccurs="0"/>
2059   </xs:sequence>
2060   <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
2061   <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>
2062 </xs:complexType>
```

2063 The `<Policy>` element is of **PolicyType** complex type.

2064 The `<Policy>` element contains the following attributes and elements:

2065 `PolicyId` [Required]

2066 Policy identifier. The party assigning this identifier MUST minimize the potential of some  
2067 other party reusing the same identifier. This MAY be achieved by following a predefined  
2068 URN or URL scheme. It is OPTIONAL for the `PolicyId` URL to be resolvable to the  
2069 corresponding `<Policy>` object.

2070 `RuleCombiningAlgId` [Required]

2071 The identifier of the rule-combining algorithm by which the `<Policy>` components MUST be  
2072 combined. Standard rule-combining algorithms are listed in Appendix C. Standard rule-  
2073 combining algorithm identifiers are listed in Section B.10.

2074 `<Description>` [Optional]

2075 A free-form description of the **policy**.

2076 `<PolicyDefaults>` [Optional]

2077 Defines a set of default values applicable to the **policy**. The scope of the  
2078 `<PolicyDefaults>` element SHALL be the enclosing policy.

2079 `<Target>` [Required]

2080 The `<Target>` element SHALL define the applicability of a `<Policy>` to **decision requests**.

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

2084 <Rule> [Any Number]

2085 A sequence of authorizations that MUST be combined according to the  
2086 RuleCombiningAlgId attribute. **Rules** whose <Target> elements match the **decision**  
2087 **request** MUST be considered. **Rules** whose <Target> elements do not match the  
2088 **decision request** MUST NOT be considered. Applicability of **rules** to the **decision**  
2089 **request** is detailed in Appendix C.

2090 <Obligations> [Optional]

2091 A **conjunctive sequence** of **obligations** that MUST be discharged by the **PEP** in  
2092 conjunction with the **authorization decision**. See Section 7.11 for a description of how the  
2093 set of obligations to be returned by the **PDP** shall be determined.

## 2094 5.21. Element <Rule>

2095 The <Rule> element SHALL define individual **rules** in the **policy**. The main components of this  
2096 element are the <Target> and <Condition> elements and the Effect attribute.

```
2097 <xs:element name="Rule" type="xacml:RuleType"/>  
2098 <xs:complexType name="RuleType">  
2099   <xs:sequence>  
2100     <xs:element ref="xacml:Description" minOccurs="0"/>  
2101     <xs:element ref="xacml:Target" minOccurs="0"/>  
2102     <xs:element ref="xacml:Condition" minOccurs="0"/>  
2103   </xs:sequence>  
2104   <xs:attribute name="RuleId" type="xs:anyURI" use="required"/>  
2105   <xs:attribute name="Effect" type="xacml:EffectType" use="required"/>  
2106 </xs:complexType>
```

2107 The <Rule> element is of **RuleType** complex type.

2108 The <Rule> element contains the following attributes and elements:

2109 RuleId [Required]

2110 A URN identifying this **rule**.

2111 Effect [Required]

2112 **Rule effect**. Values of this attribute are either "Permit" or "Deny".

2113 <Description> [optional]

2114 A free-form description of the **rule**.

2115 <Target> [optional]

2116 Identifies the set of **decision requests** that the <Rule> element is intended to evaluate. If  
2117 this element is omitted, then the **target** for the <Rule> SHALL be defined by the enclosing  
2118 <Policy> element. See Section 5.5 for details.

2119 <Condition> [optional]

2120 A *predicate* that MUST be satisfied for the *rule* to be assigned its `Effect` value. A  
2121 *condition* is a boolean function over a combination of *subject*, *resource*, *action* and  
2122 *environment attributes* or other functions.

## 2123 5.22. Simple type `EffectType`

2124 The `EffectType` simple type defines the values allowed for the `Effect` attribute of the `<Rule>`  
2125 element and for the `FulfillOn` attribute of the `<Obligation>` element.

```
2126 <xs:simpleType name="EffectType">  
2127   <xs:restriction base="xs:string">  
2128     <xs:enumeration value="Permit"/>  
2129     <xs:enumeration value="Deny"/>  
2130   </xs:restriction>  
2131 </xs:simpleType>
```

## 2132 5.23. Element `<Condition>`

2133 The `<Condition>` element is a boolean function over *subject*, *resource*, *action* and  
2134 *environment attributes* or functions of *attributes*. If the `<Condition>` element evaluates to  
2135 "True", then the enclosing `<Rule>` element is assigned its `Effect` value.

```
2136 <xs:element name="Condition" type="xacml:ApplyType"/>
```

2137 The `<Condition>` element is of `ApplyType` complex type.

## 2138 5.24. Element `<Apply>`

2139 The `<Apply>` element denotes application of a function to its arguments, thus encoding a function  
2140 call. The `<Apply>` element can be applied to any combination of `<Apply>`,  
2141 `<AttributeValue>`, `<SubjectAttributeDesignator>`,  
2142 `<ResourceAttributeDesignator>`, `<ActionAttributeDesignator>`,  
2143 `<EnvironmentAttributeDesignator>` and `<AttributeSelector>` arguments.

```
2144 <xs:element name="Apply" type="xacml:ApplyType"/>  
2145 <xs:complexType name="ApplyType">  
2146   <xs:choice minOccurs="0" maxOccurs="unbounded">  
2147     <xs:element ref="xacml:Function"/>  
2148     <xs:element ref="xacml:Apply"/>  
2149     <xs:element ref="xacml:AttributeValue"/>  
2150     <xs:element ref="xacml:SubjectAttributeDesignator"/>  
2151     <xs:element ref="xacml:ResourceAttributeDesignator"/>  
2152     <xs:element ref="xacml:ActionAttributeDesignator"/>  
2153     <xs:element ref="xacml:EnvironmentAttributeDesignator"/>  
2154     <xs:element ref="xacml:AttributeSelector"/>  
2155   </xs:choice>  
2156   <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>  
2157 </xs:complexType>
```

2158 The `<Apply>` element is of `ApplyType` complex type.

2159 The `<Apply>` element contains the following attributes and elements:

2160 `FunctionId` [Required]

2161 The URN of a function. XACML-defined functions are described in Appendix A.

2162 `<Function>` [Optional]

2163 The name of a function that is applied to the elements of a **bag**. See section A14.11.

2164 <Apply> [Optional]

2165 A nested function-call argument.

2166 <AttributeValue> [Optional]

2167 A literal value argument.

2168 <ResourceAttributeDesignator> [Optional]

2169 A **resource attribute** argument.

2170 <ActionAttributeDesignator> [Optional]

2171 An **action attribute** argument.

2172 <EnvironmentAttributeDesignator> [Optional]

2173 An **environment attribute** argument.

2174 <AttributeSelector> [Optional]

2175 An **attribute** selector argument.

## 2176 5.25. Element <Function>

2177 The `Function` element SHALL be used to name a function that is applied by the higher-order **bag**  
2178 functions to every element of a **bag**. The higher-order **bag** functions are described in Section  
2179 A14.11.

```
2180 <xs:element name="Function" type="xacml:FunctionType"/>  
2181 <xs:complexType name="FunctionType">  
2182   <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>  
2183 </xs:complexType>
```

2184 The `Function` element is of **FunctionType** complex type.

2185 The `Function` element contains the following attributes:

2186 `FunctionId` [Required]

2187 The identifier for the function that is applied to the elements of a **bag** by the higher-order  
2188 **bag** functions.

## 2189 5.26. Complex type `AttributeDesignatorType`

2190 The **AttributeDesignatorType** complex type is the type for elements and extensions that refer to  
2191 named **attributes**. A named **attribute** has specific criteria with which to match **attributes** within a  
2192 specific part of the `<xacml-context:Request>` element. The **AttributeDesignatorType**  
2193 complex type specifies the attributes used for the match criteria that are common to all named  
2194 **attributes**. Elements and extensions of the **AttributeDesignatorType** complex type MAY  
2195 determine the presence of named **attributes** or select **attribute values** associated with **attributes**  
2196 that match named **attributes**. Elements and extensions of the **AttributeDesignatorType** SHALL  
2197 NOT alter the match semantics of named **attributes**, but MAY narrow the search space.

2198

```
2199 <xs:complexType name="AttributeDesignatorType">
```

```

2200 <xs:attribute name="AttributeId" type="xs:anyURI" use="required" />
2201 <xs:attribute name="DataType" type="xs:anyURI" use="required" />
2202 <xs:attribute name="Issuer" type="xs:anyURI" use="optional" />
2203 <xs:attribute name="MustBePresent" type="xs:boolean" use="optional" />
2204 </xs:complexType>

```

2205

2206 A named **attribute** SHALL match an **attribute** if the values of their respective `AttributeId`,  
 2207 `DataType` and `Issuer` attributes match. The `AttributeId` attribute MUST match, by URI  
 2208 equality, that of the `AttributeId` attribute of the **attribute**. The `DataType` attribute MUST match,  
 2209 by URI equality, that of the `DataType` attribute of the same **attribute**. If the `Issuer` attribute is  
 2210 supplied, it MUST match, by URI equality, the `Issuer` attribute of the same **attribute**. If the  
 2211 `Issuer` attribute is not supplied, the matching of the **attribute** to the **named attribute** SHALL be  
 2212 governed by `AttributeId` and `DataType` attributes alone, regardless of the presence, absence,  
 2213 or actual value of the `Issuer` attribute.

2214 The `<AttributeDesignatorType>` contains the following attributes:

2215 `AttributeId` [Required]

2216 This attribute SHALL specify the `AttributeId` with which to match the **attribute**.

2217 `DataType` [Required]

2218 This attribute SHALL specify the data-type with which to match the **attribute**.

2219 `Issuer` [Optional]

2220 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **attribute**.

2221 `MustBePresent` [Optional]

2222 This attribute governs whether the element returns “Indeterminate” in the case of the  
 2223 absence of the named **attribute**. If the **named attribute** is absent and `MustBePresent` is  
 2224 set to “True”, then this element SHALL result in “Indeterminate”. If `MustBePresent` is not  
 2225 supplied, its default value SHALL be `false`.

## 2226 5.27. Element `<ResourceAttributeDesignator>`

2227

2228 The `<ResourceAttributeDesignator>` element retrieves a **bag** of values for a **named**  
 2229 **resource attribute**. A **resource attribute** is an **attribute** that SHALL only be located within the  
 2230 `<Resource>` element of the `<xacml-context:Request>` element. A **named resource attribute**  
 2231 is a **named attribute** that matches a **resource attribute**. A **named resource attribute** SHALL be  
 2232 considered **present** if there is at least one **resource attribute** that matches the criteria set out  
 2233 below. A **resource attribute** value is an **attribute** value that is contained within a **resource**  
 2234 **attribute**.

2235 The `<ResourceAttributeDesignator>` element SHALL return a **bag** of all the **resource**  
 2236 **attribute** values that are matched by the **named resource attribute**. The `MustBePresent` attribute  
 2237 governs whether this element returns an empty **bag** or “Indeterminate” in the case of the absence  
 2238 of the **named resource attribute**. If the **named resource attribute** is not present and the  
 2239 `MustBePresent` attribute is set to “False” (its default value), then this element SHALL result in an  
 2240 empty **bag**. If the **named resource attribute** is not present and the `MustBePresent` attribute is set  
 2241 to “True”, then this element SHALL result in “Indeterminate”. Regardless of the `MustBePresent`  
 2242 attribute, if it cannot be determined whether the **named resource attribute** is present or not in the

2243 **request context**, or the value of the *named resource attribute* is unavailable, then the expression  
2244 SHALL evaluate to “Indeterminate”.

2245 A *named resource attribute* SHALL match a **resource attribute** as per the match semantics  
2246 specified in the **AttributeDesignatorType** complex type [Section 5.26]

2247 The <ResourceAttributeDesignator> MAY appear in the <ResourceMatch> element and  
2248 MAY be passed to the <Apply> element as an argument.

```
2249 <xs:element name="ResourceAttributeDesignator"  
2250           type="xacml:AttributeDesignatorType" />
```

2251 The <ResourceAttributeDesignator> element is of the **AttributeDesignatorType** complex  
2252 type.

2253 The <ResourceAttributeDesignator> element has the following attributes:

2254 AttributeId [Required]

2255 This attribute SHALL specify the AttributeId with which to match the **resource**  
2256 **attribute**.

2257 DataType [Required]

2258 This attribute SHALL specify the data-type with which to match the **resource attribute**.

2259 Issuer [Optional]

2260 This attribute, if supplied, SHALL specify the Issuer with which to match the **resource**  
2261 **attribute**.

2262 MustBePresent [Optional]

2263 This attribute governs whether the <ResourceAttributeDesignator> element returns  
2264 an empty **bag** or “Indeterminate” in the case of the absence of the *named resource*  
2265 *attribute*. If the *named resource attribute* is absent and MustBePresent is set to “False”,  
2266 then this element SHALL result in an empty **bag**. If the *named resource attribute* is absent  
2267 and MustBePresent is set to “True”, then this element SHALL evaluate to  
2268 “Indeterminate”. If MustBePresent is not supplied, then its default value SHALL be  
2269 “False”.

## 2270 **5.28. Element <ActionAttributeDesignator>**

2271

2272 The <ActionAttributeDesignator> element retrieves a **bag** of values for a *named action*  
2273 *attribute*. An **action attribute** is an **attribute** that SHALL only be located within the <Action>  
2274 element of the <xacml-context:Request> element. A *named action attribute* has specific  
2275 criteria (described below) with which to match an **action attribute**. A *named action attribute*  
2276 SHALL be considered *present*, if there is at least one **action attribute** that matches the criteria. An  
2277 **action attribute value** is an **attribute value** that is contained within an **action attribute**.

2278 The <ActionAttributeDesignator> element SHALL return a **bag** of all the **action attribute**  
2279 values that are matched by the *named action attribute*. The MustBePresent attribute governs  
2280 whether this element returns an empty **bag** or “Indeterminate” in the case of the absence of the  
2281 *named action attribute*. If the *named action attribute* is not present and the MustBePresent  
2282 attribute is set to “False” (its default value), then this element SHALL result in an empty **bag**. If the  
2283 *named action attribute* is not present and the MustBePresent attribute is set to “True”, then this  
2284 element SHALL result in “Indeterminate”. Regardless of the MustBePresent attribute, if it cannot

2285 be determined whether the *named action attribute* is present or not present in the request **context**,  
2286 or the value of the *named action attribute* is unavailable, then the expression SHALL result in  
2287 “Indeterminate”.

2288 A *named action attribute* SHALL match an **action attribute** as per the match semantics specified in  
2289 the **AttributeDesignatorType** complex type [Section 5.26].

2290 The <ActionAttributeDesignator> MAY appear in the <ActionMatch> element and MAY  
2291 be passed to the <Apply> element as an argument.

```
2292 <xs:element name="ActionAttributeDesignator"  
2293           type="xacml:AttributeDesignatorType" />
```

2294 The <ActionAttributeDesignator> element is of the **AttributeDesignatorType** complex  
2295 type.

2296 The <ActionAttributeDesignator> element has the following attributes:

2297 AttributeId [Required]

2298 This attribute SHALL specify the AttributeId with which to match the **action attribute**.

2299 DataType [Required]

2300 This attribute SHALL specify the data-type with which to match the **action attribute**.

2301 Issuer [Optional]

2302 This attribute, if supplied, SHALL specify the Issuer with which to match the **action**  
2303 **attribute**.

2304 MustBePresent [Optional]

2305 This attribute governs whether the <ActionAttributeDesignator> element returns an  
2306 empty **bag** or “Indeterminate” in the case of the absence of the *named action attribute*. If  
2307 the *named action attribute* is absent and MustBePresent is set to “False”, then this  
2308 element SHALL result in an empty **bag**. If the *named action attribute* is absent and  
2309 MustBePresent is set to “True”, this element SHALL result in “Indeterminate”. If  
2310 MustBePresent is not supplied, then its default value SHALL be “False”.

## 2311 5.29. Element <EnvironmentAttributeDesignator>

2312

2313 The <EnvironmentAttributeDesignator> element retrieves a **bag** of values for a *named*  
2314 *environment attribute*. An **environment attribute** is an **attribute** that SHALL only be located within  
2315 the <Environment> element of the <xacml-context:Request> element. A *named*  
2316 *environment attribute* has specific criteria (described below) with which to match an **environment**  
2317 **attribute**. A *named environment attribute* SHALL be considered *present*, if there is at least one  
2318 **environment attribute** that matches the criteria. An **environment attribute value** is an **attribute**  
2319 value that is contained within an **environment attribute**.

2320 The <EnvironmentAttributeDesignator> element SHALL return a **bag** of all the  
2321 **environment attribute** values that are matched by the *named environment attribute*. The  
2322 MustBePresent attribute governs whether this element returns an empty **bag** or “Indeterminate”  
2323 in the case of the absence of the *named environment attribute*. If the *named environment attribute*  
2324 is not present and the MustBePresent attribute is set to “False” (its default value), then this  
2325 element SHALL result in an empty **bag**. If the *named environment attribute* is not present and the  
2326 MustBePresent attribute is set to “True”, then this element SHALL result in “Indeterminate”.

2327 Regardless of the `MustBePresent` attribute, if it cannot be determined whether the *named*  
2328 *environment attribute* is present or not present in the request **context**, or the value of the *named*  
2329 *environment attribute* is unavailable, then the expression SHALL result in “Indeterminate”.

2330 A *named environment attribute* SHALL match an **environment attribute** as per the match  
2331 semantics specified in the **AttributeDesignatorType** complex type [Section 5.26].

2332 The `<EnvironmentAttributeDesignator>` MAY be passed to the `<Apply>` element as an  
2333 argument.

```
2334 <xs:element name="EnvironmentAttributeDesignator"  
2335           type="xacml:AttributeDesignatorType"/>
```

2336 The `<EnvironmentAttributeDesignator>` element is of the **AttributeDesignatorType**  
2337 complex type.

2338 The `<EnvironmentAttributeDesignator>` element has the following attributes:

2339 `AttributeId` [Required]

2340 This attribute SHALL specify the `AttributeId` with which to match the **environment**  
2341 **attribute**.

2342 `DataType` [Required]

2343 This attribute SHALL specify the data-type with which to match the **environment attribute**.

2344 `Issuer` [Optional]

2345 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **environment**  
2346 **attribute**.

2347 `MustBePresent` [Optional]

2348 This attribute governs whether the `<EnvironmentAttributeDesignator>` element returns an  
2349 empty **bag** or “Indeterminate” in the case of the absence of the *named environment attribute*. If the  
2350 *named environment attribute* is absent and `MustBePresent` is set to “False”, then this element  
2351 SHALL result in an empty **bag**. If the *named environment attribute* is absent and `MustBePresent`  
2352 is set to “True”, then this element SHALL result in “Indeterminate”. If `MustBePresent` is not  
2353 supplied, its default value SHALL be “False”.

## 2354 5.30. Complex type **SubjectAttributeDesignatorType**

2355 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType**  
2356 complex type. It is the base type for elements and extensions that refer to *named categorized*  
2357 **subject attributes**. A *named categorized subject attribute* is defined as follows:

2358 A **subject** is represented by a `<Subject>` element of the `<Subjects>` element in the `<xacml-`  
2359 `context:Request>` element. Each `<Subject>` element SHALL contain one and only one  
2360 attribute with an `AttributeId` attribute that is equal to “urn:oasis:tc:xacml:1.0:subject-  
2361 `category”`. This attribute is called the *subject category attribute*. The `DataType` attribute of the  
2362 *subject category attribute* SHALL be equal to “http://www.w3.org/2001/XMLSchema-  
2363 `instance#string”`.

2364 A *categorized subject* is a **subject** that is identified by a particular **subject category attribute**.

2365 A **subject attribute** is an **attribute** of a particular **subject**, i.e. located within a `<Subject>`  
2366 element.

- 2367 A named **subject attribute** is a named **attribute** for a **subject**.
- 2368 A named categorized **subject attribute** is a named **subject attribute** for a particular **categorized**  
2369 **subject**.
- 2370 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType** with a  
2371 **SubjectCategory** attribute. The **SubjectAttributeDesignatorType** extends the match  
2372 semantics of the **AttributeDesignatorType** such that it narrows the **attribute** search space to the  
2373 specific **categorized subject** such that the value of the **SubjectCategory** attribute matches by  
2374 string-equality the value of the subject's **subject category attribute**.
- 2375 If there are multiple **subjects** with the same **subject category attribute**, then they SHALL be  
2376 treated as if they were one **categorized subject**.
- 2377 Elements and extensions of the **SubjectAttributeDesignatorType** complex type determine the  
2378 presence of select **attribute values** associated with **named categorized subject attributes**.
- 2379 Elements and extensions of the **SubjectAttributeDesignatorType** SHALL NOT alter the match  
2380 semantics of **named categorized subject attributes**, but MAY narrow the search space.

```

2381 <xs:complexType name="SubjectAttributeDesignatorType">
2382   <xs:complexContent>
2383     <xs:extension base="xacml:AttributeDesignatorType">
2384       <xs:attribute name="SubjectCategory"
2385         type="xs:anyURI"
2386         use="optional"
2387         default="
2388           "urn:oasis:tc:xacml:1.0:subject-category:access-subject" />
2389     </xs:extension>
2390   </xs:complexContent>
2391 </xs:complexType>

```

- 2392 The <SubjectAttributeDesignatorType> complex type contains the following attribute in  
2393 addition to the attributes of the **AttributeDesignatorType** complex type:

2394 **SubjectCategory** [Optional]

- 2395 This attribute SHALL specify the **categorized subject** from which to match **named subject**  
2396 **attributes**. If **SubjectCategory** is not supplied, then its default value SHALL be  
2397 "urn:oasis:tc:xacml:1.0:subject-category:access-subject".

## 2398 **5.31. Element <AttributeSelector>**

- 2399 The **AttributeSelector** element's **RequestContextPath** XML attribute SHALL contain a  
2400 legal XPath expression over the <xacml-context:Request> element. The  
2401 **AttributeSelector** element evaluates to a **bag** of values of a single primitive type that is  
2402 specified by the selector's **DataType** attribute. In the case where the XPath expression matches  
2403 **attributes** in the request **context** by **AttributeId**, it must also match the **attribute's** data-type with  
2404 the selector's **DataType**. In the case of using XPath 1.0, the value of the XPath expression is  
2405 either a node-set, string value, numeric value, or boolean value. If the XPath 1.0 expression  
2406 evaluates to a node-set, then each node may consist of a string, numeric or boolean value, or a  
2407 child node (i.e. structured node). In this case, each node is logically converted to string data by  
2408 applying the "string" function defined in the XPath 1.0 specification, resulting in a sequence of  
2409 string data. In the single string, numeric or boolean value case, the value is converted to string  
2410 data by applying the "string" function defined in the XPath 1.0 specification, resulting in a  
2411 sequence of one string data element. In XPath 2.0, the result of the XPath expression is a  
2412 sequence of items (where an item is an atomic value or a node) or the error value. When the error  
2413 value is returned, the **PDP** SHOULD return "Indeterminate". Otherwise, each node is logically  
2414 converted to a string using the **xf:string** accessor function, resulting in a sequence of string

2415 data. The resulting sequence of string data is converted to a **bag** of primitive values that is implied  
2416 by the type system.

2417 Support for the <AttributeSelector> element is OPTIONAL.

```
2418 <xs:element name="AttributeSelector" type="xacml:AttributeSelectorType" />
2419 <xs:complexType name="AttributeSelectorType">
2420   <xs:attribute name="RequestContextPath" type="xs:anyURI" use="required" />
2421   <xs:attribute name="DataType" type="xs:anyURI" use="required" />
2422   <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"
2423   default="false"
2424 </xs:complexType>
```

2425 The <AttributeSelector> element is of **AttributeSelectorType** complex type.

2426 The <AttributeSelector> element has the following attributes:

2427 RequestContextPath [Required]

2428 An XPath expression into the request **context**. There SHALL be no restriction on the XPath  
2429 syntax.

2430 DataType [Required]

2431 The data-type of the **attribute**.

2432 MustBePresent [Optional]

2433 Whether or not designated **attribute** must be present in the **context**.

## 2434 5.32. Element <AttributeValue>

2435 The <AttributeValue> element SHALL contain a literal **attribute** value.

```
2436 <xs:element name="AttributeValue" type="xacml:AttributeValueType" />
2437 <xs:complexType name="AttributeValueType" mixed="true">
2438   <xs:sequence>
2439     <xs:any namespace="##any" processContents="lax" minOccurs="0"
2440     maxOccurs="unbounded" />
2441   </xs:sequence>
2442   <xs:attribute name="DataType" type="xs:anyURI" use="required" />
2443   <xs:anyAttribute namespace="##any" processContents="lax" />
2444 </xs:complexType>
```

2445 The <AttributeValue> element is of **AttributeValueType** complex type.

2446 The <AttributeValue> element has following attributes:

2447 DataType [Required]

2448 The data-type of the **attribute** value.

## 2449 5.33. Element <Obligations>

2450 The <Obligations> element SHALL contain a set of <Obligation> elements.

```
2451 <xs:element name="Obligations" type="xacml:ObligationsType" />
2452 <xs:complexType name="ObligationsType">
2453   <xs:sequence>
2454     <xs:element ref="xacml:Obligation" maxOccurs="unbounded" />
2455   </xs:sequence>
```

2456 </xs:complexType>

2457 The <Obligations> element is of **ObligationsType** complexType.

2458 <Obligation> [One to Many]

2459 A sequence of **obligations**

### 2460 **5.34. Element <Obligation>**

2461 The <Obligation> element SHALL contain an identifier for the **obligation** and a set of **attributes**  
2462 that form arguments of the action defined by the **obligation**. The FulfillOn attribute SHALL  
2463 indicate the **effect** for which this **obligation** applies.

```
2464 <xs:element name="Obligation" type="xacml:ObligationType"/>
2465 <xs:complexType name="ObligationType">
2466   <xs:sequence>
2467     <xs:element ref="xacml:AttributeAssignment" maxOccurs="unbounded"/>
2468   </xs:sequence>
2469   <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
2470   <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required"/>
2471 </xs:complexType>
```

2472 The <Obligation> element is of **ObligationType** complexType. See Section 7.11 for a  
2473 description of how the set of **obligations** to be returned by the PDP is determined.

2474 The ObligationId [required]

2475 **Obligation** identifier. The value of the **obligation** identifier SHALL be interpreted by the  
2476 **PEP**.

2477 FulfillOn [required]

2478 The **effect** for which this **obligation** applies.

2479 <AttributeAssignment> [required]

2480 **Obligation** arguments assignment. The values of the **obligation** arguments SHALL be  
2481 interpreted by the **PEP**.

### 2482 **5.35. Element <AttributeAssignment>**

2483 The <AttributeAssignment> element SHALL contain an AttributeId and the corresponding  
2484 **attribute** value. The AttributeId is part of **attribute** meta-data, and is used when the **attribute**  
2485 cannot be referenced by its location in the <xacml-context:Request>. This situation may arise  
2486 in an <Obligation> element if the **obligation** includes parameters.

```
2487 <xs:element name="AttributeAssignment" type="xacml:AttributeAssignmentType"/>
2488 <xs:complexType name="AttributeAssignmentType">
2489   <xs:complexContent>
2490     <xs:extension base="xacml:AttributeValueType">
2491       <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
2492     </xs:extension>
2493   </xs:complexContent>
2494 </xs:complexType>
```

2495 The <AttributeAssignment> element is of **AttributeAssignmentType** complex type.

2496 AttributeId [Required]

2497 The *attribute* Identifier  
2498 DataType [Required]  
2499 The data-type for the assigned value.

---

## 2500 6. Context syntax (normative with the exception of 2501 the schema fragments)

### 2502 6.1. Element <Request>

2503 The <Request> element is a top-level element in the XACML *context* schema. The <Request>  
2504 element is an abstraction layer used by the *policy* language. Any proprietary system using the  
2505 XACML specification MUST transform its input into the form of an XACML *context*<Request>.

2506 The <Request> element consists of sections denoted by the <Subject>, <Resource>,  
2507 <Action>, and <Environment> elements. There may be multiple <Subject> elements. Each  
2508 section contains a sequence of XACML context <Attribute> elements associated with the  
2509 *subject*, *resource*, *action*, and *environment* respectively.

```
2510 <xs:element name="Request" type="xacml-context:RequestType" />  
2511 <xs:complexType name="RequestType">  
2512   <xs:sequence>  
2513     <xs:element ref="xacml-context:Subject" maxOccurs="unbounded" />  
2514     <xs:element ref="xacml-context:Resource" />  
2515     <xs:element ref="xacml-context:Action" />  
2516     <xs:element ref="xacml-context:Environment" minOccurs="0" />  
2517   </xs:sequence>  
2518 </xs:complexType>
```

2519 The <Request> element is of **RequestType** complex type.

2520 The <Request> element contains the following elements:

2521 <Subject> [One to Many]

2522 Specifies information about a *subject* of the request *context* by listing a sequence of  
2523 <Attribute> elements associated with the *subject*. One or more <Subject> elements  
2524 are allowed. A *subject* is an entity associated with making the *access* request. One  
2525 *subject* might be a human user that initiated the application from which the request is  
2526 being issued. Another *subject* might be the application's executable code that issued this  
2527 request. Another *subject* might be the machine on which the application is executing.  
2528 Another *subject* might be the target entity that is to be the recipient of the resource.  
2529 Attributes of each of these entities MUST be enclosed in a separate <Subject> element.

2530 <Resource> [Required]

2531 Specifies information about the *resource* for which access is being requested by listing a  
2532 sequence of <Attribute> elements associated with the resource. It MAY

2533 include a <ResourceContent> element.

2534 <Action> [Required]

- 2535 Specifies the requested **action** to be performed on the **resource** by listing a set of  
 2536 <Attribute> elements associated with the action.
- 2537 <Environment> [Optional]
- 2538 Contains a set of <Attribute> elements of the **environment**. These <Attribute>  
 2539 elements MAY form a part of **policy** evaluation.

## 2540 6.2. Element <Subject>

- 2541 The <Subject> element specifies a **subject** of a **decision request context** by listing a sequence  
 2542 of <Attribute> elements associated with the **subject**.

```

2543 <xs:element name="Subject" type="xacml-context:SubjectType" />
2544 <xs:complexType name="SubjectType">
2545   <xs:sequence>
2546     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2547 maxOccurs="unbounded" />
2548   </xs:sequence>
2549 </xs:complexType>
  
```

- 2550 The <Subject> element is of **SubjectType** complex type.

- 2551 <Attribute> [Any Number]

- 2552 A sequence of **attributes** that apply to the **subject**.

- 2553 Every <Subject> element MUST contain one and only one <Attribute> with AttributeId  
 2554 "urn:oasis:names:tc:xacml:1.0:subject:subject-category". This **attribute**  
 2555 indicates a role that the parent <Subject> entity plays in making the **access** request. If this  
 2556 **attribute** is not present in a given <Subject> element, that <Subject> implicitly contains this  
 2557 **attribute** with the value of "urn:oasis:names:tc:xacml:1.0:subject:subject-  
 2558 category:access-subject", indicating that the **subject** is the entity ultimately associated  
 2559 with initiating the **access** request. Typically, a <Subject> element will also contain an  
 2560 <Attribute> with AttributeId "urn:oasis:names:tc:xacml:1.0:subject:subject-  
 2561 id", containing the identity of the **subject** entity.

- 2562 No more than one <Subject> element may contain an <Attribute> with the given value for  
 2563 AttributeId "urn:oasis:names:tc:xacml:1.0:subject:subject-category".

- 2564 A <Subject> element MAY contain additional <Attribute> elements.

## 2565 6.3. Element <Resource>

- 2566 The <Resource> element specifies information about the **resource** for which access is being  
 2567 requested by listing a sequence of <Attribute> elements associated with the resource. It MAY  
 2568 include the **resource** content.

```

2569 <xs:element name="Resource" type="xacml-context:ResourceType" />
2570 <xs:complexType name="ResourceType">
2571   <xs:sequence>
2572     <xs:element ref="xacml-context:ResourceContent" minOccurs="0" />
2573     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2574 maxOccurs="unbounded" />
2575   </xs:sequence>
2576 </xs:complexType>
  
```

- 2577 The <Resource> element is of **ResourceType** complex type.

2578 The <Resource> element contains the following elements:

2579 <ResourceContent> [Optional]

2580 The **resource** content.

2581 <Attribute> [Any Number]

2582 A sequence of **resource attributes**. The <Resource> element MUST contain one and  
2583 only one <Attribute> with AttributeId

2584 "urn:oasis:names:tc:xacml:1.0:resource:resource-id". This **attribute**  
2585 specifies the identity of the **resource** for which **access** is requested. The <Resource>  
2586 element MAY contain additional <Attribute> elements.

## 2587 6.4. Element <ResourceContent>

2588 The <ResourceContent> element is a notional placeholder for the **resource** content. If an  
2589 XACML **policy** references the contents of the **resource**, then the <ResourceContent> element  
2590 is used as the reference point.

```
2591 <xs:complexType name="ResourceContentType" mixed="true">  
2592 <xs:sequence>  
2593 <xs:any namespace="##any" processContents="lax" minOccurs="0"  
2594 maxOccurs="unbounded" />  
2595 </xs:sequence>  
2596 <xs:anyAttribute namespace="##any" processContents="lax" />  
2597 </xs:complexType>
```

2598 The <ResourceContent> element is of **ResourceContentType** complex type.

2599 The <ResourceContent> element allows arbitrary elements and attributes.

## 2600 6.5. Element <Action>

2601 The <Action> element specifies the requested **action** to be performed on the **resource** by listing  
2602 a set of <Attribute> elements associated with the **action**.

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

2610 The <Action> element is of **ActionType** complex type.

2611 The <Attribute> [Any Number]

2612 List of **attributes** of the **action** to be performed on the **resource**.

## 2613 6.6. Element <Environment>

2614 The <Environment> element contains a set of **attributes** of the **environment**. These **attributes**  
2615 MAY form part of the **policy** evaluation.

```
2616 <xs:element name="Environment" type="xacml-context:EnvironmentType" />  
2617 <xs:complexType name="EnvironmentType">
```

```

2618     <xs:sequence>
2619         <xs:element ref="xacml-context:Attribute" minOccurs="0"
2620 maxOccurs="unbounded" />
2621     </xs:sequence>
2622 </xs:complexType>

```

2623 The <Environment> element is of **EnvironmentType** complex type.

2624 The <Environment> element contains the following elements:

2625 <Attribute> [Any Number]

2626         A list of **environment attributes**. Environment attributes are attributes that are not  
2627         associated with the **resource**, the **action**, or with any of the **subjects** of the **access**  
2628         request.

## 2629         6.7. Element <Attribute>

2630 The <Attribute> element is the central abstraction of the request **context**. It contains an  
2631 **attribute** value and **attribute** meta-data. The **attribute** meta-data comprises the **attribute**  
2632 identifier, the **attribute** issuer and the **attribute** issue instant. **Attribute** designators and **attribute**  
2633 selectors in the **policy** refer to **attributes** by this meta-data.

```

2634     <xs:element name="Attribute" type="xacml-context:AttributeType" />
2635     <xs:complexType name="AttributeType">
2636         <xs:sequence>
2637             <xs:element ref="xacml-context:AttributeValue" minOccurs="0" />
2638         </xs:sequence>
2639         <xs:attribute name="AttributeId" type="xs:anyURI" use="required" />
2640         <xs:attribute name="DataType" type="xs:anyURI" use="required" />
2641         <xs:attribute name="Issuer" type="xs:string" use="optional" />
2642         <xs:attribute name="IssueInstant" type="xs:dateTime" use="optional" />
2643     </xs:complexType>

```

2644 The <Attribute> element is of **AttributeType** complex type.

2645 The <Attribute> element contains the following attributes and elements:

2646 AttributeId [Required]

2647         **Attribute** identifier. A number of identifiers are reserved by XACML to denote commonly  
2648         used **attributes**.

2649 DataType [Required]

2650         **Attribute** data type.

2651 Issuer [Optional]

2652         **Attribute** issuer. This attribute value MAY be an x500Name that binds to a public key, or it  
2653         may be some other identifier exchanged out-of-band by issuing and relying parties.

2654 IssueInstant [Optional]

2655         The date and time at which the **attribute** was issued.

2656 <AttributeValue> [Optional]

2657         At most one **attribute** value.

2658

## 6.8. Element <AttributeValue>

2659

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

2660

```
<xs:element name="AttributeValue" type="xacml-context:AttributeValueType"/>
```

2661

```
<xs:complexType name="AttributeValueType" mixed="true">
```

2662

```
<xs:sequence>
```

2663

```
<xs:any namespace="##any" processContents="lax" minOccurs="0"
```

2664

```
maxOccurs="unbounded"/>
```

2665

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

2666

```
<xs:anyAttribute namespace="##any" processContents="lax"/>
```

2667

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

2668

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

2669

The data type of the <AttributeValue> MAY be specified by using the DataType attribute of the

2670

parent <Attribute> element.

2671

## 6.9. Element <Response>

2672

The <Response> element encapsulates the *authorization decision* returned by the *PDP*. It

2673

includes a sequence of one or more results with one <Result> element per requested *resource*.

2674

Multiple results MAY be returned when the value of the "urn:oasis:xacml:1.0:resource:scope"

2675

resource *attribute* in the request *context* is "Descendants". Support for multiple results is

2676

OPTIONAL.

2677

```
<xs:element name="Response" type="xacml-context:ResponseType"/>
```

2678

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

2679

```
<xs:sequence>
```

2680

```
<xs:element ref="xacml-context:Result" maxOccurs="unbounded"/>
```

2681

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

2682

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

2683

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

2684

The <Response> element contains the following elements:

2685

<Result> [One to Many]

2686

An authorization decision result.

2687

## 6.10. Element <Result>

2688

The <Result> element represents an *authorization decision* result for the *resource* specified by

2689

the ResourceId *attribute*. It MAY include a set of *obligations* that MUST be fulfilled by the *PEP*.

2690

If the *PEP* does not understand an *obligation*, then it MUST act as if the *PDP* had denied *access*

2691

to the requested *resource*.

2692

```
<xs:element name="Result" type="xacml-context:ResultType"/>
```

2693

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

2694

```
<xs:sequence>
```

2695

```
<xs:element ref="xacml-context:Decision"/>
```

2696

```
<xs:element ref="xacml-context:Status" minOccurs="0"/>
```

2697

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

2698

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

2699

```
<xs:attribute name="ResourceId" type="xs:anyURI" use="optional"/>
```

2700

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

2701

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

2702 The <Result> element contains the following attributes and elements:

2703 ResourceId [Optional]

2704       The identifier of the requested **resource**. If this attribute is omitted, then the **resource**  
2705       identity is specified by the "urn:oasis:names:tc:xacml:1.0:resource:resource-  
2706       id" **resource attribute** in the <Request> element.

2707 <Decision> [Required]

2708       The **authorization decision**: "Permit", "Deny", "Indeterminate", or "Not-applicable".

2709 <Status> [Optional]

2710       Indicates whether errors occurred during evaluation of the request, and optionally,  
2711       information about those errors.

2712 <xacml:Obligations> [Optional]

2713       A list of **obligations** that MUST be discharged by the **PEP**. If the **PEP** does not  
2714       understand an **obligation**, then it MUST act as if the **PDP** had denied **access** to the  
2715       requested **resource**. See Section 7.11 for a description of how the set of obligations to be  
2716       returned by the PDP is determined.

## 2717 6.11. Element <Decision>

2718 The <Decision> element contains the result of **policy** evaluation.

```

2719 <xs:element name="Decision" type="xacml-context:DecisionType" />
2720 <xs:simpleType name="DecisionType">
2721   <xs:restriction base="xs:string">
2722     <xs:enumeration value="Permit" />
2723     <xs:enumeration value="Deny" />
2724     <xs:enumeration value="Indeterminate" />
2725     <xs:enumeration value="Not-applicable" />
2726   </xs:restriction>
2727 </xs:simpleType>

```

2728 The <Decision> element is of **DecisionType** simple type.

2729 The values of the <Decision> element have the following meanings:

2730       "Permit": the requested resource access is permitted.

2731       "Deny": the requested resource access is denied.

2732       "Indeterminate": the **PDP** is unable to evaluate the requested **resource access**. Reasons  
2733       for such inability include: missing **attributes**, network errors while retrieving policies,  
2734       division by zero during policy evaluation, syntax errors in the request or in the policy.

2735       "Not-applicable": the **PDP** does not have any policy that applies to this request.

## 2736 6.12. Element <Status>

2737 The <Status> element represents the status of the **authorization decision** result.

```

2738 <xs:element name="Status" type="xacml-context:StatusType" />
2739 <xs:complexType name="StatusType">
2740   <xs:sequence>
2741     <xs:element ref="xacml-context:StatusCode" />

```

2742  
2743  
2744  
2745

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

2746 The <Status> element is of **StatusType** complex type.

2747 The <Status> element contains the following elements:

2748 <StatusCode> [Required]

2749       Status code.

2750 <StatusMessage> [Optional]

2751       A status message describing the status code.

2752 <StatusDetail> [Optional]

2753       Additional status information.

### 2754       **6.13. Element <StatusCode>**

2755 The <StatusCode> element contains a major status code value and an optional sequence of  
2756 minor status codes.

2757  
2758  
2759  
2760  
2761  
2762  
2763

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

2764 The <StatusCode> element is of **StatusCodeType** complex type.

2765 The <StatusCode> element contains the following attributes and elements:

2766 Value [Required]

2767       See Section B.7 for a list of values.

2768 <StatusCode> [Any Number]

2769       Minor status code. This status code qualifies its parent status code.

### 2770       **6.14. Element <StatusMessage>**

2771 The <StatusMessage> element is a free-form description of the status code.

2772

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

2773 The <StatusMessage> element is of **xs:string** type.

### 2774       **6.15. Element <StatusDetail>**

2775 The <StatusDetail> element qualifies the <Status> element with additional information.

2776  
2777

```
<xs:element name="StatusDetail" type="xacml-context:StatusDetailType" />  
<xs:complexType name="StatusDetailType">
```

2778  
2779  
2780  
2781  
2782

```
<xs:sequence>
  <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded" />
</xs:sequence>
</xs:complexType>
```

2783 The <StatusDetail> element is of **StatusDetailType** complex type.

2784 The <StatusDetail> element allows arbitrary xml content.

2785 Inclusion of a <StatusDetail> element is optional. However, if a **PDP** returns one of the  
2786 following XACML-defined <StatusCode> values and includes a <StatusDetail> element, then  
2787 the following rules apply.

2788 urn:oasis:names:tc:xacml:1.0:status:ok

2789 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “ok” status value.

2790 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

2791 A **PDP** MAY choose not to return any <StatusDetail> information or MAY choose to return a  
2792 <StatusDetail> element containing one or more <xacml-context:Attribute> elements. If  
2793 the **PDP** includes <AttributeValue> elements in the <Attribute> element, then this indicates  
2794 the acceptable values for that **attribute**. If no <AttributeValue> elements are included, then  
2795 this indicates the names of **attributes** that the **PDP** failed to resolve during its evaluation. The list  
2796 of **attributes** may be partial or complete. There is no guarantee by the **PDP** that supplying the  
2797 missing values or **attributes** will be sufficient to satisfy the **policy**.

2798 urn:oasis:names:tc:xacml:1.0:status:syntax-error

2799 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “syntax-error” status  
2800 value. A syntax error may represent either a problem with the **policy** being used or with the  
2801 request **context**. The **PDP** MAY return a <StatusMessage> describing the problem.

2802 urn:oasis:names:tc:xacml:1.0:status:processing-error

2803 A **PDP** MUST NOT return <StatusDetail> element in conjunction with the “processing-error”  
2804 status value. This status code indicates an internal problem in the **PDP**. For security reasons, the  
2805 **PDP** MAY choose to return no further information to the **PEP**. In the case of a divide-by-zero error  
2806 or other computational error, the **PDP** MAY return a <StatusMessage> describing the nature of  
2807 the error.

---

## 2808 7. Functional requirements (normative)

2809 This section specifies certain functional requirements that are not directly associated with the  
2810 production or consumption of a particular XACML element.

### 2811 7.1. Policy enforcement point

2812 This section describes the requiremenst for the **PEP**.

2813 An application functions in the role of the **PEP** if it guards access to a set of **resources** and asks  
2814 the **PDP** for an **authorization decision**. The **PEP** MUST abide by the **authorization decision** in  
2815 the following way:

2816 A **PEP** SHALL allow access to the **resource** only if a valid XACML response of "Permit" is returned  
2817 by the **PDP**. The **PEP** SHALL deny access to the **resource** in all other cases. An XACML

2818 response of "Permit" SHALL be considered valid only if the *PEP* understands all of the *obligations*  
2819 contained in the response.

## 2820 7.2. Base policy

2821 A *PDP* SHALL represent one *policy* or *policy set*, called its *base policy*. This base *policy* MAY be  
2822 a <Policy> element containing a <Target> element that matches every possible *decision*  
2823 *request*, or (for instance) it MAY be a <Policy> element containing a <Target> element that  
2824 matches only a specific *subject*. In such cases, the base policy SHALL form the root-node of a  
2825 tree of policies connected by <PolicyIdReference> and <PolicySetIdReference>  
2826 elements to all the *rules* that may be applicable to any *decision request* that the *PDP* is capable  
2827 of evaluating.

2828 In the case of a *PDP* that retrieves *policies* according to the *decision request* that it is processing,  
2829 the base policy SHALL contain a <Policy> element containing a <Target> element that matches  
2830 every possible *decision request* and a PolicyCombiningAlgId attribute with the value "Only-  
2831 one-applicable". In other words, the *PDP* SHALL return an error if it retrieves policies that do not  
2832 form a single tree.

## 2833 7.3. Target evaluation

2834 The *target* value SHALL be "Match" if the *subjects*, *resource* and *action* specified in the request  
2835 *context* are all present in (i.e., within the scope of) the *target*. Its value SHALL be "No-match" if  
2836 one or more of the *subjects*, *resource* or *action* specified in the request *context* is not present in  
2837 the *target*. Its value SHALL be "Indeterminate" if any *attribute* value referenced in the *target*  
2838 cannot be obtained.

## 2839 7.4. Condition evaluation

2840 The *condition* value SHALL be "True" if the <Condition> element is absent, or if it evaluates to  
2841 "True" for the *attribute* values supplied in the request *context*. Its value is "False" if the  
2842 <Condition> element evaluates to "False" for the *attribute* values supplied in the request  
2843 *context*. If any *attribute* value referenced in the *condition* cannot be obtained, then the *condition*  
2844 SHALL evaluate to "Indeterminate".

## 2845 7.5. Rule evaluation

2846 A *rule* has a value that can be calculated by evaluating its contents. *Rule* evaluation involves  
2847 separate evaluation of the *rule's target* and *condition*. The *rule* truth table is shown in Table 1.

Target	Condition	Rule Value
"Match"	"True"	Effect
"Match"	"False"	"Not-applicable"
"Match"	"Indeterminate"	"Indeterminate"
Not "Match"	Don't care	"Not-applicable"
"Indeterminate"	Don't care	"Indeterminate"

2848

Table 1 - Rule truth table

2849 If the **target** value is "No-match" or "Indeterminate" then the **rule** value SHALL be "Not-applicable"  
2850 or "Indeterminate", respectively, regardless of the value of the **condition**. For these cases,  
2851 therefore, the **condition** need not be evaluated in order to determine the **rule** value.

2852 If the **target** value is "Match" and the **condition** value is "True", then the **effect** specified in the **rule**  
2853 SHALL determine the **rule** value.

## 2854 7.6. Policy evaluation

2855 A **policy** has a value that can be calculated by evaluating its contents. **Policy** evaluation involves  
2856 separate evaluation of the **policy's target** and **rules**. 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 "Not-applicable"	"Not-applicable"
"Match"	At least one rule value is "Indeterminate"	Specified by the <b>rule-combining algorithm</b>
Not "Match"	Don't-care	"Not-applicable"
"Indeterminate"	Don't-care	"Indeterminate"

2857

Table 2 - Rule truth table

2858 A Rules value of "At-least-one-applicable" SHALL be used if the <Rule> element is absent, or if  
2859 one or more of the **rules** contained in the **policy** is applicable to the **decision request** (i.e., returns  
2860 a value of "Effect"; see Section 7.5). A value of "None-applicable" SHALL be used if no **rule**  
2861 contained in the **policy** is applicable to the request and if no **rule** contained in the **policy** returns a  
2862 value of "Indeterminate". If no **rule** contained in the **policy** is applicable to the request but one or  
2863 more **rule** returns a value of "Indeterminate", then **rules** SHALL evaluate to "Indeterminate".

2864 If the **target** value is "No-match" or "Indeterminate" then the **policy** value SHALL be "Not-  
2865 applicable" or "Indeterminate", respectively, regardless of the value of the **rules**. For these cases,  
2866 therefore, the **rules** need not be evaluated in order to determine the **policy** value.

2867 If the **target** value is "Match" and the **rules** value is "At-least-one-applicable" or "Indeterminate",  
2868 then the **rule-combining algorithm** specified in the **policy** SHALL determine the **policy** value.

## 2869 7.7. Policy Set evaluation

2870 A **policy set** has a value that can be calculated by evaluating its contents. **Policy set** evaluation  
2871 involves separate evaluation of the **policy set's target** and **policies**. The **policy set** truth table is  
2872 shown in Table 3.

Target	Policy values	Policy Set Value
Match	At least one policy value is its Effect	Specified by the <b>policy-combining algorithm</b>

Match	All policy values are "Not-applicable"	"Not-applicable"
Match	At least one policy value is "Indeterminate"	Specified by the <b>policy-combining algorithm</b>
Not match	Don't-care	"Not-applicable"
Indeterminate	Don't-care	"Indeterminate"

Table 3 - Rule truth table

2873

2874 A **policies** value of "At-least-one-applicable" SHALL be used if there are no contained or  
 2875 referenced **policies** or **policy sets**, or if one or more of the **policies** or **policy sets** contained in or  
 2876 referenced by the **policy set** is applicable to the **decision request** (i.e., returns a value determined  
 2877 by its **rule-combining algorithm**; see Section 7.6). A value of "None-applicable" SHALL be used if  
 2878 no **policy** or **policy set** contained in or referenced by the **policy set** is applicable to the request  
 2879 and if no **policy** or **policy set** contained in or referenced by the **policy set** returns a value of  
 2880 "Indeterminate". If no **policy** or **policy set** contained in or referenced by the **policy set** is  
 2881 applicable to the request but one or more **policy** or **policy set** returns a value of "Indeterminate",  
 2882 then **policies** SHALL evaluate to "Indeterminate".

2883 If the **target** value is "No-match" or "Indeterminate" then the **policy set** value SHALL be "Not-  
 2884 applicable" or "Indeterminate", respectively, regardless of the value of the **policies**. For these  
 2885 cases, therefore, the **policies** need not be evaluated in order to determine the **policy set** value.

2886 If the **target** value is "Match" and the **policies** value is "At-least-one-applicable" or "Indeterminate",  
 2887 then the **policy-combining algorithm** specified in the **policy set** SHALL determine the **policy set**  
 2888 value.

## 7.8. Hierarchical resources

2889

2890 It is often the case that a **resource** is organized as a hierarchy (e.g. file system, XML document).  
 2891 Some access requesters may request **access** to an entire subtree of a **resource** specified by a  
 2892 node. XACML allows the **PEP** (or **context handler**) to specify whether the **decision request** is  
 2893 just for a single **resource** or for a subtree below the specified **resource**. The latter is equivalent to  
 2894 repeating a single request for each node in the entire subtree. When a request **context** contains a  
 2895 resource attribute of type

2896 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2897 with a value of "Immediate", or if it does not contain that **attribute**, then the **decision request**  
 2898 SHALL be interpreted to apply to just the single **resource** specified by the **ResourceId attribute**.

2899 When the

2900 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2901 **attribute** has the value "Children", the **decision request** SHALL be interpreted to apply to the  
 2902 specified **resource** and its immediate children **resources**.

2903 When the

2904 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2905 **attribute** has the value "Descendants", the **decision request** SHALL be interpreted to apply to  
 2906 both the specified **resource** and all its descendant **resources**.

2907 In the case of "Children" and "Descendants", the **authorization decision** MAY include multiple  
2908 results for the multiple sub-nodes in the **resource** sub-tree.

2909 An XACML **authorization response** MAY contain multiple <Result> elements. In this case, the  
2910 <Status> element SHOULD be included only in the first <Result> element (the remaining  
2911 <Result> elements SHOULD NOT include the <Status> element).

2912 Note that the method by which the **PDP** discovers whether the **resource** is hierarchically organized  
2913 or not is outside the scope of XACML.

## 2914 **7.9. Attributes**

2915 **Attributes** are specified in the request **context** and are referred to in the **policy** by **subject**,  
2916 **resource**, **action** and **environment attribute** designators and **attribute** selectors. A **named**  
2917 **attribute** is the term used for the criteria that the specific **subject**, **resource**, **action** and  
2918 **environment attribute** designators and selectors use to refer to **attributes** in the **subject**,  
2919 **resource**, **action** and **environment** elements of the request **context**, respectively.

### 2920 **7.9.1. Attribute Matching**

2921 A **named attribute** has specific criteria with which to match **attributes** within the **context**. An  
2922 **attribute** specifies `AttributeId`, `DataType` and `Issuer` attributes, and each **named attribute**  
2923 also specifies `AttributeId`, `DataType` and optional `Issuer` attributes. A **named attribute**  
2924 SHALL match an **attribute** if the values of their respective `AttributeId`, `DataType` and optional  
2925 `Issuer` attributes match within their particular element, e.g. **subject**, **resource**, **action** or  
2926 **environment**, of the **context**. The `AttributeId` attribute of the named attribute MUST match, by  
2927 URI equality, the `AttributeId` attribute of the context **attribute** attribute. The `DataType`  
2928 attribute of the named attribute MUST match, by URI equality, the `DataType` attribute of the same  
2929 context **attribute**. If the `Issuer` attribute is supplied in the named attribute, then it MUST match,  
2930 by URI equality, the `Issuer` attribute of the same context **attribute**. If the `Issuer` attribute is not  
2931 supplied in the **named attribute**, then the matching of the context **attribute** to the **named attribute**  
2932 SHALL be governed by `AttributeId` and `DataType` attributes alone, regardless of the presence,  
2933 absence, or actual value of the `Issuer` attribute. In the case of an **attribute** selector, the matching  
2934 of the **attribute** to the **named attribute** SHALL be governed by the XPath expression, `DataType`  
2935 and `Issuer` attributes.

### 2936 **7.9.2. Attribute Retrieval**

2937 The **PDP** SHALL request the values of **attributes** in the request **context** from the **context handler**.  
2938 The **PDP** SHALL reference the **attributes** as if they were in a physical request **context** document,  
2939 but the **context handler** is responsible for obtaining and supplying the requested values. The  
2940 **context handler** SHALL return the values of **attributes** that match the **attribute** designator or  
2941 **attribute** selector and form them into a **bag** of values with the specified `DataType` attribute. If no  
2942 **attributes** from the request **context** match, then the **attribute** SHALL be considered missing. If  
2943 the **attribute** is missing, the `MustBePresent` attribute governs whether the **attribute**  
2944 designator or **attribute** selector returns an empty **bag** or an **indeterminate** result. If  
2945 `MustBePresent` is "False" (default value), then a missing attribute results in an empty **bag**. If  
2946 `MustBePresent` is "True", then a missing **attribute** results in "Indeterminate". This  
2947 "Indeterminate" result SHALL be handled in accordance with the specification of the encompassing  
2948 expressions, rules, policies, and policy sets. If the result is "Indeterminate", then the  
2949 `AttributeId`, `DataType` and `Issuer` of the **attribute** MAY be listed in the **authorization**  
2950 **decision** as described in Section 7.10. However, a **PDP** MAY choose not to return such  
2951 information for security reasons.

2952

### 7.9.3. Environment Attributes

2953 **Environment attributes** are listed in Section B.8. If a value for one of these **attributes** is supplied  
2954 in the **decision request**, then the **context handler** SHALL use that value. Otherwise, the **context**  
2955 **handler** SHALL supply a value. For the date and time **attributes**, the supplied value SHALL have  
2956 the semantics of "date and time that apply to the **decision request**".

2957

### 7.9.4. Subject Attributes

2958 The "subject-category" **attribute** is a *named attribute* with an AttributeId of  
2959 "urn:oasis:names:tc:xacml:1.0:subject:subject-category" and DataType attribute  
2960 of "http://www.w3.org/2001/XMLSchema#string". For each <Subject> element in the  
2961 **decision request**, if a value for the "subject-category" **attribute** is supplied, then the **context**  
2962 **handler** SHALL use that value. Otherwise, the **context handler** SHALL supply the default value  
2963 "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject". If there is  
2964 more than one "subject-category" **attribute** supplied in the **decision request** for any given  
2965 <Subject> element, then the **decision request** is invalid.

2966

## 7.10. Authorization decision

2967 Given a valid XACML **policy** or **policy set**, a compliant XACML **PDP** MUST evaluate the **policy** as  
2968 specified in Sections 5, 0 and 4.2. The **PDP** MUST return a response **context**, with one  
2969 <Decision> element of value "Permit", "Deny", "Indeterminate" or "Not-applicable".

2970 If the **PDP** cannot make a decision, then an "Indeterminate" <Decision> element contents SHALL  
2971 be returned. The **PDP** MAY return a <Decision> element contents of "Indeterminate" with a  
2972 status code of:

2973 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

2974 signifying that more information is needed. In this case, the <Status> element MAY list the  
2975 names and data-types of any **attributes** of the **subjects** and the **resource** that are needed by the  
2976 **PDP** to refine its decision. A **PEP** MAY resubmit a refined request **context** in response to a  
2977 <Decision> element contents of "Indeterminate" with a status code of

2978 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

2979 by adding **attribute** values for the **attribute** names that were listed in the previous response. When  
2980 the **PDP** returns a <Decision> element contents of "Indeterminate", with a status code of

2981 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

2982 it MUST NOT list the names and data-types of any **attribute** of the **subject** or the **resource** for  
2983 which values were supplied in the original request. Note, this requirement forces the **PDP** to  
2984 eventually return an **authorization decision** of "Permit", "Deny" or "Indeterminate" with some other  
2985 status code, in response to successively-refined requests.

2986

## 7.11. Obligations

2987 A **policy** or **policy set** may contain one or more **obligations**. When such a **policy** or **policy set** is  
2988 evaluated, an **obligation** SHALL be passed up to the next level of evaluation (the enclosing or  
2989 referencing **policy set** or **authorization decision**) only if the **effect** of the **policy** or **policy set**  
2990 being evaluated matches the value of the `xacml:FulfillOn` attribute of the **obligation**.  
2991

2992 As a consequence of this procedure, no **obligations** SHALL be returned to the **PEP** if the **policies**  
2993 or **policy sets** from which they are drawn are not evaluated, or if their evaluated result is  
2994 "Indeterminate" or "Not-applicable", or if the **decision** resulting from evaluating the **policy** or **policy**  
2995 **set** does not match the **decision** resulting from evaluating an enclosing **policy set**.  
2996  
2997 If the **PDP's** evaluation is viewed as a tree of **policy sets** and **policies**, each of which returns  
2998 "Permit" or "Deny", then the set of **obligations** returned by the **PDP** to the **PEP** will include only the  
2999 **obligations** associated with those paths where the **effect** at each level of evaluation is the same as  
3000 the **effect** being returned by the **PDP**.  
3001 A **PEP** that receives a valid XACML response of "Permit" with **obligations** SHALL be responsible  
3002 for fulfilling *all* of those **obligations**. A **PEP** that receives an XACML response of "Deny" with  
3003 **obligations** SHALL be responsible for fulfilling all of the **obligations** that it *understands*.

---

## 3004 8. XACML extensibility points (non-normative)

3005 This section describes the points within the XACML model and schema where extensions can be  
3006 added

### 3007 8.1. Extensible XML attribute types

3008 The following XML attributes have values that are URIs or QNames. These may be extended by  
3009 the creation of new URIs or QNames associated with new semantics for these attributes.

3010 AttributeId,

3011 AttributeValue,

3012 DataType,

3013 FunctionId,

3014 MatchId,

3015 ObligationId,

3016 PolicyCombiningAlgId,

3017 RuleCombiningAlgId,

3018 StatusCode.

3019 See Section 5 for definitions of these attribute types.

### 3020 8.2. Extensible XACML attribute types

3021 The following XACML standard `AttributeIds` associated with the XACML standard element:  
3022 `<Attribute>` have values that are URIs or QNames. These may be extended by the creation of  
3023 new URIs or QNames associated with new semantics for these attributes.

3024 "urn:oasis:names:tc:xacml:1.0:subject:subject-category".

3025

### 8.3. Structured attributes

3026

An XACML <AttributeValue> element MAY contain an instance of a structured XML data-type.

3027

Section A.3 describes a number of standard techniques to identify data items within such a

3028

structured attribute. Listed here are some additional techniques that require XACML extensions.

3029

1. For a given structured data type, a community of XACML users MAY define new attribute identifiers for each leaf sub-element of the structured data type that has a type conformant with one of the XACML-defined primitive data-types. Using these new attribute identifiers, the **PEPs** or **context handlers** used by that community of users can flatten instances of the structured data-type into a sequence of individual <Attribute> elements. Each such <Attribute> element can be compared using the XACML-defined functions. Using this method, the structured data type itself never appears in an <AttributeValue> element.

3030

3031

3032

3033

3034

3035

3036

2. A community of XACML users MAY define a new function that can be used to compare a value of the structured data-type against some other value. This method may only be used by **PDPs** that support the new function.

3037

3038

---

3039

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

3040

3041

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

3042

3043

3044

3045

### 9.1. Threat model

3046

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

3047

3048

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

3049

3050

3051

3052

3053

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

3054

3055

3056

3057

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

3058

3059

3060

3061

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

3062

3063

### 9.1.1. Unauthorized disclosure

3064 XACML does not specify any inherent mechanisms for confidentiality of the messages exchanged  
3065 between actors. Therefore, an adversary could observe the messages in transit. Under certain  
3066 security policies, disclosure of this information is a violation. Disclosure of **attributes** or the types  
3067 of **decision requests** that a **subject** submits may be a breach of privacy policy. In the commercial  
3068 sector, the consequences of unauthorized disclosure of personal data may range from  
3069 embarrassment to the custodian to imprisonment and large fines in the case of medical or financial  
3070 data.

3071 Unauthorized disclosure is addressed by confidentiality mechanisms.

3072

### 9.1.2. Message replay

3073 A message replay attack is one in which the adversary records and replays legitimate messages  
3074 between XACML actors. This attack may lead to denial of service, the use of out-of-date  
3075 information or impersonation.

3076 Prevention of replay attacks requires the use of message freshness mechanisms.

3077 Note that encryption of the message does not mitigate a replay attack since the message is just  
3078 replayed and does not have to be understood by the adversary.

3079

### 9.1.3. Message insertion

3080 A message insertion attack is one in which the adversary inserts messages in the sequence of  
3081 messages between XACML actors.

3082 The solution to a message insertion attack is to use mutual authentication and a message  
3083 sequence integrity mechanism between the actors. It should be noted that just using SSL mutual  
3084 authentication is not sufficient. This only proves that the other party is the one identified by the  
3085 subject of the X.509 certificate. In order to be effective, it is necessary to confirm that the certificate  
3086 subject is authorized to send the message.

3087

### 9.1.4. Message deletion

3088 A message deletion attack is one in which the adversary deletes messages in the sequence of  
3089 messages between XACML actors. Message deletion may lead to denial of service. However, a  
3090 properly designed XACML system should not render an incorrect authorization decision as a result  
3091 of a message deletion attack.

3092 The solution to a message deletion attack is to use a message integrity mechanism between the  
3093 actors.

3094

### 9.1.5. Message modification

3095 If an adversary can intercept a message and change its contents, then they may be able to alter an  
3096 **authorization decision**. Message integrity mechanisms can prevent a successful message  
3097 modification attack.

3098

### 9.1.6. Not-applicable results

3099 A result of "Not-applicable" means that the **PDP** did not have a policy whose target matched the  
3100 information in the **decision request**. In general, we highly recommend using a "default-deny"

3101 policy, so that when a **PDP** would have returned "Not-applicable", a result of "Deny" is returned  
3102 instead.

3103 In some security models, however, such as is common in many Web Servers, a result of "Not-  
3104 applicable" is treated as equivalent to "Permit". There are particular security considerations that  
3105 must be taken into account for this to be safe. These are explained in the following paragraphs.

3106 If "Not-applicable" is to be treated as "Permit", it is vital that the matching algorithms used by the  
3107 policy to match elements in the decision request are closely aligned with the data syntax used by  
3108 the applications that will be submitting the decision request. A failure to match will be treated as  
3109 "Permit", so an unintended failure to match may allow unintended access.

3110 A common example of this is a Web Server. Commercial http responders allow a variety of  
3111 syntaxes to be treated equivalently. The "%" can be used to represent characters by hex value.  
3112 The URL path "../" provides multiple ways of specifying the same value. Multiple character sets  
3113 may be permitted and, in some cases, the same printed character can be represented by different  
3114 binary values. Unless the matching algorithm used by the policy is sophisticated enough to catch  
3115 these variations, unintended access may be permitted.

3116 It is safe to treat "Not-applicable" as "Permit" only in a closed environment where all applications  
3117 that formulate a decision request can be guaranteed to use the exact syntax expected by the  
3118 policies used by the **PDP**. In a more open environment, where decision requests may be received  
3119 from applications that may use any legal syntax, it is strongly recommended that "Not-applicable"  
3120 NOT be treated as "Permit" unless matching rules have been very carefully designed to match all  
3121 possible applicable inputs, regardless of syntax or type variations.

### 3122 **9.1.7. Negative rules**

3123 A negative **rule** is one that is based on a **predicate** not being "True". If not used with care,  
3124 negative **rules** can lead to policy violation, therefore some authorities recommend that they not be  
3125 used. However, negative **rules** can be extremely efficient in certain cases, so XACML has chosen  
3126 to include them. Nevertheless, it is recommended that they be used with care and avoided if  
3127 possible.

3128 A common use for negative **rules** is to deny **access** to an individual or subgroup when their  
3129 membership in a larger group would otherwise permit them access. For example, we might want to  
3130 write a **rule** that allows all Vice Presidents to see the unpublished financial data, except for Joe,  
3131 who is only a Ceremonial Vice President and can be indiscreet in his communications. If we have  
3132 complete control of the administration of **subject attributes**, a superior approach would be to  
3133 define "Vice President" and "Ceremonial Vice President" as distinct groups and then define **rules**  
3134 accordingly. However, in some environments this approach may not be feasible. (It is worth noting  
3135 in passing that, generally speaking, referring to individuals in **rules** does not scale well. Generally,  
3136 shared **attributes** are preferred.)

3137 If not used with care, negative **rules** can lead to policy violation in two common cases. They are:  
3138 when **attributes** are suppressed and when the base group changes. An example of suppressed  
3139 **attributes** would be if we have a policy that **access** should be permitted, *unless* the **subject** is a  
3140 credit risk. If it is possible that the **attribute** of being a credit risk may be unknown to the **PDP** for  
3141 some reason, then unauthorized **access** may be permitted. In some environments, the **subject**  
3142 may be able to suppress the publication of **attributes** by the application of privacy controls, or the  
3143 server or repository that contains the information may be unavailable for accidental or intentional  
3144 reasons.

3145 An example of a changing base group would be if there is a policy that everyone in the engineering  
3146 department may change software source code, except for secretaries. Suppose now that the  
3147 department was to merge with another engineering department and the intent is to maintain the  
3148 same policy. However, the new department also includes individuals identified as administrative

3149 assistants, who ought to be treated in the same way as secretaries. Unless the policy is altered,  
3150 they will unintentionally be permitted to change software source code. Problems of this type are  
3151 easy to avoid when one individual administers all **policies**, but when administration is distributed,  
3152 as XACML allows, this type of situation must be explicitly guarded against.

## 3153 **9.2. Safeguards**

### 3154 **9.2.1. Authentication**

3155 Authentication provides the means for one party in a transaction to determine the identity of the  
3156 other party in the transaction. Authentication may be in one direction, or it may be bilateral.

3157 Given the sensitive nature of **access control** systems, it is important for a **PEP** to authenticate the  
3158 identity of the **PDP** to which it sends **decision requests**. Otherwise, there is a risk that an  
3159 adversary could provide false or invalid **authorization decisions**, leading to a policy violation.

3160 It is equally important for a **PDP** to authenticate the identity of the **PEP** and assess the level of trust  
3161 to determine what, if any, sensitive data should be passed. One should keep in mind that even  
3162 simple "Permit" or "Deny" responses could be exploited if an adversary were allowed to make  
3163 unlimited requests to a **PDP**.

3164 Many different techniques may be used to provide authentication, such as co-located code, a  
3165 private network, a VPN or digital signatures. Authentication may also be performed as part of the  
3166 communication protocol used to exchange the **contexts**. In this case, authentication may be  
3167 performed at the message level or at the session level.

### 3168 **9.2.2. Policy administration**

3169 If the contents of **policies** are exposed outside of the **access control** system, potential **subjects**  
3170 may use this information to determine how to gain unauthorized **access**.

3171 To prevent this threat, the repository used for the storage of **policies** may itself require **access**  
3172 **control**. In addition, the <Status> element should be used to return values of missing **attributes**  
3173 only when exposure of the identities of those **attributes** will not compromise security.

### 3174 **9.2.3. Confidentiality**

3175 Confidentiality mechanisms ensure that the contents of a message can be read only by the desired  
3176 recipients and not by anyone else who encounters the message while it is in transit. There are two  
3177 areas in which confidentiality should be considered: one is confidentiality during transmission; the  
3178 other is confidentiality within a <Policy> element.

#### 3179 **9.2.3.1. Communication confidentiality**

3180 In some environments it is deemed good practice to treat all data within an **access control** system  
3181 as confidential. In other environments, **policies** may be made freely available for distribution,  
3182 inspection and audit. The idea behind keeping **policy** information secret is to make it more difficult  
3183 for an adversary to know what steps might be sufficient to obtain unauthorized **access**. Regardless  
3184 of the approach chosen, the security of the **access control** system should not depend on the  
3185 secrecy of the **policy**.

3186 Any security concerns or requirements related to transmitting or exchanging XACML <policy>  
3187 elements are outside the scope of the XACML standard. While it is often important to ensure that  
3188 the integrity and confidentiality of <policy> elements is maintained when they are exchanged

3189 between two parties, it is left to the implementers to determine the appropriate mechanisms for their  
3190 environment.

3191 Communications confidentiality can be provided by a confidentiality mechanism, such as SSL.  
3192 Using a point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points  
3193 is compromised.

### 3194 **9.2.3.2. Statement level confidentiality**

3195 In some cases, an implementation may want to encrypt only parts of an XACML <Policy>  
3196 element.

3197 The XML Encryption Syntax and Processing Candidate Recommendation from W3C can be used  
3198 to encrypt all or parts of an XML document. This specification is recommended for use with  
3199 XACML.

3200 It should go without saying that if a repository is used to facilitate the communication of cleartext  
3201 (i.e., unencrypted) **policy** between the **PAP** and **PDP**, then a secure repository should be used to  
3202 store this sensitive data.

### 3203 **9.2.4. Policy integrity**

3204 The XACML **policy**, used by the **PDP** to evaluate the request **context**, is the heart of the system.  
3205 Therefore, maintaining its integrity is essential. There are two aspects to maintaining the integrity of  
3206 the **policy**. One is to ensure that <Policy> elements have not been altered since they were  
3207 originally created by the **PAP**. The other is to ensure that <Policy> elements have not been  
3208 inserted or deleted from the set of **policies**.

3209 In many cases, both aspects can be achieved by ensuring the integrity of the actors and  
3210 implementing session-level mechanisms to secure the communication between actors. The  
3211 selection of the appropriate mechanisms is left to the implementers. However, when **policy** is  
3212 distributed between organizations to be acted on at a later time, or when the **policy** travels with the  
3213 protected resource, it would be useful to sign the **policy**. In these cases, the XML Signature  
3214 Syntax and Processing standard from W3C is recommended to be used with XACML.

3215 Digital signatures should only be used to ensure the integrity of the statements. Digital signatures  
3216 should not be used as a method of selecting or evaluating **policy**. That is, the **PDP** should not  
3217 request a **policy** based on who signed it or whether or not it has been signed (as such a basis for  
3218 selection would, itself, be a matter of policy). However, the **PDP** must verify that the key used to  
3219 sign the **policy** is one controlled by the purported issuer of the **policy**. The means to do this are  
3220 dependent on the specific signature technology chosen and are outside the scope of this document.

### 3221 **9.2.5. Policy identifiers**

3222 Since **policies** can be referenced by their identifiers, it is the responsibility of the **PAP** to ensure  
3223 that these are unique. Confusion between identifiers could lead to misidentification of the  
3224 **applicable policy**. This specification is silent on whether a **PAP** must generate a new identifier  
3225 when a **policy** is modified or may use the same identifier in the modified **policy**. This is a matter of  
3226 administrative practice. However, care must be taken in either case. If the identifier is reused,  
3227 there is a danger that other **policies** or **policy sets** that reference it may be adversely affected.  
3228 Conversely, if a new identifier is used, these other **policies** may continue to use the prior **policy**,  
3229 unless it is deleted. In either case the results may not be what the **policy** administrator intends.

3230

## 9.2.6. Trust model

3231 Discussions of authentication, integrity and confidentiality mechanisms necessarily assume an  
3232 underlying trust model: how can one actor come to believe that a given key is uniquely associated  
3233 with a specific, identified actor so that the key can be used to encrypt data for that actor or verify  
3234 signatures (or other integrity structures) from that actor? Many different types of trust model exist,  
3235 including strict hierarchies, distributed authorities, the Web, the bridge and so on.

3236 It is worth considering the relationships between the various actors of the **access control** system in  
3237 terms of the interdependencies that do and do not exist.

- 3238 • None of the entities of the authorization system are dependent on the **PEP**. They may  
3239 collect data from it, for example authentication, but are responsible for verifying it.
- 3240 • The correct operation of the system depends on the ability of the **PEP** to actually enforce  
3241 **policy** decisions.
- 3242 • The **PEP** depends on the **PDP** to correctly evaluate **policies**. This in turn implies that the  
3243 **PDP** is supplied with the correct inputs. Other than that, the **PDP** does not depend on the  
3244 **PEP**.
- 3245 • The **PDP** depends on the **PAP** to supply appropriate policies. The **PAP** is not dependent  
3246 on other components.

3247

## 9.2.7. Privacy

3248 It is important to be aware that any transactions that occur with respect to **access control** may  
3249 reveal private information about the actors. For example, if an XACML **policy** states that certain  
3250 data may only be read by **subjects** with "Gold Card Member" status, then any transaction in which  
3251 a **subject** is permitted **access** to that data leaks information to an adversary about the **subject's**  
3252 status. Privacy considerations may therefore lead to encryption and/or to **access control policies**  
3253 surrounding the enforcement of XACML **policy** instances themselves: confidentiality-protected  
3254 channels for the request/response protocol messages, protection of **subject attributes** in storage  
3255 and in transit, and so on.

3256 Selection and use of privacy mechanisms appropriate to a given environment are outside the scope  
3257 of XACML. The decision regarding whether, how and when to deploy such mechanisms is left to  
3258 the implementers associated with the environment.

---

# 10. Conformance (normative)

3259

3260

## 10.1. Introduction

3261 The XACML specification addresses two aspects of conformance:

3262 1. The OASIS procedure for ratification of a committee specification as an OASIS standard requires  
3263 that three independent implementers attest that they are "successfully using" the committee  
3264 specification, and

3265 2. The XACML specification defines a number of functions, etc. that have somewhat specialist  
3266 application, therefore they are not required to be implemented in an implementation that claims to  
3267 conform with the OASIS standard.

3268

## 10.2. Attestation

3269 An implementer MAY attest to be "successfully using" the XACML committee specification provided  
 3270 the implementation successfully executes a set of test-cases. The test cases are hosted by Sun  
 3271 Microsystems and can be located from the XACML web page. The site hosting the test cases  
 3272 contains a full description of the test cases and how to execute them.

3273

## 10.3. Conformance tables

3274 This section lists those portions of the specification that MUST be included in an implementation of  
 3275 a **PDP** that claims to conform with XACML v1.0.

3276 Note: "M" means mandatory-to-implement. "O" means optional.

3277

### 10.3.1. Schema elements

3278 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	O
xacml:Context:StatusCode	O
xacml:Context:StatusDetail	O
xacml:Context:StatusMessage	O
xacml:Context:Subject	M
xacml:Policy:Action	M
xacml:Policy:ActionAttributeDesignator	M
xacml:Policy:ActionMatch	M
xacml:Policy:Actions	M
xacml:Policy:AnyAction	M
xacml:Policy:AnyResource	M
xacml:Policy:AnySubject	M
xacml:Policy:Apply	M
xacml:Policy:AttributeAssignment	O
xacml:Policy:AttributeSelector	O
xacml:Policy:AttributeValue	M
xacml:Policy:Condition	M
xacml:Policy:Description	M
xacml:Policy:EnvironmentAttributeDesignator	M
xacml:Policy:Function	M
xacml:Policy:Obligation	O
xacml:Policy:Obligations	O
xacml:Policy:Policy	M
xacml:Policy:PolicyIdReference	M
xacml:Policy:PolicySet	M
xacml:Policy:PolicySetDefaults	O

xacml:Policy:PolicySetIdReference	M
xacml:Policy:Resource	M
xacml:Policy:ResourceAttributeDesignator	M
xacml:Policy:ResourceMatch	M
xacml:Policy:Resources	M
xacml:Policy:Rule	M
xacml:Policy:Subject	M
xacml:Policy:SubjectMatch	M
xacml:Policy:Subjects	M
xacml:Policy:Target	M
xacml:Policy:XpathVersion	M

3279 **10.3.2. Identifier Prefixes**

3280 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

3281 **10.3.3. Algorithms**

3282 The implementation MUST include the rule- and policy-combining algorithms associated with the  
3283 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

3284 **10.3.4. Status Codes**

3285 Implementation support for the urn:oasis:names:tc:xacml:1.0:context:status element is optional, but  
3286 if the element is supported, then the following status codes must be supported and must be used in  
3287 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

3288                    **10.3.5.        Attributes**

3289        The implementation MUST support the attributes associated with the following attribute identifiers  
3290        as specified by XACML. The value for these attributes MUST be provided by the **PDP**, so, unlike  
3291        most other attributes, their semantics are not transparent to the **PDP** implementation.

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
Urn:oasis:names:tc:xacml:1.0:subject:subject-category	M

3292                    **10.3.6.        Identifiers**

3293        The implementation MUST use the attributes associated with the following identifiers in the way  
3294        XACML has defined. This requirement pertains primarily to implementations of a **PAP** or **PEP** that  
3295        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:resource:resource-location	O
Urn:oasis:names:tc:xacml:1.0:resource:resource-id	O
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
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

3296                    **10.3.7.        Data Types**

3297        The implementation MUST support the data types associated with the following identifiers marked  
3298        "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#date	M
http://www.w3.org/2001/XMLSchema#dateTime	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
http://www.w3.org/TR/xquery-operators:dayTimeDuration	M
http://www.w3.org/TR/xquery-operators:yearMonthDuration	M
Urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name	M
Urn:oasis:names:tc:xacml:1.0:data-type:x500Name	M

3299                    **10.3.8.        Functions**

3300        The implementation MUST properly process those functions associated with the identifiers marked  
3301        with an "M".

3302        xmlns:function="urn:oasis:names:tc:xacml:1.0:function"

Function	M/O
function:string-equal	M
function:boolean-equal	M
function:integer-equal	M
function:double-equal	M
function:date-equal	M
function:time-equal	M
function:dateTime-equal	M
function:anyURI-equal	M
function:x500Name-equal	M
function:rfc822name-equal	M
function:hexBinary-equal	M
function:base64Binary-equal	M
function:integer-add	M
function:double-add	M
function:integer-subtract	M
function:double-subtract	M
function:integer-multiply	M
function:double-multiply	M
function:integer-divide	M
function:double-divide	M
function:integer-mod	M
function:integer-abs	M
function:double-abs	M
function:round	M
function:floor	M
function:string-normalize-space	M
function:string-normalize-to-lower-case	M
function:double-to-integer	M
function:integer-to-double	M
function:or	M
function:and	M
function:n-of	M
function:not	M
function:present	M
function:integer-greater-than	M
function:integer-greater-than-or-equal	M
function:integer-less-than	M
function:integer-less-than-or-equal	M
function:double-greater-than	M
function:double-greater-than-or-equal	M
function:double-less-than	M
function:double-less-than-or-equal	M
function:dateTime-add-dayTimeDuration	M

function:dateTime-add-yearMonthDuration	M
function:dateTime-subtract-dayTimeDuration	M
function:dateTime-subtract-yearMonthDuration	M
function:date-add-yearMonthDuration	M
function:date-subtract-yearMonthDuration	M
function:string-greater-than	M
function:string-greater-than-or-equal	M
function:string-less-than	M
function:string-less-than-or-equal	M
function:time-greater-than	M
function:time-greater-than-or-equal	M
function:time-less-than	M
function:time-less-than-or-equal	M
function:dateTime-greater-than	M
function:dateTime-greater-than-or-equal	M
function:dateTime-less-than	M
function:dateTime-less-than-or-equal	M
function:date-greater-than	M
function:date-greater-than-or-equal	M
function:date-less-than	M
function:date-less-than-or-equal	M
function:string-one-and-only	M
function:string-bag-size	M
function:string-is-in	M
function:string-bag	M
function:boolean-one-and-only	M
function:boolean-bag-size	M
function:boolean-is-in	M
function:boolean-bag	M
function:integer-one-and-only	M
function:integer-bag-size	M
function:integer-is-in	M
function:integer-bag	M
function:double-one-and-only	M
function:double-bag-size	M
function:double-is-in	M
function:double-bag	M
function:date-one-and-only	M
function:date-bag-size	M
function:date-is-in	M
function:date-bag	M
function:dateTime-one-and-only	M
function:dateTime-bag-size	M
function:dateTime-is-in	M
function:dateTime-bag	M
function:anyURI-one-and-only	M
function:anyURI-bag-size	M
function:anyURI-is-in	M
function:anyURI-bag	M
function:hexBinary-one-and-only	M
function:hexBinary-bag-size	M
function:hexBinary-is-in	M
function:hexBinary-bag	M
function:base64Binary-one-and-only	M
function:base64Binary-bag-size	M
function:base64Binary-is-in	M
function:base64Binary-bag	M

function:dayTimeDuration-one-and-only	M
function:dayTimeDuration-bag-size	M
function:dayTimeDuration-is-in	M
function:dayTimeDuration-bag	M
function:yearMonthDuration-one-and-only	M
function:yearMonthDuration-bag-size	M
function:yearMonthDuration-is-in	M
function:yearMonthDuration-bag	M
function:x500Name-one-and-only	M
function:x500Name-bag-size	M
function:x500Name-is-in	M
function:x500Name-bag	M
function:rfc822Name-one-and-only	M
function:rfc822Name-bag-size	M
function:rfc822Name-is-in	M
function:rfc822Name-bag	M
function:any-of	M
function:all-of	M
function:any-of-any	M
function:all-of-any	M
function:any-of-all	M
function:all-of-all	M
function:map	M
function:x500Name-match	M
function:rfc822Name-match	M
function:xpath-node-count	O
function:xpath-node-equal	O
function:xpath-node-match	O

## 3303 11. References

- 3304 [DS] D. Eastlake et al., *XML-Signature Syntax and Processing*,  
3305 <http://www.w3.org/TR/xmldsig-core/>, World Wide Web Consortium.
- 3306 [Haskell] Haskell, a purely functional language. Available at  
3307 <http://www.haskell.org/>
- 3308 [Hinton94] Hinton, H, M, Lee,, E, S, The Compatibility of Policies, Proceedings 2nd  
3309 ACM Conference on Computer and Communications Security, Nov 1994,  
3310 Fairfax, Virginia, USA.
- 3311 [IEEE754] IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-  
3312 7653-8, IEEE Product No. SH10116-TBR
- 3313 [Kudo00] Kudo M and Hada S, XML document security based on provisional  
3314 authorization, Proceedings of the Seventh ACM Conference on Computer  
3315 and Communications Security, Nov 2000, Athens, Greece, pp 87-96.
- 3316 [LDAP-1] RFC2256, A summary of the X500(96) User Schema for use with LDAPv3,  
3317 section 5, M Wahl, December 1997 <http://www.ietf.org/rfc/rfc2798.txt>
- 3318 [LDAP-2] RFC2798, Definition of the inetOrgPerson, M. Smith, April 2000  
3319 <http://www.ietf.org/rfc/rfc2798.txt>
- 3320 [MathML] Mathematical Markup Language (MathML), Version 2.0, W3C  
3321 Recommendation, 21 February 2001. Available at:  
3322 <http://www.w3.org/TR/MathML2/>
- 3323 [Perritt93] Perritt, H. Knowbots, Permissions Headers and Contract Law, Conference  
3324 on Technological Strategies for Protecting Intellectual Property in the  
3325 Networked Multimedia Environment, April 1993. Available at:  
3326 <http://www.ifla.org/documents/infopol/copyright/perh2.txt>

3327	<b>[RBAC]</b>	Role-Based Access Controls, David Ferraiolo and Richard Kuhn, 15th National Computer Security Conference, 1992. Available at: <a href="http://csrc.nist.gov/rbac">http://csrc.nist.gov/rbac</a>
3328		
3329		
3330	<b>[RegEx]</b>	XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, Appendix D. Available at: <a href="http://www.w3.org/TR/xmlschema-0/">http://www.w3.org/TR/xmlschema-0/</a>
3331		
3332	<b>[RFC2119]</b>	S. Bradner, <i>Key words for use in RFCs to Indicate Requirement Levels</i> , <a href="http://www.ietf.org/rfc/rfc2119.txt">http://www.ietf.org/rfc/rfc2119.txt</a> , IETF RFC 2119, March 1997
3333		
3334	<b>[SAML]</b>	Security Assertion Markup Language available from <a href="http://www.oasis-open.org/committees/security/#documents">http://www.oasis-open.org/committees/security/#documents</a>
3335		
3336	<b>[Sloman94]</b>	Sloman, M. Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Volume 2, part 4. Plenum Press. 1994.
3337		
3338		
3339	<b>[XF]</b>	XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft 16 August 2002. Available at: <a href="http://www.w3.org/TR/xquery-operators">http://www.w3.org/TR/xquery-operators</a>
3340		
3341	<b>[XS]</b>	XML Schema. Available at: <a href="http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/">http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/</a>
3342		
3343	<b>[XPath]</b>	XML Path Language (XPath), Version 1.0, W3C Recommendation 16 November 1999. Available at: <a href="http://www.w3.org/TR/xpath">http://www.w3.org/TR/xpath</a>
3344		
3345	<b>[XSLT]</b>	XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999. Available at: <a href="http://www.w3.org/TR/xslt">http://www.w3.org/TR/xslt</a>
3346		
3347		

---

3348 **Appendix A. Standard data types, functions and**  
3349 **their semantics (normative)**

3350 **A.1. Introduction**

3351 This section contains a specification of the data-types and functions used in XACML to create  
3352 **predicates** for a **rule's condition** and **target** matches.

3353 This specification combines the various standards set forth by IEEE and ANSI for string  
3354 representation of numeric values, as well as the evaluation of arithmetic functions.

3355 This section describes the primitive data-types, **bags** and construction of expressions using  
3356 XACML constructs. Finally, each standard function is named and its operational semantics are  
3357 described.

3358 **A.2. Primitive types**

3359 Although XML instances represent all data-types as strings, an XACML **PDP** must reason about  
3360 types of data that, while they have string representations, are not just strings. Types such as  
3361 boolean, integer and double **MUST** be converted from their XML string representations to values  
3362 that can be compared with values in their domain of discourse, such as numbers. The following  
3363 primitive data-types are specified for use with XACML and have explicit data representations:

- 3364 • <http://www.w3.org/2001/XMLSchema#string>
- 3365 • <http://www.w3.org/2001/XMLSchema#boolean>
- 3366 • <http://www.w3.org/2001/XMLSchema#integer>
- 3367 • <http://www.w3.org/2001/XMLSchema#double>
- 3368 • <http://www.w3.org/2001/XMLSchema#date>
- 3369 • <http://www.w3.org/2001/XMLSchema#dateTime>
- 3370 • <http://www.w3.org/2001/XMLSchema#anyURI>
- 3371 • <http://www.w3.org/2001/XMLSchema#hexBinary>
- 3372 • <http://www.w3.org/2001/XMLSchema#base64Binary>
- 3373 • <http://www.w3.org/TR/xquery-operators#dayTimeDuration>
- 3374 • <http://www.w3.org/TR/xquery-operators#yearMonthDuration>
- 3375 • <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
- 3376 • <urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name>

## 3377 A.3. Structured types

3378 An XACML `<AttributeValue>` element MAY contain an instance of a structured XML data-type,  
3379 for example `<ds:KeyInfo>`. XACML 1.0 supports several ways for comparing such  
3380 `<AttributeValue>` elements.

3381 1. In some cases, such an `<AttributeValue>` element MAY be compared using one of the  
3382 XACML string functions, such as “`regexp-string-match`”, described below. This requires  
3383 that the structured data `<AttributeValue>` to be given the `DataType="xsi:string"`. For  
3384 example, a structured data type that is actually a `ds:KeyInfo/KeyName` would appear in the  
3385 Context as:

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

3386 In general, this method will not be adequate unless the structured data type is quite simple.

3387 2. An `<AttributeSelector>` element MAY be used to select the value of a leaf sub-  
3388 element of the structured data-type by means of an XPath expression. That value MAY  
3389 then be compared using one of the supported XACML functions appropriate for its primitive  
3390 data-type. This method requires support by the PDP for the optional XPath expressions  
3391 feature.

3392 3. An `<AttributeSelector>` element MAY be used to select the value of any node in the  
3393 structured type by means of an XPath expression. This node MAY then be compared  
3394 using one of the XPath-based functions described in Section A14.13. This method requires  
3395 support by the PDP for the optional XPath expressions and XPath functions features.

## 3396 A.4. Representations

3397 An XACML *PDP* SHALL be capable of converting string representations into various primitive data  
3398 types. For integers and doubles, XACML SHALL use the conversions described in IBM Standard  
3399 Decimal Arithmetic [IBMDSA].

3400 This document combines the various standards set forth by IEEE and ANSI for string  
3401 representation of numeric values.

3402 XACML defines two additional data-types; these are “`urn:oasis:names:tc:xacml:1.0:data-`  
3403 `type:x500Name`” and “`urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name`”. These types  
3404 represent identifiers for **subjects** and appear in several standard applications, such as TLS/SSL  
3405 and electronic mail.

3406 The “`urn:oasis:names:tc:xacml:1.0:data-type:x500Name`” primitive type represents an X.500  
3407 Distinguished Name. The string representation of an X.500 distinguished name is specified in IETF  
3408 RFC 2253 “Lightweight Directory Access Protocol (v3): UTF-8 String Representation of  
3409 Distinguished Names”.<sup>1</sup>

---

1 An earlier RFC, RFC 1779 “A String Representation of Distinguished Names”, is less restrictive, so `xacml:x500Name` uses the syntax in RFC 2253 for better interoperability.

3410 The “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name” primitive type represents electronic mail  
3411 addresses, and its string representation is specified by RFC 822.

3412 An RFC822 name consists of a *local-part* followed by “@” followed by a *domain-part*. The *local-*  
3413 *part* is case-sensitive, while the *domain-part* (which is usually a DNS host name) is not case-  
3414 sensitive.<sup>2</sup>

## 3415 A.5. Bags

3416 XACML defines implicit collections of its primitive types. XACML refers to a collection of values that  
3417 are of a single primitive type as a **bag**. **Bags** of primitive types are needed because selections of  
3418 nodes from an XML **resource** or XACML request **context** may return more than one value.

3419 The <AttributeSelector> element uses an XPath expression to specify the selection of data  
3420 from an XML **resource**. The result of an XPath expression is termed a *node-set*, which contains all  
3421 the leaf nodes from the XML **resource** that match the predicate in the XPath expression. Based on  
3422 the various indexing functions provided in the XPath specification, it SHALL be implied that a  
3423 resultant node-set is the collection of the matching nodes. XACML also defines the  
3424 <AttributeDesignator> **element** to have the same matching methodology for attributes in the  
3425 XACML request **context**.

3426 The values in a **bag** are not ordered, and some of the values may be duplicates. There SHALL be  
3427 no notion of a **bag** containing **bags**, or a **bag** containing values of differing types. I.e. a **bag** in  
3428 XACML SHALL contain only values that are of the same primitive type.

## 3429 A.6. Expressions

3430 XACML specifies expressions in terms of the following elements. Each expression evaluates to  
3431 one of the primitive types, or a **bag** of one of the primitive types. In addition, XACML defines an  
3432 evaluation result of “Indeterminate”, which is said to be the result of an invalid expression, or an  
3433 operational error occurring during the evaluation of the expression.

3434 XACML defines the following elements to be legal XACML expressions:

- 3435 • <AttributeValue>
- 3436 • <SubjectAttributeDesignator>
- 3437 • <SubjectAttributeSelector>
- 3438 • <ResourceAttributeDesignator>
- 3439 • <ActionAttributeDesignator>
- 3440 • <EnvironmentAttributeDesignator>
- 3441 • <AttributeSelector>

---

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.

- 3442 • <Apply>
- 3443 • <Condition>
- 3444 • <Function>

## 3445 A.7. Element <AttributeValue>

3446 The <AttributeValue> element SHALL represent an explicit value of a primitive type. For  
3447 example:

```
3448 <Apply FunctionId="function:integer-equal">  
3449   <AttributeValue  
3450     DataType="http://www.w3.org/2001/XMLSchema#integer">123</AttributeValue>  
3451   <AttributeValue  
3452     DataType="http://www.w3.org/2001/XMLSchema#integer">123</AttributeValue>  
3453 </Apply>
```

## 3454 A.8. Elements <AttributeDesignator> and 3455 <AttributeSelector>

3456 The <AttributeDesignator> and <AttributeSelector> elements SHALL evaluate to a **bag**  
3457 of a specific primitive type. The type SHALL be inferred from the function in which it appears. Each  
3458 element SHALL contain a URI or XPath expression, respectively, to identify the required **attribute**  
3459 values. If an operational error were to occur while finding the values, the value of the element  
3460 SHALL be set to "Indeterminate". If the required **attribute** cannot be located, then the value of the  
3461 element SHALL be set to an empty **bag** of the inferred primitive type.

## 3462 A.9. Element <Apply>

3463 XACML function calls are represented by the <Apply> element. The function to be applied is  
3464 named in the `FunctionId` attribute of this element. The value of the <Apply> element SHALL be  
3465 set to either a primitive type or a **bag** of a primitive type, whose type SHALL be inferred from the  
3466 `functionId`. The arguments of a function SHALL be the values of the XACML expressions that  
3467 are contained as ordered elements in an <Apply> element. The legal number of arguments within  
3468 an <Apply> element SHALL depend upon the `functionId`.

## 3469 A.10. Element <Condition>

3470 The <Condition> element MAY appear in the <Rule> element as the premise for emitting the  
3471 corresponding **effect** of the **rule**. The <Condition> element has the same structure as the  
3472 <Apply> element, with the restriction that its result SHALL be of type  
3473 "http://www.w3.org/2001/XMLSchema#boolean". The evaluation of the <Condition> element  
3474 SHALL follow the same evaluation semantics as those of the <Apply> element.

## 3475 A.11. Element <Function>

3476 The <Function> element names a standard XACML function or an extension function in its  
3477 FunctionId attribute. The <Function> element MAY be used as an argument in functions that  
3478 take a function as an argument.

## 3479 A.12. Matching elements

3480 Matching elements appear in the <Target> element of *rules*, *policies* and *policy sets*. They are  
3481 the following:

- 3482 • <SubjectMatch>
- 3483 • <ResourceMatch>
- 3484 • <ActionMatch>
- 3485 • <EnvironmentMatch>

3486 These elements represent boolean expressions over *attributes* of the *subject*, *resource*, *action*  
3487 and *environment*, respectively.

3488 The match elements: <SubjectMatch>, <ResourceMatch>, <ActionMatch> and  
3489 <EnvironmentMatch> SHALL use functions that match two arguments, returning a result type of  
3490 "xs:boolean", to perform the match evaluation. The function used for determining a match is named  
3491 in the MatchId attribute of these elements. Each argument to the named function MUST match  
3492 the appropriate primitive types for the <AttributeDesignator> or <AttributeSelector>  
3493 element and the following explicit *attribute* value, such that the explicit *attribute* value is placed as  
3494 the first argument to the function, while an element of the *bag* returned by the  
3495 <AttributeDesignator> or <AttributeSelector> element is placed as the second  
3496 argument to the function.

3497 The XACML standard functions that may be used as a MatchId attribute value are:

- 3498 function: *type-equal*
- 3499 function: *type-greater-than*
- 3500 function: *type-greater-than-or-equal*
- 3501 function: *type-less-than*
- 3502 function: *type-less-than-or-equal*
- 3503 function: *t e-match*

3504 The evaluation semantics for a match is as follows. If an operational error were to occur while  
3505 evaluating the <AttributeDesignator> or <AttributeSelector> element, then the result of  
3506 the entire expression SHALL be "Indeterminate". If the <AttributeDesignator> or  
3507 <AttributeSelector> element were to evaluate to an empty *bag*, then the result of the  
3508 expression SHALL be "False". Otherwise, the match function SHALL be applied between the  
3509 explicit *attribute* value and each element of the *bag* returned from the <AttributeDesignator>  
3510 or <AttributeSelector> element. If at least one of those function applications were to evaluate  
3511 to "True", then the result of the entire expression SHALL be "True". Otherwise, if at least one of the  
3512 function applications results in "Indeterminate", then the result SHALL be "Indeterminate". Finally,

3513 only if all function applications evaluate to "False", SHALL the result of the entire expression be  
3514 "False".

3515 A match can equivalently be expressed in a **target** or a **condition**. For instance, the match  
3516 expression that compares a "subject-name" starting with the name "John" can be expressed as  
3517 follows:

```
3518 <SubjectMatch MatchId="function:regexp-string-match">  
3519   <SubjectAttributeDesignator AttributeId="subject-name"/>  
3520   <AttributeValue  
3521     DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>  
3522 </SubjectMatch>
```

3523 Alternatively, it can be expressed as an <Apply> element in the **condition** by using the  
3524 "function:any-of" function, as follows:

```
3525 <Apply FunctionId="function:any-of">  
3526   <Function FunctionId="function:regexp-string-match"/>  
3527   <AttributeValue  
3528     DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>  
3529   <SubjectAttributeDesignator AttributeId="subject-name"/>  
3530 </Apply>
```

3531 For the match elements: <SubjectMatch>, <ResourceMatch>, <ActionMatch> and  
3532 <EnvironmentMatch> that appear in the <Target> element of a <Rule>, <Policy> or  
3533 <PolicySet> the value specified by the MatchId attribute SHALL be restricted to the following  
3534 functions:

- 3535 • "function:type-equal" (for each primitive *type*),
- 3536 • "function:regexp-string-match",
- 3537 • "function:rfc822Name-match" and
- 3538 • "function:x500Name-match",

3539 and only those functions. Functions that are strictly within an extension to XACML should not  
3540 appear as a value to the MatchId attribute in this case. Restricting the MatchId attribute to these  
3541 functions facilitates the use of indexing to find the **applicable policy** for a particular **authorization**  
3542 **request**.

## 3543 A.13. Arithmetic evaluation

3544 IEEE 754 [IEEE 754] specifies how to evaluate arithmetic functions in a context, which specifies  
3545 defaults for precision, rounding, etc. XACML SHALL use this specification for the evaluation of all  
3546 integer and double functions relying on the *Extended Default Context*, enhanced with double  
3547 precision:

3548 flags - all set to 0

3549 trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the "division-by-zero" trap  
3550 enabler, which SHALL be set to 1

3551 precision - is set to the designated double precision

3552 rounding - is set to round-half-even (IEEE 854 §4.1)

## 3553 **A.14. XACML standard functions**

3554 XACML specifies the following functions that are prefixed with the “function:” relative name space  
3555 identifier.

### 3556 **A14.1 Equality predicates**

3557 The following functions are the *equality* functions for the various primitive types. Each function for a  
3558 particular type follows a specified standard convention for that type. If an argument of one of these  
3559 functions were to evaluate to "Indeterminate", then the function SHALL be set to "Indeterminate".

3560 • string-equal

3561 This function SHALL take two arguments of “http://www.w3.org/2001/XMLSchema#string”  
3562 and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. The function  
3563 SHALL return "True" if and only if the value of both of its arguments are of equal length and  
3564 each string is determined to be equal byte-by-byte according to the function “integer-equal”.

3565 • boolean-equal

3566 This function SHALL take two arguments of  
3567 “http://www.w3.org/2001/XMLSchema#boolean” and SHALL return "True" if and only if both  
3568 values are equal.

3569 • integer-equal

3570 This function SHALL take two arguments of type  
3571 “http://www.w3.org/2001/XMLSchema#integer” and SHALL return an  
3572 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL perform its evaluation on  
3573 integers according to IEEE 754 [IEEE 754].

3574 • double-equal

3575 This function SHALL take two arguments of type  
3576 “http://www.w3.org/2001/XMLSchema#double” and SHALL return an  
3577 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL perform its evaluation on  
3578 doubles according to IEEE 754 [IEEE 754].

3579 • date-equal

3580 This function SHALL take two arguments of type  
3581 “http://www.w3.org/2001/XMLSchema#date” and SHALL return an  
3582 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL perform its evaluation  
3583 according to the “op:date-equal” function [XQO Section 8.3.11].

3584 • time-equal

3585 This function SHALL take two arguments of type  
3586 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an  
3587 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL perform its evaluation according  
3588 to the “op:time-equal” function [XQO Section 8.3.14].

3589 • dateTime-equal

3590 This function SHALL take two arguments of type  
3591 “http://www.w3.org/2001/XMLSchema#dateTime” and SHALL return an  
3592 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL perform its evaluation  
3593 according to the “op:dateTime-equal” function [XQO Section 8.3.8].

- 3594 • anyURI-equal
- 3595 This function SHALL take two arguments of type  
 3596 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return an  
 3597 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation  
 3598 according to the "op:anyURI-equal" function [XQO Section 10.2.1].
- 3599 • x500Name-equal
- 3600 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-  
 3601 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It  
 3602 shall return "True" if and only if each Relative Distinguished Name (RDN) in the two  
 3603 arguments matches. Two RDNs shall be said to match if and only if the result of the  
 3604 following operations is "True"<sup>3</sup>.
- 3605 1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory  
 3606 Access Protocol (v3): UTF-8 String Representation of Distinguished Names".
  - 3607 2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute  
 3608 ValuePairs in that RDN in ascending order when compared as octet strings  
 3609 (described in ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").
  - 3610 3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key  
 3611 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section  
 3612 4.1.2.4 "Issuer".
- 3613 • rfc822Name-equal
- 3614 This function SHALL take two arguments of type "urn:oasis:names:tc:xacml:1.0:data-  
 3615 type:rfc822Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean".  
 3616 This function SHALL determine whether two "urn:oasis:names:tc:xacml:1.0:data-  
 3617 type:rfc822Name" arguments are equal. An RFC822 name consists of a *local-part* followed  
 3618 by "@" followed by a *domain-part*. The *local-part* is case-sensitive, while the *domain-part*  
 3619 (which is usually a DNS host name) is not case-sensitive. Perform the following  
 3620 operations:
- 3621 1. Normalize the *domain-part* of each argument to lower case
  - 3622 2. Compare the expressions by applying the function "function:string-equal" to the  
 3623 normalized arguments.
- 3624 • hexBinary-equal
- 3625 This function SHALL take two arguments of type  
 3626 "http://www.w3.org/2001/XMLSchema#hexBinary" and SHALL return an  
 3627 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL return "True" if the  
 3628 octet sequences represented by the value of both arguments have equal length and are  
 3629 equal in a conjunctive, point-wise, comparison using the "function:integer-equal". The  
 3630 conversion from the string representation to an octet sequence SHALL be as specified in  
 3631 [XS Section 8.2.15]
- 3632 • base64Binary-equal

---

<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.

3633 This function SHALL take two arguments of type  
3634 "http://www.w3.org/2001/XMLSchema#base64Binary" and SHALL return an  
3635 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL return "True" if the  
3636 octet sequences represented by the value of both arguments have equal length and are  
3637 equal in a conjunctive, point-wise, comparison using the "function:integer-equal". The  
3638 conversion from the string representation to an octet sequence SHALL be as specified in  
3639 [XS Section 8.2.16]

## 3640 **A14.2 Arithmetic functions**

3641 All of the following functions SHALL take two arguments of the specified *type*, integer or double,  
3642 and SHALL return an element of integer or double type, respectively. However, the "add" functions  
3643 MAY take more than two arguments. Each function evaluation SHALL proceed as specified by  
3644 their logical counterparts in IEEE 754 [IEEE 754]. In an expression that contains any of these  
3645 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to  
3646 "Indeterminate". In the case of the divide functions, if the divisor is zero, then the function SHALL  
3647 evaluate to "Indeterminate".

3648 • integer-add

3649 This function MAY have two or more arguments.

3650 • double-add

3651 This function MAY have two or more arguments.

3652 • integer-subtract

3653 • double-subtract

3654 • integer-multiply

3655 • double-multiply

3656 • integer-divide

3657 • double-divide

3658 • integer-mod

3659 The following functions SHALL take a single argument of the specified *type*. The round and floor  
3660 functions SHALL take a single argument of type "http://www.w3.org/2001/XMLSchema#double" and  
3661 return type "http://www.w3.org/2001/XMLSchema#double". In an expression that contains any of  
3662 these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to  
3663 "Indeterminate".

3664 • integer-abs

3665 • double-abs

3666 • round

3667 • floor

## 3668 **A14.3 String conversion functions**

3669 The following functions convert between values of the XACML  
3670 "http://www.w3.org/2001/XMLSchema#string" primitive types. In an expression that contains any of

3671 these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to  
3672 "Indeterminate".

- 3673 • string-normalize-space

3674 This function SHALL take one argument of type  
3675 "http://www.w3.org/2001/XMLSchema#string" and SHALL normalize the value by stripping  
3676 off all leading and trailing whitespace characters.

- 3677 • string-normalize-to-lower-case

3678 This function SHALL take one argument of "http://www.w3.org/2001/XMLSchema#string"  
3679 and SHALL normalize the value by converting each upper case character to its lower case  
3680 equivalent.

## 3681 **A14.4 Numeric type conversion functions**

3682 The following functions convert between the XACML  
3683 "http://www.w3.org/2001/XMLSchema#integer" and "http://www.w3.org/2001/XMLSchema#double"  
3684 primitive types. In any expression in which the functions defined below are applied, if any argument  
3685 while being evaluated results in "Indeterminate", the expression SHALL return "Indeterminate".

- 3686 • double-to-integer

3687 This function SHALL take one argument of type  
3688 "http://www.w3.org/2001/XMLSchema#double" and SHALL truncate its numeric value to a  
3689 whole number and return an element of type  
3690 "http://www.w3.org/2001/XMLSchema#integer".

- 3691 • integer-to-double

3692 This function SHALL take one argument of type  
3693 "http://www.w3.org/2001/XMLSchema#integer" and SHALL promote its value to an element  
3694 of type "http://www.w3.org/2001/XMLSchema#double" of the same numeric value.

## 3695 **A14.5 Logical functions**

3696 This section contains the specification for logical functions that operate on arguments of the  
3697 "http://www.w3.org/2001/XMLSchema#boolean" type.

- 3698 • or

3699 This function SHALL return "False" if it has no arguments and SHALL return "True" if one of  
3700 its arguments evaluates to "True". The order of evaluation SHALL be from first argument to  
3701 last. The evaluation SHALL stop with a result of "True" if any argument evaluates to "True",  
3702 leaving the rest of the arguments unevaluated. In an expression that contains any of these  
3703 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to  
3704 "Indeterminate".

- 3705 • and

3706 This function SHALL return "True" if it has no arguments and SHALL return "False" if one of  
3707 its arguments evaluates to "False". The order of evaluation SHALL be from first argument  
3708 to last. The evaluation SHALL stop with a result of "False" if any argument evaluates to  
3709 "False", leaving the rest of the arguments unevaluated. In an expression that contains any  
3710 of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate  
3711 to "Indeterminate".

- 3712 • n-of
- 3713 The first argument to this function SHALL be of type  
 3714 "http://www.w3.org/2001/XMLSchema#integer", specifying the number of the remaining  
 3715 arguments that MUST evaluate to "True" for the expression to be considered "True". If the  
 3716 first argument is 0, the result SHALL be "True". If the number of arguments after the first  
 3717 one is less than the value of the first argument, then the expression SHALL result in  
 3718 "Indeterminate". The order of evaluation SHALL be: first evaluate the integer value, then  
 3719 evaluate each subsequent argument. The evaluation SHALL stop and return "True" if the  
 3720 specified number of arguments evaluate to "True". The evaluation of arguments SHALL  
 3721 stop if it is determined that evaluating the remaining arguments will not satisfy the  
 3722 requirement. In an expression that contains any of these functions, if any argument is  
 3723 "Indeterminate", then the expression SHALL evaluate to "Indeterminate".
- 3724 • not
- 3725 This function SHALL take one logical argument. If the argument evaluates to "True", then  
 3726 the result of the expression SHALL be "False". If the argument evaluates to "False", then  
 3727 the result of the expression SHALL be "True". In an expression that contains any of these  
 3728 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to  
 3729 "Indeterminate".
- 3730 • present
- 3731 This function SHALL take an attribute value of type  
 3732 "http://www.w3.org/2001/XMLSchema#anyURI" as used as the `AttributeId` in an  
 3733 `<AttributeDesignator>` element. This expression SHALL return "True" if the named  
 3734 **attribute** can be located in the request **context**, which means that an  
 3735 `<AttributeDesignator>` or `<AttributeSelector>` element for this named **attribute**  
 3736 will return a **bag** containing at least one value. If it cannot be determined that the **attribute**  
 3737 is present in the request **context**, or the attribute cannot be evaluated, then the expression  
 3738 SHALL result in "Indeterminate".

## 3739 **A14.6 Arithmetic comparison functions**

3740 These functions form a minimal set for comparing two numbers, yielding a boolean result. They  
 3741 SHALL comply with the rules governed by IEEE 754 [IEEE 754]. In an expression that contains  
 3742 any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to  
 3743 "Indeterminate".

- 3744 • integer-greater-than
- 3745 • integer-greater-than-or-equal
- 3746 • integer-less-than
- 3747 • integer-less-than-or-equal
- 3748 • double-greater-than
- 3749 • double-greater-than-or-equal
- 3750 • double-less-than
- 3751 • double-less-than-or-equal

3752

## A14.7 Date and time arithmetic functions

3753 These functions perform arithmetic operations with the date and time. In an expression that  
3754 contains any of these functions, if any argument is "Indeterminate", then the expression SHALL  
3755 evaluate to "Indeterminate".

- 3756 • `dateTime-add-dayTimeDuration`

3757 This function SHALL take two arguments, the first is of type  
3758 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is of type  
3759 "xf:dayTimeDuration". It SHALL return a result of  
3760 "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL return the value by  
3761 adding the second argument to the first argument according to the specification of adding  
3762 durations to date and time [XS Appendix E].

- 3763 • `dateTime-add-yearMonthDuration`

3764 This function SHALL take two arguments, the first is a  
3765 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a  
3766 "xf:yearMonthDuration". It SHALL return a result of  
3767 "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL return the value by  
3768 adding the second argument to the first argument according to the specification of adding  
3769 durations to date and time [XS Appendix E].

- 3770 • `dateTime-subtract-dayTimeDuration`

3771 This function SHALL take two arguments, the first is a  
3772 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a  
3773 "xf:dayTimeDuration". It SHALL return a result of  
3774 "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument is a positive  
3775 duration, then this function SHALL return the value by adding the corresponding negative  
3776 duration, as per the specification [XS Appendix E]. If the second argument is a negative  
3777 duration, then the result SHALL be as if the function "function:dateTime-add-  
3778 dayTimeDuration" had been applied to the corresponding positive duration.

- 3779 • `dateTime-subtract-yearMonthDuration`

3780 This function SHALL take two arguments, the first is a  
3781 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a  
3782 "xf:yearMonthDuration". It SHALL return a result of  
3783 "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument is a positive  
3784 duration, then this function SHALL return the value by adding the corresponding negative  
3785 duration, as per the specification [XS Appendix E]. If the second argument is a negative  
3786 duration, then the result SHALL be as if the function "function:dateTime-add-  
3787 yearMonthDuration" had been applied to the corresponding positive duration.

- 3788 • `date-add-yearMonthDuration`

3789 This function SHALL take two arguments, the first is a  
3790 "http://www.w3.org/2001/XMLSchema#date" and the second is a "xf:yearMonthDuration". It  
3791 return a result of "http://www.w3.org/2001/XMLSchema#date". This function SHALL return  
3792 the value by adding the second argument to the first argument according to the  
3793 specification of adding durations to date [XS Appendix E].

- 3794 • `date-subtract-yearMonthDuration`

3795 This function SHALL take two arguments, the first is a  
3796 "http://www.w3.org/2001/XMLSchema#date" and the second is a "xf:yearMonthDuration". It  
3797 SHALL return a result of "http://www.w3.org/2001/XMLSchema#date". If the second

3798 argument is a positive duration, then this function SHALL return the value by adding the  
3799 corresponding negative duration, as per the specification [XS Appendix E]. If the second  
3800 argument is a negative duration, then the result SHALL be as if the function “function:date-  
3801 add-yearMonthDuration” had been applied to the corresponding positive duration.

## 3802 **A14.8 Non-numeric comparison functions**

3803 These functions perform comparison operations on two arguments of non-numerical types. In an  
3804 expression that contains any of these functions, if any argument is "Indeterminate", then the  
3805 expression SHALL evaluate to "Indeterminate".

### 3806 • string-greater-than

3807 This function SHALL take two arguments of type  
3808 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an  
3809 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if and only if the  
3810 arguments are compared byte by byte and, after an initial prefix of corresponding bytes  
3811 from both arguments that are considered equal by “function:integer-equal”, the next byte by  
3812 byte comparison is such that the byte from the first argument is greater than the byte from  
3813 the second argument by the use of the function “function:integer-equal”.

### 3814 • string-greater-than-or-equal

3815 This function SHALL take two arguments of type  
3816 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an  
3817 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated  
3818 with the logical function “function:or” with two arguments containing the functions  
3819 “function:string-greater-than” and “function:string-equal” containing the original arguments

### 3820 • string-less-than

3821 This function SHALL take two arguments of type  
3822 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an  
3823 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if and only if the  
3824 arguments are compared byte by byte and, after an initial prefix of corresponding bytes  
3825 from both arguments are considered equal by “function:integer-equal”, the next byte by  
3826 byte comparison is such that the byte from the first argument is less than the byte from the  
3827 second argument by the use of the function “function:integer-less-than”.

### 3828 • string-less-than-or-equal

3829 This function SHALL take two arguments of type  
3830 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an  
3831 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated  
3832 with the function “function:or” with two arguments containing the functions “function:string-  
3833 less-than” and “function:string-equal” containing the original arguments.

### 3834 • time-greater-than

3835 This function SHALL take two arguments of type  
3836 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an  
3837 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first  
3838 argument is greater than the second argument according to the order relation specified for  
3839 “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].

### 3840 • time-greater-than-or-equal

3841 This function SHALL take two arguments of type  
3842 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an

3843 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
3844 argument is greater than or equal to the second argument according to the order relation  
3845 specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

3846 • time-less-than

3847 This function SHALL take two arguments of type  
3848 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
3849 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
3850 argument is less than the second argument according to the order relation specified for  
3851 "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

3852 • time-less-than-or-equal

3853 This function SHALL take two arguments of type  
3854 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
3855 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
3856 argument is less than or equal to the second argument according to the order relation  
3857 specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

3858 • dateTime-greater-than

3859 This function SHALL take two arguments of type  
3860 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
3861 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
3862 argument is greater than the second argument according to the order relation specified for  
3863 "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

3864 • dateTime-greater-than-or-equal

3865 This function SHALL take two arguments of type  
3866 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
3867 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
3868 argument is greater than or equal to the second argument according to the order relation  
3869 specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

3870 • dateTime-less-than

3871 This function SHALL take two arguments of type  
3872 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
3873 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
3874 argument is less than the second argument according to the order relation specified for  
3875 "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

3876 • dateTime-less-than-or-equal

3877 This function SHALL take two arguments of type "http://www.w3.org/2001/XMLSchema#  
3878 dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It  
3879 SHALL return "True" if the first argument is less than or equal to the second argument  
3880 according to the order relation specified for  
3881 "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

3882 • date-greater-than

3883 This function SHALL take two arguments of type  
3884 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
3885 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
3886 argument is greater than the second argument according to the order relation specified for  
3887 "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

3888 • date-greater-than-or-equal  
3889 This function SHALL take two arguments of type  
3890 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
3891 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
3892 argument is greater than or equal to the second argument according to the order relation  
3893 specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

3894 • date-less-than  
3895 This function SHALL take two arguments of type  
3896 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
3897 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
3898 argument is less than the second argument according to the order relation specified for  
3899 "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

3900 • date-less-than-or-equal  
3901 This function SHALL take two arguments of type  
3902 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
3903 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
3904 argument is less than or equal to the second argument according to the order relation  
3905 specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

## 3906 **A14.9 Bag functions**

3907 These functions operate on a **bag** of *type* values, where *type* is one of the primitive types. In an  
3908 expression that contains any of these functions, if any argument is "Indeterminate", then the  
3909 expression SHALL evaluate to "Indeterminate". Some additional conditions defined for each  
3910 function below SHALL cause the expression to evaluate to "Indeterminate".

3911 • *type*-one-and-only  
3912 This function SHALL take an argument of a **bag** of *type* values and SHALL return a value  
3913 of *type*. It SHALL return the only value in the **bag**. If the **bag** does not have one and only  
3914 one value, then the expression SHALL evaluate to "Indeterminate".

3915 • *type*-bag-size  
3916 This function SHALL take a **bag** of *type* values as an argument and SHALL return an  
3917 "http://www.w3.org/2001/XMLSchema#integer" indicating the number of values in the **bag**.

3918 • *type*-is-in  
3919 This function SHALL take an argument of type *type* as the first argument and a **bag** of *type*  
3920 values as the second argument. The expression SHALL evaluate to "True" if the first  
3921 argument matches by the "function:type-equal" to any value in the **bag**.

3922 • *type*-bag  
3923 This function SHALL take any number of arguments of a single type and return a **bag** of  
3924 *type* values containing the values of the arguments. An application of this function to zero  
3925 arguments SHALL produce an empty **bag** of the specified type.

3926

## A14.10 Set functions

3927 These functions operate on **bags** mimicking sets by eliminating duplicate elements from a **bag**. In  
3928 an expression that contains any of these functions, if any argument is "Indeterminate", then the  
3929 expression SHALL evaluate to "Indeterminate".

3930 • *type-intersection*

3931 This function SHALL take two arguments that are both a **bag** of *type* values. The  
3932 expression SHALL return a **bag** of *type* values such that it contains only elements that are  
3933 common between the two **bags**, which is determined by "function:type-equal". No  
3934 duplicates as determined by "function:type-equal" SHALL exist in the result.

3935 • *type-at-least-one-member-of*

3936 This function SHALL take two arguments that are both a **bag** of *type* values. The  
3937 expression SHALL evaluate to "True" if at least one element of the first argument is  
3938 contained in the second argument as determined by "function:type-is-in".

3939 • *type-union*

3940 This function SHALL take two arguments that are both a **bag** of *type* values. The  
3941 expression SHALL return a **bag** of *type* such that it contains all elements of both **bags**. No  
3942 duplicates as determined by "function:type-equal" SHALL exist in the result.

3943 • *type-subset*

3944 This function SHALL take two arguments that are both a **bag** of *type* values. It SHALL  
3945 return "True" if the first argument is a subset of the second argument. Each argument is  
3946 considered to have its duplicates removed as determined by "function:type-equal" before  
3947 subset calculation.

3948 • *type-set-equals*

3949 This function SHALL take two arguments that are both a **bag** of *type* values and SHALL  
3950 return the result of applying "function:and" to the application of "function:type-subset" to the  
3951 first and second arguments and the application of "function:type-subset" to the second and  
3952 first arguments.

## 3953 A14.11 Higher-order bag functions

3954 This section describes functions in XACML that perform operations on **bags** such that functions  
3955 may be applied to the **bags** in general.

3956 In this section, a general-purpose functional language called Haskell [**Haskell**] is used to formally  
3957 specify the semantics of these functions. Although the English description is adequate, a formal  
3958 specification of the semantics is helpful.

3959 For a quick summary, in the following Haskell notation, a function definition takes the form of  
3960 clauses that are applied to patterns of structures, namely lists. The symbol "[]" denotes the empty  
3961 list, whereas the expression "(x:xs)" matches against an argument of a non-empty list of which "x"  
3962 represents the first element of the list, and "xs" is the rest of the list, which may be an empty list. We  
3963 use the Haskell notion of a list, which is an ordered collection of elements, to model the XACML  
3964 **bags** of values.

3965 A simple Haskell definition of a familiar function "function:and" that takes a list of booleans is  
3966 defined as follows:

3967 `and:: [Bool] -> Bool`

3968           and []       = "True"

3969           and (x:xs) = x && (and xs)

3970       The first definition line denoted by a ":" formally describes the type of the function, which takes a  
3971       list of booleans, denoted by "[Bool]", and returns a boolean, denoted by "Bool". The second  
3972       definition line is a clause that states that the function "and" applied to the empty list is "True". The  
3973       second definition line is a clause that states that for a non-empty list, such that the first element is  
3974       "x", which is a value of type Bool, the function "and" applied to x SHALL be combined with, using  
3975       the logical conjunction function, which is denoted by the infix symbol "&&", the result of recursively  
3976       applying the function "and" to the rest of the list. Of course, an application of the "and" function is  
3977       "True" if and only if the list to which it is applied is empty or every element of the list is "True". For  
3978       example, the evaluation of the following Haskell expressions,

3979            (and []), (and ["True"]), (and ["True", "True"]), (and ["True", "True", "False"])

3980       evaluate to "True", "True", "True", and "False", respectively.

3981       In an expression that contains any of these functions, if any argument is "Indeterminate", then the  
3982       expression SHALL evaluate to "Indeterminate".

3983       • any-of

3984           This function applies a boolean function between a specific primitive value and a **bag** of  
3985           values, and SHALL return "True" if and only if the predicate is "True" for at least one  
3986           element of the **bag**.

3987           This function SHALL take three arguments. The first argument SHALL be a <Function>  
3988           element that names a boolean function that takes two arguments of primitive types. The  
3989           second argument SHALL be a value of a primitive type. The third argument SHALL be a  
3990           **bag** of a primitive type. The expression SHALL be evaluated as if the function named in  
3991           the <Function> element is applied to the second argument and each element of the third  
3992           argument (the **bag**) and the results are combined with "function:or".

3993           In Haskell, the semantics of this operation are as follows:

```
3994           any_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
3995           any_of f a []       = "False"
3996           any_of f a (x:xs) = (f a x) || (any_of f a xs)
```

3997           In the above notation, "f" is the function name to be applied, "a" is the primitive value, and  
3998           "(x:xs)" represents the first element of the list as "x" and the rest of the list as "xs".

3999           For example, the following expression SHALL return "True":

```
4000       <Apply FunctionId="function:any-of">
4001        <Function FunctionId="function:string-equal" />
4002        <AttributeValue
4003        DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4004        <Apply FunctionId="function:string-bag">
4005        <AttributeValue
4006        DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4007        <AttributeValue
4008        DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4009        <AttributeValue
4010        DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4011        <AttributeValue
4012        DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4013        </Apply>
4014       </Apply>
```

4015 This expression is "True" because the first argument is equal to at least one of the  
4016 elements of the **bag**.

4017 • all-of

4018 This function applies a boolean function between a specific primitive value and a **bag** of  
4019 values, and returns "True" if and only if the predicate is "True" for every element of the **bag**.

4020 This function SHALL take three arguments. The first argument SHALL be a <Function>  
4021 element that names a boolean function that takes two arguments of primitive types. The  
4022 second argument SHALL be a value of a primitive type. The third argument SHALL be a  
4023 **bag** of a primitive type. The expression SHALL be evaluated as if the function named in  
4024 the <Function> element were applied to the second argument and each element of the  
4025 third argument (the **bag**) and the results were combined using "function:and".

4026 In Haskell, the semantics of this operation are as follows:

```
4027 all_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
4028 all_of f a [] = "False"
4029 all_of f a (x:xs) = (f a x) && (all_of f a xs)
```

4030 In the above notation, "f" is the function name to be applied, "a" is the primitive value, and  
4031 "(x:xs)" represents the first element of the list as "x" and the rest of the list as "xs".

4032 For example, the following expression SHALL evaluate to "True":

```
4033 <Apply FunctionId="function:all-of">
4034   <Function FunctionId="function:integer-greater" />
4035   <AttributeValue
4036     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4037   <Apply FunctionId="function:integer-bag">
4038     <AttributeValue
4039       DataType="http://www.w3.org/2001/XMLSchema#integer">9</AttributeValue>
4040     <AttributeValue
4041       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4042     <AttributeValue
4043       DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4044     <AttributeValue
4045       DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4046   </Apply>
4047 </Apply>
```

4048 This expression is "True" because the first argument is greater than *all* of the elements of  
4049 the **bag**.

4050 • any-of-any

4051 This function applies a boolean function between each element of a **bag** of values and  
4052 each element of another **bag** of values, and returns "True" if and only if the predicate is  
4053 "True" for at least one comparison.

4054 This function SHALL take three arguments. The first argument SHALL be a <Function>  
4055 element that names a boolean function that takes two arguments of primitive types. The  
4056 second argument SHALL be a **bag** of a primitive type. The third argument SHALL be a  
4057 **bag** of a primitive type. The expression SHALL be evaluated as if the function named in  
4058 the <Function> element were applied between *every* element in the second argument  
4059 and *every* element of the third argument (the **bag**) and the results were combined using  
4060 "function:or". The semantics are that the result of the expression SHALL be "True" if and  
4061 only if the applied predicate is "True" for *any* comparison of elements from the two **bags**.

4062 In Haskell, taking advantage of the “any\_of” function defined above, the semantics of the  
4063 “any\_of\_any” function are as follows:

```
4064 any_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4065 any_of_any f [] ys = "False"
4066 any_of_any f (x:xs) ys = (any_of f x ys) || (any_of_any f xs ys)
```

4067 In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first  
4068 element of the list as “x” and the rest of the list as “xs”.

4069 For example, the following expression SHALL evaluate to "True":

```
4070 <Apply FunctionId="function:any-of-any">
4071   <Function FunctionId="function:string-equal"/>
4072   <Apply FunctionId="function:string-bag">
4073     <AttributeValue
4074       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4075     <AttributeValue
4076       DataType="http://www.w3.org/2001/XMLSchema#string">Mary</AttributeValue>
4077   </Apply>
4078   <Apply FunctionId="function:string-bag">
4079     <AttributeValue
4080       DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4081     <AttributeValue
4082       DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4083     <AttributeValue
4084       DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4085     <AttributeValue
4086       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4087   </Apply>
4088 </Apply>
```

4089 This expression is "True" because at least one of the elements of the first **bag**, namely  
4090 “Ringo”, is equal to at least one of the string values of the second **bag**.

4091 • all-of-any

4092 This function applies a boolean function between the elements of two **bags**. The  
4093 expression is "True" if and only if the predicate is "True" between each and all of the  
4094 elements of the first **bag** collectively against at least one element of the second **bag**.

4095 This function SHALL take three arguments. The first argument SHALL be a <Function>  
4096 element that names a boolean function that takes two arguments of primitive types. The  
4097 second argument SHALL be a **bag** of a primitive type. The third argument SHALL be a  
4098 **bag** of a primitive type. The expression SHALL be evaluated as if function named in the  
4099 <Function> element were applied between every element in the second argument and  
4100 every element of the third argument (the **bag**) using “function:and”. The semantics are that  
4101 the result of the expression SHALL be "True" if and only if the applied predicate is "True"  
4102 for each element of the first **bag** and any element of the second **bag**.

4103 In Haskell, taking advantage of the “any\_of” function defined in Haskell above, the  
4104 semantics of the “all\_of\_any” function are as follows:

```
4105 all_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4106 all_of_any f [] ys = "False"
4107 all_of_any f (x:xs) ys = (any_of f x ys) && (all_of_any f xs ys)
```

4108 In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first  
4109 element of the list as “x” and the rest of the list as “xs”.

4110 For example, the following expression SHALL evaluate to "True":

```

4111 <Apply FunctionId="function:all-of-any">
4112   <Function FunctionId="function:integer-greater"/>
4113   <Apply FunctionId="function:integer-bag">
4114     <AttributeValue
4115       DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4116     <AttributeValue
4117       DataType="http://www.w3.org/2001/XMLSchema#integer">20</AttributeValue>
4118   </Apply>
4119   <Apply FunctionId="function:integer-bag">
4120     <AttributeValue
4121       DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4122     <AttributeValue
4123       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4124     <AttributeValue
4125       DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4126     <AttributeValue
4127       DataType="http://www.w3.org/2001/XMLSchema#integer">21</AttributeValue>
4128   </Apply>
4129 </Apply>

```

4130 This expression is "True" because all of the elements of the first **bag**, each "10" and "20",  
4131 are greater than at least one of the integer values "1", "3", "5", "21" of the second **bag**.

4132 • any-of-all

4133 This function applies a boolean function between the elements of two **bags**. The  
4134 expression SHALL be "True" if and only if the predicate is "True" between at least one of  
4135 the elements of the first **bag** collectively against all the elements of the second **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 **bag** of a primitive type. The third argument SHALL be a  
4139 **bag** of a primitive type. The expression SHALL be evaluated as if the function named in  
4140 the <Function> element were applied between *every* element in the second argument  
4141 and *every* element of the third argument (the **bag**) and the results were combined using  
4142 "function:or". The semantics are that the result of the expression SHALL be "True" if and  
4143 only if the applied predicate is "True" for *any* element of the first **bag** compared to *all* the  
4144 elements of the second **bag**.

4145 In Haskell, taking advantage of the "all\_of" function defined in Haskell above, the semantics  
4146 of the "any\_of\_all" function are as follows:

```

4147 any_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4148 any_of_all f [] ys = "False"
4149 any_of_all f (x:xs) ys = (all_of f x ys) || ( any_of_all f xs ys)

```

4150 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first  
4151 element of the list as "x" and the rest of the list as "xs".

4152 For example, the following expression SHALL evaluate to "True":

```

4153 <Apply FunctionId="function:any-of-all">
4154   <Function FunctionId="function:integer-greater"/>
4155   <Apply FunctionId="function:integer-bag">
4156     <AttributeValue
4157       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4158     <AttributeValue
4159       DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4160   </Apply>
4161   <Apply FunctionId="function:integer-bag">
4162     <AttributeValue
4163       DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4164     <AttributeValue
4165       DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4166     <AttributeValue
4167       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4168     <AttributeValue
4169       DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4170   </Apply>
4171 </Apply>

```

4172 This expression is "True" because at least one element of the first **bag**, namely "5", is  
4173 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4174 • all-of-all

4175 This function applies a boolean function between the elements of two **bags**. The  
4176 expression SHALL be "True" if and only if the predicate is "True" between each and all of  
4177 the elements of the first **bag** collectively against all the elements of the second **bag**.

4178 This function SHALL take three arguments. The first argument SHALL be a <Function>  
4179 element that names a boolean function that takes two arguments of primitive types. The  
4180 second argument SHALL be a **bag** of a primitive type. The third argument SHALL be a  
4181 **bag** of a primitive type. The expression is evaluated as if the function named in the  
4182 <Function> element were applied between *every* element in the second argument and  
4183 *every* element of the third argument (the **bag**) and the results were combined using  
4184 "function:and". The semantics are that the result of the expression is "True" if and only if  
4185 the applied predicate is "True" for *all* elements of the first **bag** compared to *all* the elements  
4186 of the second **bag**.

4187 In Haskell, taking advantage of the "all\_of" function defined in Haskell above, the semantics  
4188 of the "all\_of\_all" function is as follows:

```

4189 all_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4190 all_of_all f [] ys = "False"
4191 all_of_all f (x:xs) ys = (all_of f x ys) && (all_of_all f xs ys)

```

4192 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first  
4193 element of the list as "x" and the rest of the list as "xs".

4194 For example, the following expression SHALL evaluate to "True":

```

4195 <Apply FunctionId="function:all-of-all">
4196   <Function FunctionId="function:integer-greater"/>
4197   <Apply FunctionId="function:integer-bag">
4198     <AttributeValue
4199       DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>
4200     <AttributeValue
4201       DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4202     </Apply>
4203   <Apply FunctionId="function:integer-bag">
4204     <AttributeValue
4205       DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4206     <AttributeValue
4207       DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4208     <AttributeValue
4209       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4210     <AttributeValue
4211       DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4212     </Apply>
4213   </Apply>

```

4214 This expression is "True" because all elements of the first **bag**, "5" and "6", are each  
4215 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4216 • map

4217 This function converts a **bag** of values to another **bag** of values.

4218 This function SHALL take two arguments. The first function SHALL be a <Function>  
4219 element naming a function that takes a single argument of a primitive type and returns a  
4220 value of a primitive type. The second argument SHALL be a **bag** of a primitive type. The  
4221 expression SHALL be evaluated as if the function named in the <Function> element were  
4222 applied to each element in the **bag** resulting in a **bag** of the converted value. The result  
4223 SHALL be a **bag** of the primitive type that is the same type that is returned by the function  
4224 named in the <Function> element.

4225 In Haskell, this function is defined as follows:

```

4226     map:: (a -> b) -> [a] -> [b]
4227     map f []     = []
4228     map f (x:xs) = (f x) : (map f xs)

```

4229 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first  
4230 element of the list as "x" and the rest of the list as "xs".

4231 For example, the following expression,

```

4232 <Apply FunctionId="function:map">
4233   <Function FunctionId="function:string-normalize-to-lower-case">
4234   <Apply FunctionId="function:string-bag">
4235     <AttributeValue
4236       DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>
4237     <AttributeValue
4238       DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>
4239     </Apply>
4240   </Apply>

```

4241 evaluates to a **bag** containing "hello" and "world!".

4242

## A14.12 Special match functions

4243 These functions operate on various types and evaluate to  
4244 "http://www.w3.org/2001/XMLSchema#boolean" based on the specified standard matching  
4245 algorithm. In an expression that contains any of these functions, if any argument is "Indeterminate",  
4246 then the expression SHALL evaluate to "Indeterminate".

4247 • regex-string-match

4248 This function decides a regular expression match. It SHALL take two arguments of  
4249 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4250 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
4251 expression and the second argument SHALL be a general string. The function  
4252 specification SHALL be that of the "xf:match" function with the arguments reversed [XF  
4253 Section 6.3.15.1].

4254 • x500Name-match

4255 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-  
4256 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It  
4257 shall return "True" if and only if the first argument matches some terminal sequence of  
4258 RDNs from the second argument when compared using x500Name-equal.

4259 • rfc822Name-match

4260 This function SHALL take two arguments, the first is of type  
4261 "http://www.w3.org/2001/XMLSchema#string" and the second is of type  
4262 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an  
4263 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if  
4264 the first argument matches the second argument according to the following specification.

4265 An RFC822 name consists of a local-part followed by "@" followed by domain-part. The  
4266 local-part is case-sensitive, while the domain-part (which is usually a DNS name) is not  
4267 case-sensitive.<sup>4</sup>

4268 The second argument contains a complete rfc822Name. The first argument is a complete  
4269 or partial rfc822Name used to select appropriate values in the second argument as follows.

4270 In order to match a particular mailbox in the second argument, the first argument must  
4271 specify the complete mail address to be matched. For example, if the first argument is  
4272 "Anderson@sun.com", this matches a value in the second argument of  
4273 "Anderson@sun.com" and "Anderson@SUN.COM", but not "Anne.Anderson@sun.com",  
4274 "anderson@sun.com" or "Anderson@east.sun.com".

4275 In order to match any mail address at a particular domain in the second argument, the first  
4276 argument must specify only a domain name (usually a DNS name). For example, if the first  
4277 argument is "sun.com", this matches a value in the first argument of "Anderson@sun.com?"  
4278 or "Baxter@SUN.COM", but not "Anderson@east.sun.com".

4279 In order to match any mail address in a particular domain in the second argument, the first  
4280 argument must specify the desired domain-part with a leading ".". For example, if the first  
4281 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.

4282 "Anderson@east.sun.com" and "anne.anderson@ISRG.EAST.SUN.COM" but not  
4283 "Anderson@sun.com".

## 4284 **A14.13 XPath-based functions**

4285 This section specifies functions that take XPath expressions for arguments. An XPath expression  
4286 evaluates to a *node-set*, which is a set of XML nodes that match the expression. A node or node-  
4287 set is not in the formal type system of XACML. All comparison or other operations on node-sets are  
4288 performed in the isolation of the particular function specified. The XPath expressions in these  
4289 functions are restricted to the XACML request *context*. The following functions are defined:

- 4290 • xpath-node-count

4291 This function SHALL take an "http://www.w3.org/2001/XMLSchema#string" as an  
4292 argument, which SHALL be interpreted as an XPath expression, and evaluates to an  
4293 "http://www.w3.org/2001/XMLSchema#integer". The value returned from the function  
4294 SHALL be the count of the nodes within the node-set that matches the given XPath  
4295 expression.

- 4296 • xpath-node-equal

4297 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,  
4298 which SHALL be interpreted as XPath expressions, and SHALL return an  
4299 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if any  
4300 XML node from the node-set matched by the first argument equals according to the  
4301 "op:node-equal" function [XQO] any XML node from the node-set matched by the second  
4302 argument.

- 4303 • xpath-node-match

4304 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,  
4305 which SHALL be interpreted as XPath expressions and SHALL return an  
4306 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL first extend the first  
4307 argument to match an XML document in a hierarchical fashion. If *a* is an XPath expression  
4308 and it is specified as the first argument, it SHALL be interpreted to mean match the set of  
4309 nodes specified by the enhanced XPath expression "*a* | *a*/\* | *a*/\*@\*". In other words, the  
4310 expression *a* SHALL match all elements and attributes below the element specified by *a*.  
4311 This function SHALL evaluate to "True" if any XML node that matches the enhanced XPath  
4312 expression is equal according to "op:node-equal" [XQO] to any XML node from the node-  
4313 set matched by the second argument.

## 4314 **A14.14 Extension functions and primitive types**

4315 Functions and primitive types are specified by string identifiers allowing for the introduction of  
4316 functions in addition to those specified by XACML. This approach allows one to extend the XACML  
4317 module with special functions and special primitive data types.

4318 In order to preserve some integrity to the XACML evaluation strategy, the result of all function  
4319 applications SHALL depend only on the values of its arguments. Global and hidden parameters  
4320 SHALL NOT affect the evaluation of an expression. Functions SHALL NOT have side effects, as  
4321 evaluation order cannot be guaranteed in a standard way.

---

## 4322 Appendix B. XACML identifiers (normative)

4323 This section defines standard identifiers for commonly used entities. All XACML-defined identifiers  
4324 have the common base:

---

4325 `urn:oasis:names:tc:xacml:1.0`

---

### 4326 B.1. XACML namespaces

4327 There are currently two defined XACML namespaces.

4328 Policies are defined using this identifier.

---

4329 `urn:oasis:names:tc:xacml:1.0:policy`

---

4330 Request and response *contexts* are defined using this identifier.

---

4331 `urn:oasis:names:tc:xacml:1.0:context`

---

4332 XACML data-types are defined using this identifier.

---

4333 `urn:oasis:names:tc:xacml:1.0:data-type`

---

### 4334 B.2. Access subject categories

4335 This identifier indicates the system entity that is directly requesting **access**. That is, the final entity  
4336 in a request chain. If **subject** category is not specified, this is the default value.

---

4337 `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

---

4338 This identifier indicates the system entity that will receive the results of the request. Used when it is  
4339 distinct from the access-subject.

---

4340 `urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject`

---

4341 This identifier indicates a system entity through which the **access** request was passed. There may  
4342 be more than one. No means is provided to specify the order in which they passed the message.

---

4343 `urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject`

---

4344 This identifier indicates a system entity associated with a local or remote codebase that generated  
4345 the request. Corresponding **subject attributes** might include the URL from which it was loaded  
4346 and/or the identity of the code-signer. There may be more than one. No means is provided to  
4347 specify the order they processed the request.

---

4348 `urn:oasis:names:tc:xacml:1.0:subject-category:codebase`

---

4349 This identifier indicates a system entity associated with the computer that initiated the **access**  
4350 request. An example would be an IPsec identity.

---

4351 `urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine`

---

### 4352 B.3. XACML functions

4353 This identifier is the base for all the identifiers in the table of functions. See Section A.1.

---

4354 urn:oasis:names:tc:xacml:1.0:function

---

## 4355 B.4. Data types

4356 The following identifiers indicate useful data-types.

4357 X.500 distinguished name

---

4358 urn:oasis:names:tc:xacml:1.0:data-type:x500Name

---

4359 An x500Name contains an ITU-T Rec. X.520 Distinguished Name. The valid syntax for such a  
4360 name is described in IETF RFC 2253 "Lightweight Directory Access Protocol (v3): UTF-8 String  
4361 Representation of Distinguished Names".

4362 RFC822 Name

---

4363 urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name

---

4364 An rfc822Name contains an "e-mail name". The valid syntax for such a name is described in IETF  
4365 RFC 2821, Section 4.1.2, Command Argument Syntax, under the term "Mailbox".

4366 The following data type identifiers are defined by XML Schema and XQuery.

---

4367 http://www.w3.org/2001/XMLSchema:string  
4368 http://www.w3.org/2001/XMLSchema:boolean  
4369 http://www.w3.org/2001/XMLSchema:integer  
4370 http://www.w3.org/2001/XMLSchema:double  
4371 http://www.w3.org/2001/XMLSchema:date  
4372 http://www.w3.org/2001/XMLSchema:dateTime  
4373 http://www.w3.org/2001/XMLSchema:anyURI  
4374 http://www.w3.org/2001/XMLSchema:hexBinary  
4375 http://www.w3.org/2001/XMLSchema:base64Binary  
4376 http://www.w3.org/2002/08/xquery-functions:dayTimeDuration  
4377 http://www.w3.org/2002/08/xquery-functions:yearMonthDuration

---

## 4378 B.5. Subject attributes

4379 These identifiers indicate *attributes* of a *subject*. When used, they SHALL appear within a  
4380 <Subject> element of the request *context*. They SHALL be accessed via a  
4381 <SubjectAttributeDesignator> or an <AttributeSelector> element pointing into a  
4382 <Subject> element of the request *context*.

4383 At most one of each of these attributes is associated with each subject. Each attribute associated  
4384 with authentication included within a single <Subject> element relates to the same authentication  
4385 event.

4386 This identifier indicates the name of the *subject*. The default format is  
4387 http://www.w3.org/2001/XMLSchema#string. To indicate other formats, use `DataType` attributes  
4388 listed in B.4

---

4389 urn:oasis:names:tc:xacml:1.0:subject:subject-id

---

4390 This identifier indicates the *subject* category. "access-subject" is the default.

---

4391 urn:oasis:names:tc:xacml:1.0:subject:subject-category

---

4392 This identifier indicates the security domain of the *subject*. It identifies the administrator and policy  
4393 that manages the name-space in which the *subject* id is administered.

---

4394 urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier

---

4395	This identifier indicates a public key used to confirm the <b>subject's</b> identity.
4396	<code>urn:oasis:names:tc:xacml:1.0:subject:key-info</code>
4397	This identifier indicates the time at which the <b>subject</b> was authenticated.
4398	<code>urn:oasis:names:tc:xacml:1.0:subject:authentication-time</code>
4399	This identifier indicates the method used to authenticate the <b>subject</b> .
4400	<code>urn:oasis:names:tc:xacml:1.0:subject:authentication-method</code>
4401 4402	This identifier indicates the time at which the <b>subject</b> initiated the <b>access</b> request, according to the <b>PEP</b> .
4403	<code>urn:oasis:names:tc:xacml:1.0:subject:request-time</code>
4404 4405	This identifier indicates the time at which the <b>subject's</b> current session began, according to the <b>PEP</b> .
4406	<code>urn:oasis:names:tc:xacml:1.0:subject:session-start-time</code>
4407 4408	The following identifiers indicate the location where authentication credentials were activated. They are intended to support the corresponding entities from the SAML authentication statement.
4409	This identifier indicates that the location is expressed as an IP address.
4410	<code>urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address</code>
4411	This identifier indicates that the location is expressed as a DNS name.
4412	<code>urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name</code>
4413 4414 4415	Where a suitable attribute is already defined in LDAP [LDAP-1, LDAP-2], the XACML identifier SHALL be formed by adding the <b>attribute</b> name to the URI of the LDAP specification. For example, the <b>attribute</b> name for the userPassword defined in the rfc2256 SHALL be:
4416	<code>http://www.ietf.org/rfc/rfc2256.txt#userPassword</code>

## 4417 B.6. Resource attributes

4418 4419 4420 4421	These identifiers indicate <b>attributes</b> of the <b>resource</b> . When used, they SHALL appear within the <Resource> element of the request <b>context</b> . They SHALL be accessed via a <ResourceAttributeDesignator> or an <AttributeSelector> element pointing into the <Resource> element of the request <b>context</b> .
4422	This identifier indicates the entire URI of the <b>resource</b> .
4423	<code>urn:oasis:names:tc:xacml:1.0:resource:resource-id</code>
4424	A <b>resource attribute</b> used to indicate values extracted from the <b>resource</b> .
4425	<code>urn:oasis:names:tc:xacml:1.0:resource:resource-content</code>
4426 4427	This identifier indicates the last (rightmost) component of the file name. For example, if the URI is: "file://home/my/status#pointer", the simple-file-name is "status".
4428	<code>urn:oasis:names:tc:xacml:1.0:resource:simple-file-name</code>
4429	This identifier indicates that the <b>resource</b> is specified by an XPath expression.
4430	<code>urn:oasis:names:tc:xacml:1.0:resource:xpath</code>
4431	This identifier indicates a UNIX file-system path.

---

4432	urn:oasis:names:tc:xacml:1.0:resource:ufs-path
4433	This identifier indicates the scope of the <b>resource</b> , as described in Section 7.8.
4434	urn:oasis:names:tc:xacml:1.0:resource:scope
4435	The allowed value for this attribute is of type <code>http://www.w3.org/2001/XMLSchema#string</code> , and is either "Immediate", "Children" or "Descendants".

---

## 4437 B.7. Action attributes

4438	These identifiers indicate <b>attributes</b> of the <b>action</b> being requested. When used, they SHALL appear within the <code>&lt;Action&gt;</code> element of the request <b>context</b> . They SHALL be accessed via an
4439	<code>&lt;ActionAttributeDesignator&gt;</code> or an <code>&lt;AttributeSelector&gt;</code> element pointing into the
4440	<code>&lt;Action&gt;</code> element of the request <b>context</b> .
4441	
4442	urn:oasis:names:tc:xacml:1.0:action:action-id
4443	Action namespace
4444	urn:oasis:names:tc:xacml:1.0:action:action-namespace
4445	Implied action. This is the value for action-id attribute when action is implied.
4446	urn:oasis:names:tc:xacml:1.0:action:implied-action

---

## 4447 B.8. Environment attributes

4448	These identifiers indicate <b>attributes</b> of the <b>environment</b> within which the <b>decision request</b> is to be
4449	evaluated. When used, they SHALL appear within the <code>&lt;Resource&gt;</code> element of the request
4450	<b>context</b> . They SHALL be accessed via an <code>&lt;EnvironmentAttributeDesignator&gt;</code> or an
4451	<code>&lt;AttributeSelector&gt;</code> element pointing into the <code>&lt;Environment&gt;</code> element of the request
4452	<b>context</b> .
4453	This identifier indicates the current time at the <b>PDP</b> . In practice it is the time at which the request
4454	<b>context</b> was created.
4455	urn:oasis:names:tc:xacml:1.0:environment:current-time
4456	urn:oasis:names:tc:xacml:1.0:environment:current-date
4457	urn:oasis:names:tc:xacml:1.0:environment:current-dateTime

---

## 4458 B.9. Status codes

4459	The following status code identifiers are defined.
4460	This identifier indicates success.
4461	urn:oasis:names:tc:xacml:1.0:status:ok
4462	This identifier indicates that attributes necessary to make a policy decision were not available.
4463	urn:oasis:names:tc:xacml:1.0:status:missing-attribute
4464	This identifier indicates that some attribute value contained a syntax error, such as a letter in a
4465	numeric field.
4466	urn:oasis:names:tc:xacml:1.0:status:syntax-error

---

4467 This identifier indicates that an error occurred during policy evaluation. An example would be  
4468 division by zero.  
4469 `urn:oasis:names:tc:xacml:1.0:status:processing-error`

---

## 4470 **B.10. Combining algorithms**

4471 The deny-overrides rule-combining algorithm has the following value for `ruleCombiningAlgId`:  
4472 `urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides`

---

4473 The deny-overrides policy-combining algorithm has the following value for  
4474 `policyCombiningAlgId`:  
4475 `urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides`

---

4476 The permit-overrides rule-combining algorithm has the following value for `ruleCombiningAlgId`:  
4477 `urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides`

---

4478 The permit-overrides policy-combining algorithm has the following value for  
4479 `policyCombiningAlgId`:  
4480 `urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides`

---

4481 The first-applicable rule-combining algorithm has the following value for `ruleCombiningAlgId`:  
4482 `urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable`

---

4483 The first-applicable policy-combining algorithm has the following value for  
4484 `policyCombiningAlgId`:  
4485 `urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable`

---

4486 The only-one-applicable-policy policy-combining algorithm has the following value for  
4487 `policyCombiningAlgId`:  
4488 `urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable-policy`

---

4489

## Appendix C. Combining algorithms (normative)

4490

This section contains a description of the rule-combining and policy-combining algorithms specified by XACML.

4491

4492

### C.1. Deny-overrides

4493

The following specification defines the “Deny-overrides” *rule-combining algorithm* of a *policy*.

4494

4495

4496

4497

4498

4499

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 "Not-applicable", 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 "Not-applicable" to the *decision request*, then the *rule* combination SHALL evaluate to "Not-applicable".

4500

4501

4502

4503

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".

4504

4505

4506

If at least one *rule* evaluates to "Permit", all other *rules* that do not have evaluation errors evaluate to "Permit" or "Not-applicable" and all *rules* that do have evaluation errors contain *effects* of "Permit", then the result of the combination SHALL be "Permit".

4507

The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

4508

4509

4510

4511

4512

4513

4514

4515

4516

4517

4518

4519

4520

4521

4522

4523

4524

4525

4526

4527

4528

4529

4530

4531

4532

4533

4534

4535

4536

4537

```
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 == Not-applicable)
    {
      continue;
    }
    if (decision == Indeterminate)
    {
      atLeastOneError = true;

      if (effect(rule[i]) == Deny)
      {
        potentialDeny = true;
      }
    }
    continue;
  }
}
```

```

4538     }
4539   }
4540   if (potentialDeny)
4541   {
4542     return Indeterminate;
4543   }
4544   if (atLeastOnePermit)
4545   {
4546     return Permit;
4547   }
4548   if (atLeastOneError)
4549   {
4550     return Indeterminate;
4551   }
4552   return Not-applicable;
4553 }

```

4554 The following specification defines the "Deny-overrides" **policy-combining algorithm** of a **policy**  
4555 **set**.

4556 In the entire set of **policies** in the **policy set**, if any **policy** evaluates to "Deny", then the  
4557 result of the **policy** combination SHALL be "Deny". In other words, "Deny" takes  
4558 precedence, regardless of the result of evaluating any of the other **policies** in the **policy**  
4559 **set**. If all **policies** are found to be "Not-applicable" to the **decision request**, then the  
4560 **policy set** SHALL evaluate to "Not-applicable".

4561 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is  
4562 considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set**  
4563 SHALL evaluate to "Deny".

4564 The following pseudo-code represents the evaluation strategy of this **policy-combining algorithm**.

```

4565 Decision denyOverridesPolicyCombiningAlgorithm(Policy policy[])
4566 {
4567   Boolean atLeastOnePermit = false;
4568   for( i=0 ; i < lengthOf(policy) ; i++ )
4569   {
4570     Decision decision = evaluate(policy[i]);
4571     if (decision == Deny)
4572     {
4573       return Deny;
4574     }
4575     if (decision == Permit)
4576     {
4577       atLeastOnePermit = true;
4578       continue;
4579     }
4580     if (decision == Not-applicable)
4581     {
4582       continue;
4583     }
4584     if (decision == Indeterminate)
4585     {
4586       return Deny;
4587     }
4588   }
4589   if (atLeastOnePermit)
4590   {
4591     return Permit;
4592   }
4593   return Not-applicable;
4594 }

```

4595 **Obligations** of the individual **policies** shall be combined as described in Section 3.3.2.3.

## 4596 C.2. Permit-overrides

4597 The following specification defines the "Permit-overrides" **rule-combining algorithm** of a **policy**.

4598 In the entire set of **rules** in the **policy**, if any **rule** evaluates to "Permit", then the result of  
4599 the **rule** combination SHALL be "Permit". If any **rule** evaluates to "Deny" and all other  
4600 **rules** evaluate to "Not-applicable", then the **policy** SHALL evaluate to "Deny". In other  
4601 words, "Permit" takes precedence, regardless of the result of evaluating any of the other  
4602 **rules** in the **policy**. If all **rules** are found to be "Not-applicable" to the **decision request**,  
4603 then the **policy** SHALL evaluate to "Not-applicable".

4604 If an error occurs while evaluating the **target** or **condition** of a **rule** that contains an **effect**  
4605 of "Permit" then the evaluation SHALL continue looking for a result of "Permit". If no other  
4606 **rule** evaluates to "Permit", then the **policy** SHALL evaluate to "Indeterminate".

4607 If at least one **rule** evaluates to "Deny", all other **rules** that do not have evaluation errors  
4608 evaluate to "Deny" or "Not-applicable" and all **rules** that do have evaluation errors contain  
4609 an **effect** value of "Deny", then the **policy** SHALL evaluate to "Deny".

4610 The following pseudo-code represents the evaluation strategy of this **rule-combining algorithm**.

```
4611 Decision permitOverridesRuleCombiningAlgorithm(Rule rule[])
4612 {
4613     Boolean atLeastOneError = false;
4614     Boolean potentialPermit = false;
4615     Boolean atLeastOneDeny = false;
4616     for( i=0 ; i < lengthOf(rule) ; i++ )
4617     {
4618         Decision decision = evaluate(rule[i]);
4619         if (decision == Deny)
4620         {
4621             atLeastOneDeny = true;
4622             continue;
4623         }
4624         if (decision == Permit)
4625         {
4626             return Permit;
4627         }
4628         if (decision == Not-applicable)
4629         {
4630             continue;
4631         }
4632         if (decision == Indeterminate)
4633         {
4634             atLeastOneError = true;
4635
4636             if (effect(rule[i]) == Permit)
4637             {
4638                 potentialPermit = true;
4639             }
4640             continue;
4641         }
4642     }
4643     if (potentialPermit)
4644     {
4645         return Indeterminate;
4646     }
```

```

4647     if (atLeastOneDeny)
4648     {
4649         return Deny;
4650     }
4651     if (atLeastOneError)
4652     {
4653         return Indeterminate;
4654     }
4655     return Not-applicable;
4656 }

```

4657 The following specification defines the "Permit-overrides" *policy-combining algorithm* of a *policy*  
4658 *set*.

4659 In the entire set of *policies* in the *policy set*, if any *policy* evaluates to "Permit", then the  
4660 result of the *policy* combination SHALL be "Permit". In other words, "Permit" takes  
4661 precedence, regardless of the result of evaluating any of the other *policies* in the *policy*  
4662 *set*. If all *policies* are found to be "Not-applicable" to the *decision request*, then the  
4663 *policy set* SHALL evaluate to "Not-applicable".

4664 If an error occurs while evaluating the *target* of a *policy*, a reference to a *policy* is  
4665 considered invalid or the *policy* evaluation results in "Indeterminate", then the *policy set*  
4666 SHALL evaluate to "Indeterminate" provided no other *policies* evaluate to "Permit" or  
4667 "Deny".

4668 The following pseudo-code represents the evaluation strategy of this *policy-combining algorithm*.

```

4669 Decision permitOverridesPolicyCombiningAlgorithm(Policy policy[])
4670 {
4671     Boolean atLeastOneError = false;
4672     Boolean atLeastOneDeny = false;
4673     for( i=0 ; i < lengthOf(policy) ; i++ )
4674     {
4675         Decision decision = evaluate(policy[i]);
4676         if (decision == Deny)
4677         {
4678             atLeastOneDeny = true;
4679             continue;
4680         }
4681         if (decision == Permit)
4682         {
4683             return Permit;
4684         }
4685         if (decision == Not-applicable)
4686         {
4687             continue;
4688         }
4689         if (decision == Indeterminate)
4690         {
4691             atLeastOneError = true;
4692             continue;
4693         }
4694     }
4695     if (atLeastOneDeny)
4696     {
4697         return Deny;
4698     }
4699     if (atLeastOneError)
4700     {
4701         return Indeterminate;
4702     }
4703     return Not-applicable;

```

4704

```
}
```

4705 **Obligations** of the individual policies shall be combined as described in Section 3.3.2.3.

## 4706 C.3. First-applicable

4707 The following specification defines the "First-Applicable " **rule-combining algorithm** of a **policy**.

4708 Each **rule** SHALL be evaluated in the order in which it is listed in the **policy**. For a  
4709 particular **rule**, if the **target** matches and the **condition** evaluates to "True", then the  
4710 evaluation of the **policy** SHALL halt and the corresponding **effect** of the **rule** SHALL be the  
4711 result of the evaluation of the **policy** (i.e. "Permit" or "Deny"). For a particular **rule** selected  
4712 in the evaluation, if the **target** evaluates to "False" or the **condition** evaluates to "False",  
4713 then the next **rule** in the order SHALL be evaluated. If no further **rule** in the order exists,  
4714 then the **policy** SHALL evaluate to "Not-applicable".

4715 If an error occurs while evaluating the **target** or **condition** of a **rule**, then the evaluation  
4716 SHALL halt, and the **policy** shall evaluate to "Indeterminate", with the appropriate error  
4717 status.

4718 The following pseudo-code represents the evaluation strategy of this **rule-combining algorithm**.

```
4719 Decision firstApplicableEffectRuleCombiningAlgorithm(Rule rule[])
4720 {
4721     for( i = 0 ; i < lengthOf(rule) ; i++ )
4722     {
4723         Decision decision = evaluate(rule[i]);
4724         if (decision == Deny)
4725         {
4726             return Deny;
4727         }
4728         if (decision == Permit)
4729         {
4730             return Permit;
4731         }
4732         if (decision == Not-applicable)
4733         {
4734             continue;
4735         }
4736         if (decision == Indeterminate)
4737         {
4738             return Indeterminate;
4739         }
4740     }
4741     return Not-applicable;
4742 }
```

4743 The following specification defines the "First-applicable" **policy-combining algorithm** of a **policy**  
4744 **set**.

4745 Each **policy** is evaluated in the order that it appears in the **policy set**. For a particular  
4746 **policy**, if the **target** evaluates to "True" and the **policy** evaluates to a determinate value of  
4747 "Permit" or "Deny", then the evaluation SHALL halt and the **policy set** SHALL evaluate to  
4748 the **effect** value of that **policy**. For a particular **policy**, if the **target** evaluate to "False", or  
4749 the **policy** evaluates to "Not-applicable", then the next **policy** in the order SHALL be  
4750 evaluated. If no further **policy** exists in the order, then the **policy set** SHALL evaluate to  
4751 "Not-applicable".

4752 If an error occurs while evaluating the **target** or the **policy**, or a reference to a **policy** is  
4753 considered invalid, then the evaluation SHALL continue looking for an **applicable policy**, if  
4754 no **applicable policy** is found, then the **policy set** SHALL evaluate to "Indeterminate".

4755 If an error were to occur when evaluating the **target**, or when evaluating a specific **policy**,  
4756 the reference to the **policy** is considered invalid, or the **policy** itself evaluates to  
4757 "Indeterminate", then the evaluation of the **policy-combining algorithm** shall halt, and the  
4758 **policy set** shall evaluate to "Indeterminate" **with an appropriate error status**.

4759 The following pseudo-code represents the evaluation strategy of this **policy-combination**  
4760 **algorithm**.

```
4761 Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy policy[])
4762 {
4763     for( i = 0 ; i < lengthOf(policy) ; i++ )
4764     {
4765         Decision decision = evaluate(policy[i]);
4766         if(decision == Deny)
4767         {
4768             return Deny;
4769         }
4770         if(decision == Permit)
4771         {
4772             return Permit;
4773         }
4774         if (decision == Not-applicable)
4775         {
4776             continue;
4777         }
4778         if (decision == Indeterminate)
4779         {
4780             return Indeterminate;
4781         }
4782     }
4783     return Not-applicable;
4784 }
```

4785 **Obligations** of the individual policies shall be combined as described in Section 3.3.2.3

## 4786 C.4. Only-one-applicable

4787 The following specification defines the "Only-one-applicable" **policy-combining algorithm** of a  
4788 **policy set**.

4789 In the entire set of policies in the **policy set**, if no **policy** is considered applicable by virtue of their  
4790 **targets**, then the result of the policy combination algorithm SHALL be "Not-applicable". If more than  
4791 one policy is considered applicable by virtue of their **targets**, then the result of the policy  
4792 combination algorithm SHALL be "Indeterminate".

4793 If only one **policy** is considered applicable by evaluation of the **policy targets**, then the result of  
4794 the **policy-combining algorithm** SHALL be the result of evaluating the **policy**.

4795 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is considered  
4796 invalid or the **policy** evaluation results in "Indeterminate", then the **policy set** SHALL evaluate to  
4797 "Indeterminate".

4798 The following pseudo-code represents the evaluation strategy of this policy combining algorithm.

```
4799 Decision onlyOneApplicablePolicyPolicyCombiningAlogrithm(Policy policy[])
```

```
4800 {
4801     Boolean          atLeastOne      = false;
4802     Policy           selectedPolicy = null;
4803     ApplicableResult appResult;
4804
4805     for ( i = 0; i < lengthOf(policy) ; i++ )
4806     {
4807         appResult = isApplicable(policy[i]);
4808
4809         if ( appResult == Indeterminate )
4810         {
4811             return Indeterminate;
4812         }
4813         if( appResult == Applicable )
4814         {
4815             if ( atLeastOne )
4816             {
4817                 return Indeterminate;
4818             }
4819             else
4820             {
4821                 atLeastOne      = true;
4822                 selectedPolicy = policy[i];
4823             }
4824         }
4825         if ( appResult == NotApplicable )
4826         {
4827             continue;
4828         }
4829     }
4830     if ( atLeastOne )
4831     {
4832         return evaluate(selectedPolicy);
4833     }
4834     else
4835     {
4836         return NotApplicable;
4837     }
4838 }
4839
```

---

4840

## Appendix D. Acknowledgments

4841 The following individuals were voting members of the XACML committee at the time that this  
4842 version of the specification was issued:

4843 CrossLogix Ken Yagen kyagen@crosslogix.com  
4844 CrossLogix Daniel Engovatov dengovatov@crosslogix.com  
4845 Entegriy Hal Lockhart hal.lockhart@entegriy.com  
4846 Entrust Carlisle Adams carlisle.adams@entrust.com  
4847 Entrust Tim Moses tim.moses@entrust.com  
4848 IBM Michiharu Kudo kudo@jp.ibm.com  
4849 OpenNetwork Steve Andersen sanderson@opennetwork.com  
4850 Overxeer Bill Parducci bill.parducci@overxeer.com  
4851 Overxeer Simon Godik simon.godik@overxeer.com  
4852 Pervasive Security Systems Steve Crocker steve.crocker@pervasivesec.com  
4853 Quadrasis Don Flinn Don.Flinn@hitachisoftware.com  
4854 Quadrasis Konstantin Beznosov konstantin.beznosov@quadrasis.com  
4855 Self Polar Humenn polar@syr.edu  
4856 Sun Microsystems Anne Anderson Anne.Anderson@Sun.com  
4857 Xtradyne Gerald Brose Gerald.Brose@xtradyne.com

4858

---

## Appendix E. Revision history

Rev	Date	By whom	What
V1.0	6 Nov 2002	XACML Technical Committee	First committee specification.

4859

---

## Appendix F. Notices

4860

4861 OASIS takes no position regarding the validity or scope of any intellectual property or other rights  
4862 that might be claimed to pertain to the implementation or use of the technology described in this  
4863 document or the extent to which any license under such rights might or might not be available;  
4864 neither does it represent that it has made any effort to identify any such rights. Information on  
4865 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS  
4866 website. Copies of claims of rights made available for publication and any assurances of licenses to  
4867 be made available, or the result of an attempt made to obtain a general license or permission for  
4868 the use of such proprietary rights by implementors or users of this specification, can be obtained  
4869 from the OASIS Executive Director.

4870 OASIS has been notified of intellectual property rights claimed in regard to some or all of the  
4871 contents of this specification. For more information consult the online list of claimed rights.

4872 OASIS invites any interested party to bring to its attention any copyrights, patents or patent  
4873 applications, or other proprietary rights which may cover technology that may be required to  
4874 implement this specification. Please address the information to the OASIS Executive Director.

4875 Copyright (C) OASIS Open 2002. All Rights Reserved.

4876 This document and translations of it may be copied and furnished to others, and derivative works  
4877 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
4878 published and distributed, in whole or in part, without restriction of any kind, provided that the above  
4879 copyright notice and this paragraph are included on all such copies and derivative works. However,  
4880 this document itself may not be modified in any way, such as by removing the copyright notice or  
4881 references to OASIS, except as needed for the purpose of developing OASIS specifications, in  
4882 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights  
4883 document must be followed, or as required to translate it into languages other than English.

4884 The limited permissions granted above are perpetual and will not be revoked by OASIS or its  
4885 successors or assigns.

4886 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
4887 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
4888 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY  
4889 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
4890 PARTICULAR PURPOSE.