



1
2 **eXtensible Access Control Markup Language**
3 **(XACML) Version 2.0**

4 **OASIS Standard, 1 Feb 2005**

5 Document Identifier: `oasis-access_control-xacml-2.0-core-spec-os`

6 Location: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

7 **Editor:**

8 Tim Moses, Entrust Inc. (tim.moses@entrust.com)

9 **Abstract:**

10 This specification defines version 2.0 of the extensible access-control markup language.

11 **Status:**

12 This version of the specification is an approved OASIS Standard within the OASIS Access
13 Control TC.

14 Access Control TC members should send comments on this specification to the
15 xacml@lists.oasis-open.org list. Others may use the following link and complete the
16 comment form: http://oasis-open.org/committees/comments/form.php?wg_abbrev=xacml.

17 For information on whether any patents have been disclosed that may be essential to
18 implementing this specification, and any offers of patent licensing terms, please refer to the
19 Intellectual Property Rights section of the Access Control TC web page (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml).

20 For any errata page for this specification, please refer to the Access Control TC web page
21 (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml).

22 The non-normative errata page for this specification is located at

23 www.oasis-open.org/committees/access-control.

24 Copyright © OASIS Open 2004-2005 All Rights Reserved.

26 Table of contents

27	1.	Introduction (non-normative).....	8
28	1.1.	Glossary	8
29	1.1.1	Preferred terms.....	8
30	1.1.2	Related terms	10
31	1.2.	Notation	10
32	1.3.	Schema organization and namespaces	10
33	2.	Background (non-normative)	11
34	2.1.	Requirements	11
35	2.2.	Rule and policy combining	12
36	2.3.	Combining algorithms.....	12
37	2.4.	Multiple subjects	13
38	2.5.	Policies based on subject and resource attributes.....	13
39	2.6.	Multi-valued attributes	14
40	2.7.	Policies based on resource contents.....	14
41	2.8.	Operators.....	14
42	2.9.	Policy distribution.....	15
43	2.10.	Policy indexing.....	15
44	2.11.	Abstraction layer	16
45	2.12.	Actions performed in conjunction with enforcement.....	16
46	3.	Models (non-normative)	16
47	3.1.	Data-flow model.....	16
48	3.2.	XACML context.....	16
49	3.3.	Policy language model	16
50	3.3.1	Rule	16
51	3.3.2	Policy	16
52	3.3.3	Policy set	16
53	4.	Examples (non-normative).....	16
54	4.1.	Example one.....	16
55	4.1.1	Example policy	16
56	4.1.2	Example request context.....	16
57	4.1.3	Example response context	16
58	4.2.	Example two	16
59	4.2.1	Example medical record instance	16
60	4.2.2	Example request context.....	16
61	4.2.3	Example plain-language rules	16

62	4.2.4	Example XACML rule instances.....	16
63	5.	Policy syntax (normative, with the exception of the schema fragments).....	16
64	5.1.	Element <PolicySet>	16
65	5.2.	Element <Description>	16
66	5.3.	Element <PolicySetDefaults>	16
67	5.4.	Element <XPathVersion>	16
68	5.5.	Element <Target>	16
69	5.6.	Element <Subjects>	16
70	5.7.	Element <Subject>	16
71	5.8.	Element <SubjectMatch>	16
72	5.9.	Element <Resources>.....	16
73	5.10.	Element <Resource>.....	16
74	5.11.	Element <ResourceMatch>.....	16
75	5.12.	Element <Actions>	16
76	5.13.	Element <Action>	16
77	5.14.	Element <ActionMatch>	16
78	5.15.	Element <Environments>	16
79	5.16.	Element <Environment>	16
80	5.17.	Element <EnvironmentMatch>	16
81	5.18.	Element <PolicySetIdReference>	16
82	5.19.	Element <PolicyIdReference>	16
83	5.20.	Simple type VersionType.....	16
84	5.21.	Simple type VersionMatchType.....	16
85	5.22.	Element <Policy>.....	16
86	5.23.	Element <PolicyDefaults>	16
87	5.24.	Element <CombinerParameters>.....	16
88	5.25.	Element <CombinerParameter>.....	16
89	5.26.	Element <RuleCombinerParameters>	16
90	5.27.	Element <PolicyCombinerParameters>	16
91	5.28.	Element <PolicySetCombinerParameters>.....	16
92	5.29.	Element <Rule>	16
93	5.30.	Simple type EffectType.....	16
94	5.31.	Element <VariableDefinition>	16
95	5.32.	Element <VariableReference>	16
96	5.33.	Element <Expression>	16
97	5.34.	Element <Condition>	16

98	5.35.	Element <Apply>	16
99	5.36.	Element <Function>	16
100	5.37.	Complex type AttributeDesignatorType.....	16
101	5.38.	Element <SubjectAttributeDesignator>	16
102	5.39.	Element <ResourceAttributeDesignator>.....	16
103	5.40.	Element <ActionAttributeDesignator>	16
104	5.41.	Element <EnvironmentAttributeDesignator>.....	16
105	5.42.	Element <AttributeSelector>	16
106	5.43.	Element <AttributeValue>.....	16
107	5.44.	Element <Obligations>	16
108	5.45.	Element <Obligation>	16
109	5.46.	Element <AttributeAssignment>.....	16
110	6.	Context syntax (normative with the exception of the schema fragments).....	16
111	6.1.	Element <Request>.....	16
112	6.2.	Element <Subject>	16
113	6.3.	Element <Resource>.....	16
114	6.4.	Element <ResourceContent>	16
115	6.5.	Element <Action>	16
116	6.6.	Element <Environment>	16
117	6.7.	Element <Attribute>.....	16
118	6.8.	Element <AttributeValue>.....	16
119	6.9.	Element <Response>	16
120	6.10.	Element <Result>	16
121	6.11.	Element <Decision>	16
122	6.12.	Element <Status>	16
123	6.13.	Element <StatusCode>	16
124	6.14.	Element <StatusMessage>	16
125	6.15.	Element <StatusDetail>.....	16
126	6.16.	Element <MissingAttributeDetail>	16
127	7.	Functional requirements (normative).....	16
128	7.1.	Policy enforcement point	16
129	7.1.1.	Base PEP	16
130	7.1.2.	Deny-biased PEP	16
131	7.1.3.	Permit-biased PEP	16
132	7.2.	Attribute evaluation.....	16
133	7.2.1.	Structured attributes	16

134	7.2.2.	Attribute bags	16
135	7.2.3.	Multivalued attributes.....	16
136	7.2.4.	Attribute Matching.....	16
137	7.2.5.	Attribute Retrieval	16
138	7.2.6.	Environment Attributes	16
139	7.3.	Expression evaluation	16
140	7.4.	Arithmetic evaluation	16
141	7.5.	Match evaluation	16
142	7.6.	Target evaluation.....	16
143	7.7.	VariableReference Evaluation.....	16
144	7.8.	Condition evaluation.....	16
145	7.9.	Rule evaluation.....	16
146	7.10.	Policy evaluation.....	16
147	7.11.	Policy Set evaluation	16
148	7.12.	Hierarchical resources.....	16
149	7.13.	Authorization decision	16
150	7.14.	Obligations.....	16
151	7.15.	Exception handling	16
152	7.15.1.	Unsupported functionality.....	16
153	7.15.2.	Syntax and type errors	16
154	7.15.3.	Missing attributes.....	16
155	8.	XACML extensibility points (non-normative).....	16
156	8.1.	Extensible XML attribute types.....	16
157	8.2.	Structured attributes	16
158	9.	Security and privacy considerations (non-normative).....	16
159	9.1.	Threat model	16
160	9.1.1.	Unauthorized disclosure	16
161	9.1.2.	Message replay	16
162	9.1.3.	Message insertion	16
163	9.1.4.	Message deletion	16
164	9.1.5.	Message modification.....	16
165	9.1.6.	NotApplicable results.....	16
166	9.1.7.	Negative rules.....	16
167	9.2.	Safeguards	16
168	9.2.1.	Authentication.....	16
169	9.2.2.	Policy administration.....	16

170	9.2.3.	Confidentiality	16
171	9.2.4.	Policy integrity	16
172	9.2.5.	Policy identifiers.....	16
173	9.2.6.	Trust model.....	16
174	9.2.7.	Privacy.....	16
175	10.	Conformance (normative).....	16
176	10.1.	Introduction.....	16
177	10.2.	Conformance tables	16
178	10.2.1.	Schema elements.....	16
179	10.2.2.	Identifier Prefixes.....	16
180	10.2.3.	Algorithms.....	16
181	10.2.4.	Status Codes	16
182	10.2.5.	Attributes.....	16
183	10.2.6.	Identifiers	16
184	10.2.7.	Data-types	16
185	10.2.8.	Functions	16
186	11.	References	16
187	Appendix A.	Data-types and functions (normative).....	16
188	A.1.	Introduction	16
189	A.2.	Data-types.....	16
190	A.3.	Functions	16
191	A.3.1	Equality predicates	16
192	A.3.2	Arithmetic functions	16
193	A.3.3	String conversion functions	16
194	A.3.4	Numeric data-type conversion functions	16
195	A.3.5	Logical functions.....	16
196	A.3.6	Numeric comparison functions	16
197	A.3.7	Date and time arithmetic functions	16
198	A.3.8	Non-numeric comparison functions	16
199	A.3.9	String functions.....	16
200	A.3.10	Bag functions.....	16
201	A.3.11	Set functions.....	16
202	A.3.12	Higher-order bag functions	16
203	A.3.13	Regular-expression-based functions	16
204	A.3.14	Special match functions.....	16
205	A.3.15	XPath-based functions	16

206	A.3.16	Extension functions and primitive types	16
207		Appendix B. XACML identifiers (normative)	16
208	B.1.	XACML namespaces	16
209	B.2.	Access subject categories	16
210	B.3.	Data-types.....	16
211	B.4.	Subject attributes	16
212	B.6.	Resource attributes.....	16
213	B.7.	Action attributes	16
214	B.8.	Environment attributes.....	16
215	B.9.	Status codes	16
216	B.10.	Combining algorithms	16
217		Appendix C. Combining algorithms (normative).....	16
218	C.1.	Deny-overrides.....	16
219	C.2.	Ordered-deny-overrides	16
220	C.3.	Permit-overrides.....	16
221	C.4.	Ordered-permit-overrides.....	16
222	C.5.	First-applicable	16
223	C.6.	Only-one-applicable	16
224		Appendix D. Acknowledgments.....	16
225		Appendix E. Notices	16
226			

228 1. Introduction (non-normative)

229 1.1. Glossary

230 1.1.1 Preferred terms

231 **Access** - Performing an **action**

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

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

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

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

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

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

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

244 **Conjunctive sequence** - a sequence of **predicates** combined using the logical 'AND' operation

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

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

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

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

251 **Disjunctive sequence** - a sequence of **predicates** combined using the logical 'OR' operation

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

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

255 **Named attribute** – A specific instance of an **attribute**, determined by the **attribute** name and type,
256 the identity of the **attribute** holder (which may be of type: **subject**, **resource**, **action** or
257 **environment**) and (optionally) the identity of the issuing authority

258 **Obligation** - An operation specified in a **policy** or **policy set** that should be performed by the **PEP**
259 in conjunction with the enforcement of an **authorization decision**

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

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

263 **Policy-combining algorithm** - The procedure for combining the **decision** and **obligations** from
264 multiple **policies**

265 **Policy decision point (PDP)** - The system entity that evaluates **applicable policy** and renders an
266 **authorization decision**. This term is defined in a joint effort by the IETF Policy Framework
267 Working Group and the Distributed Management Task Force (DMTF)/Common Information Model
268 (CIM) in [RFC3198]. This term corresponds to "Access Decision Function" (ADF) in [ISO10181-3].

269 **Policy enforcement point (PEP)** - The system entity that performs **access control**, by making
270 **decision requests** and enforcing **authorization decisions**. This term is defined in a joint effort by
271 the IETF Policy Framework Working Group and the Distributed Management Task Force
272 (DMTF)/Common Information Model (CIM) in [RFC3198]. This term corresponds to "Access
273 Enforcement Function" (AEF) in [ISO10181-3].

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

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

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

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

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

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

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

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

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

291

1.1.2 Related terms

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

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

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

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

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

299

1.2. Notation

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

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

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

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

311 Listings of XACML schema appear like this.

312

313 [a01] Example code listings appear like this.

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

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

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

326

1.3. Schema organization and namespaces

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

328 urn:oasis:names:tc:xacml:2.0:policy

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

330 urn:oasis:names:tc:xacml:2.0:context

331 2. Background (non-normative)

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

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

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

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

358 2.1. Requirements

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

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

- 368 • To provide a method for basing an **authorization decision** on the contents of an information
369 **resource**.
- 370 • To provide a set of logical and mathematical operators on **attributes** of the **subject, resource**
371 and **environment**.
- 372 • To provide a method for handling a distributed set of **policy** components, while abstracting the
373 method for locating, retrieving and authenticating the **policy** components.
- 374 • To provide a method for rapidly identifying the **policy** that applies to a given action, based upon
375 the values of **attributes** of the **subjects, resource** and **action**.
- 376 • To provide an abstraction-layer that insulates the policy-writer from the details of the application
377 environment.
- 378 • To provide a method for specifying a set of actions that must be performed in conjunction with
379 policy enforcement.

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

383 2.2. Rule and policy combining

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

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

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

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

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

403 2.3. Combining algorithms

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

408 **policy-combining algorithm** defines a procedure for arriving at an **authorization decision** given
409 the individual results of evaluation of a set of **policies**. Standard combining algorithms are defined
410 for:

- 411 • Deny-overrides (Ordered and Unordered),
- 412 • Permit-overrides (Ordered and Unordered),
- 413 • First-applicable and
- 414 • Only-one-applicable.

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

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

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

423 The "Only-one-applicable" **policy-combining algorithm** only applies to **policies**. The result of this
424 combining algorithm ensures that one and only one **policy** or **policy set** is applicable by virtue of
425 their **targets**. If no **policy** or **policy set** applies, then the result is "NotApplicable", but if more than
426 one **policy** or **policy set** is applicable, then the result is "Indeterminate". When exactly one **policy**
427 or **policy set** is applicable, the result of the combining algorithm is the result of evaluating the
428 single **applicable policy** or **policy set**.

429 **Policies** and **policy sets** may take parameters that modify the behaviour of the **combining**
430 **algorithms**. However, none of the standard **combining algorithms** is affected by parameters.

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

432 2.4. Multiple subjects

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

439 2.5. Policies based on subject and resource attributes

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

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

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

457 2.6. Multi-valued attributes

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

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

467 2.7. Policies based on resource contents

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

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

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

478 2.8. Operators

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

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

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

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

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

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

505 The XACML method of representing functions borrows from MathML [\[MathML\]](#) and from the
506 XQuery 1.0 and XPath 2.0 Functions and Operators specification [\[XF\]](#).

507 **2.9. Policy distribution**

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

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

519 **2.10. Policy indexing**

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

525 Two approaches are supported:

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

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

534 **2.11. Abstraction layer**

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

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

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

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

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

557 **2.12. Actions performed in conjunction with enforcement**

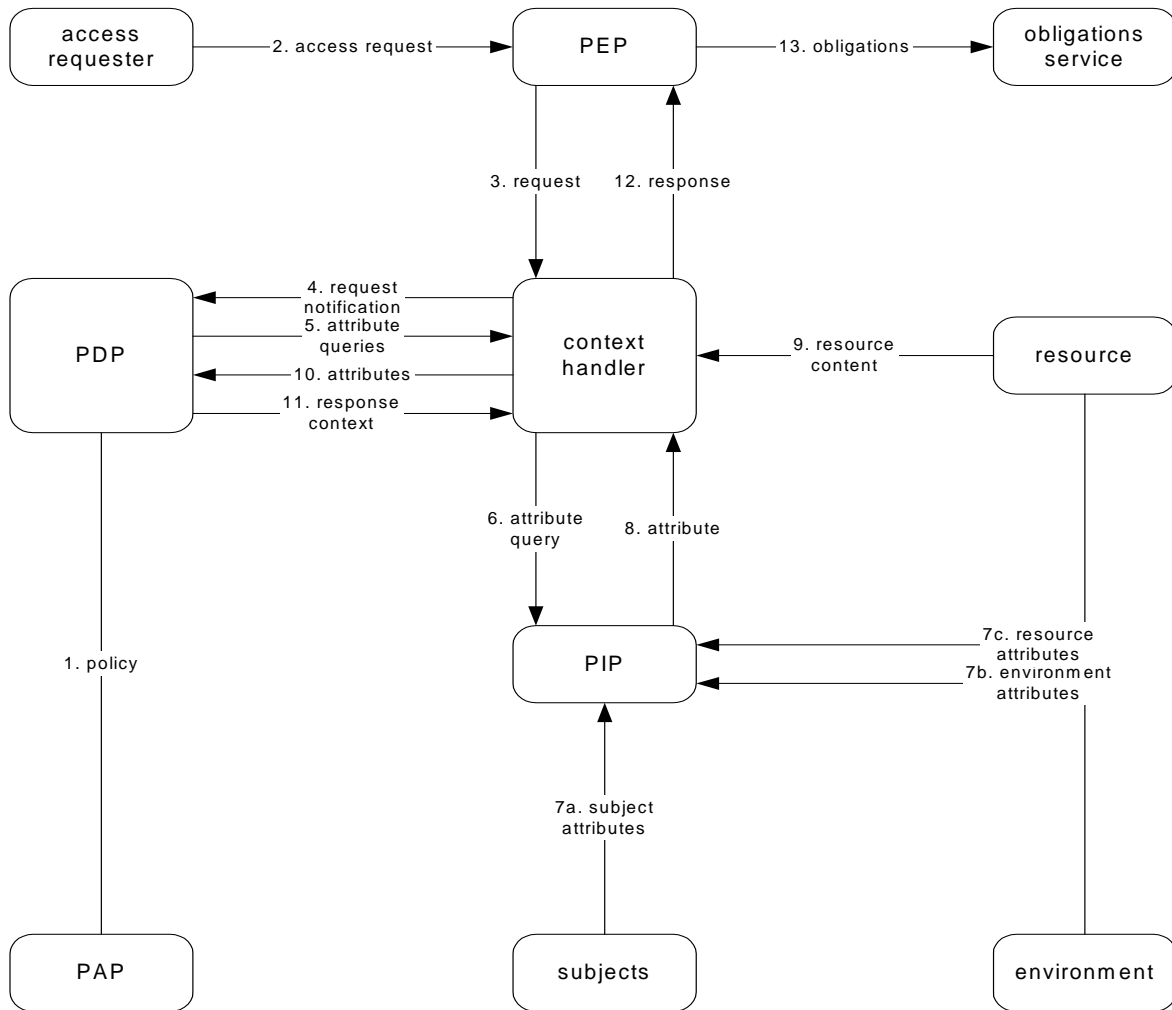
558 In many applications, policies specify actions that MUST be performed, either instead of, or in
559 addition to, actions that MAY be performed. This idea was described by Sloman [Sloman94].
560 XACML provides facilities to specify actions that MUST be performed in conjunction with policy
561 evaluation in the <Obligations> element. This idea was described as a provisional action by
562 Kudo [Kudo00]. There are no standard definitions for these actions in version 2.0 of XACML.
563 Therefore, bilateral agreement between a *PAP* and the *PEP* that will enforce its *policies* is required
564 for correct interpretation. *PEPs* that conform with v2.0 of XACML are required to deny *access*
565 unless they understand and can discharge all of the <Obligations> elements associated with the
566 *applicable policy*. <Obligations> elements are returned to the *PEP* for enforcement.

567 **3. Models (non-normative)**

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

569 **3.1. Data-flow model**

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



571

572

Figure 1 - Data-flow diagram

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

578 The model operates by the following steps.

579 1. **PAPs** write **policies** and **policy sets** and make them available to the **PDP**. These **policies** or
 580 **policy sets** represent the complete policy for a specified **target**.

581 2. The access requester sends a request for access to the **PEP**.

582 3. The **PEP** sends the request for **access** to the **context handler** in its native request format,
 583 optionally including **attributes** of the **subjects**, **resource**, **action** and **environment**.

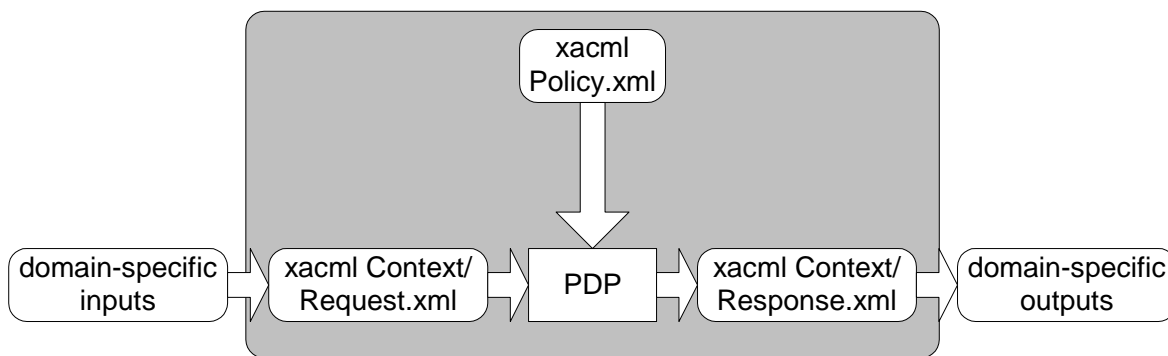
584 4. The **context handler** constructs an XACML request **context** and sends it to the **PDP**.

585 5. The **PDP** requests any additional **subject**, **resource**, **action** and **environment attributes** from
 586 the **context handler**.

- 587 6. The context handler requests the attributes from a *PIP*.
- 588 7. The *PIP* obtains the requested *attributes*.
- 589 8. The *PIP* returns the requested *attributes* to the *context handler*.
- 590 9. Optionally, the *context handler* includes the *resource* in the *context*.
- 591 10. The *context handler* sends the requested *attributes* and (optionally) the *resource* to the *PDP*.
- 592 The *PDP* evaluates the *policy*.
- 593 11. The *PDP* returns the response *context* (including the *authorization decision*) to the *context handler*.
- 594
- 595 12. The *context handler* translates the response *context* to the native response format of the
- 596 *PEP*. The *context handler* returns the response to the *PEP*.
- 597 13. The *PEP* fulfills the *obligations*.
- 598 14. (Not shown) If *access* is permitted, then the *PEP* permits *access* to the *resource*; otherwise, it
- 599 denies *access*.

600 3.2. XACML context

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



613

614

Figure 2 - XACML context

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

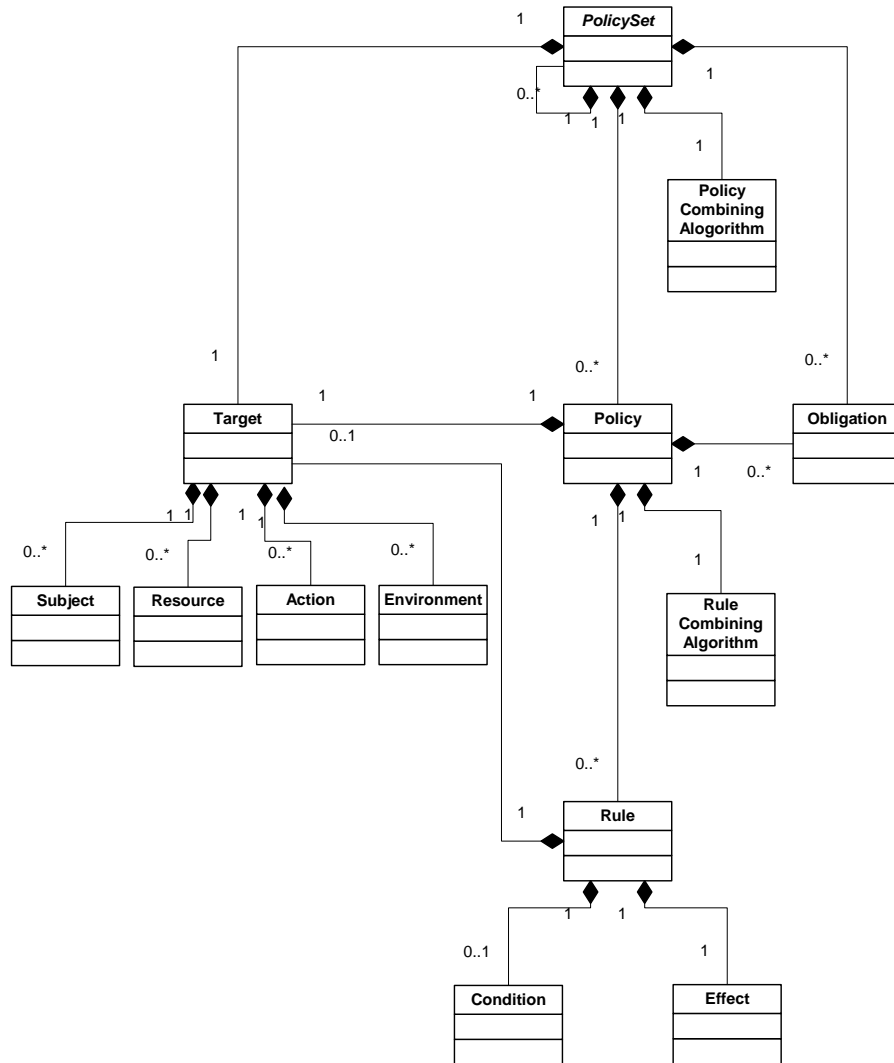
617 See Section 7.2.5 for a more detailed discussion of the request *context*.

618 **3.3. Policy language model**

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

- 620 • **Rule**;
- 621 • **Policy**; and
- 622 • **Policy set**.

623 These are described in the following sub-sections.



624

625

Figure 3 - Policy language model

626 **3.3.1 Rule**

627 A **rule** is the most elementary unit of **policy**. It may exist in isolation only *within* one of the major
 628 actors of the XACML domain. In order to exchange **rules** between major actors, they must be

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

- 631 • a **target**;
- 632 • an **effect** and
- 633 • a **condition**.

634 These are discussed in the following sub-sections.

635 **3.3.1.1. Rule target**

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

- 637 • **resources**;
- 638 • **subjects**;
- 639 • **actions** and
- 640 • **environment**

641 to which the **rule** is intended to apply. The <Condition> element may further refine the
642 applicability established by the **target**. If the **rule** is intended to apply to all entities of a particular
643 data-type, then the corresponding entity is omitted from the **target**. An XACML **PDP** verifies that
644 the matches defined by the **target** are satisfied by the **subjects, resource, action** and
645 **environment attributes** in the request **context**. **Target** definitions are discrete, in order that
646 applicable **rules** may be efficiently identified by the **PDP**.

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

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

654 Generally, the name of a node (other than a leaf node) in a structured name-form is also a legal
655 instance of the name-form. So, for instance, the RFC822 name "med.example.com" is a legal
656 RFC822 name identifying the set of mail addresses hosted by the med.example.com mail server.
657 And the XPath/XPointer value `//xacml-context:Request/xacml-context:Resource/xacml-`
658 `context:ResourceContent/md:record/md:patient/` is a legal XPath/XPointer value identifying a
659 node-set in an XML document.

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

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

669

3.3.1.2. Effect

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

672

3.3.1.3. Condition

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

675

3.3.2 Policy

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

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

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

684

3.3.2.1. Policy target

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

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

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

706

3.3.2.2. Rule-combining algorithm

707 The **rule-combining algorithm** specifies the procedure by which the results of evaluating the
708 component **rules** are combined when evaluating the **policy**, i.e. the `Decision` value placed in the
709 response **context** by the **PDP** is the value of the **policy**, as defined by the **rule-combining**
710 **algorithm**. A **policy** may have combining parameters that affect the operation of the **rule-**
711 **combining algorithm**.

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

713

3.3.2.3. Obligations

714 **Obligations** may be added by the writer of the **policy**.

715 When a **PDP** evaluates a **policy** containing **obligations**, it returns certain of those **obligations** to
716 the **PEP** in the response **context**. Section 7.14 explains which **obligations** are to be returned.

717

3.3.3 Policy set

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

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

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

725

3.3.3.1. Policy-combining algorithm

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

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

732

3.3.3.2. Obligations

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

735 When a **PDP** evaluates a **policy set** containing **obligations**, it returns certain of those **obligations**
736 to the **PEP** in its response **context**. Section 7.14 explains which **obligations** are to be returned.

737

4. Examples (non-normative)

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

742 4.1. Example one

743 4.1.1 Example policy

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

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

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

```

750 [a02] <?xml version="1.0" encoding="UTF-8"?>
751 [a03] <Policy
752 [a04]   xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
753 [a05]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
754 [a06]   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
755 [a07]   http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd"
756 [a07]   PolicyId="urn:oasis:names:tc:example:SimplePolicy1"
757 [a08]   RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-overrides">
758 [a09]   <Description>
759 [a10]     Medi Corp access control policy
760 [a11]   </Description>
761 [a12]   <Target/>
762 [a13]   <Rule
763 [a14]     RuleId="urn:oasis:names:tc:xacml:2.0:example:SimpleRule1"
764 [a15]     Effect="Permit">
765 [a16]     <Description>
766 [a17]       Any subject with an e-mail name in the med.example.com domain
767 [a18]       can perform any action on any resource.
768 [a19]     </Description>
769 [a20]     <Target>
770 [a21]       <Subjects>
771 [a22]         <Subject>
772 [a23]           <SubjectMatch
773 [a24]             MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
774 [a25]             <AttributeValue
775 [a26]               DataType="http://www.w3.org/2001/XMLSchema#string">
776 [a27]               med.example.com
777 [a28]             </AttributeValue>
778 [a29]             <SubjectAttributeDesignator
779 [a30]               AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
780 [a31]               DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
781 [a32]             </SubjectMatch>
782 [a33]           </Subject>
783 [a34]         </Subjects>
784 [a35]       </Target>
785 [a36]     </Rule>
786 [a37]   </Policy>
  
```

787 [a02] is a standard XML document tag indicating which version of XML is being used and what the
788 character encoding is.

789 [a03] introduces the XACML Policy itself.

790 [a04] - [a05] are XML namespace declarations.

791 [a04] gives a URN for the XACML **policies** schema.

792 [a07] assigns a name to this **policy** instance. The name of a **policy** has to be unique for a given
793 **PDP** so that there is no ambiguity if one **policy** is referenced from another **policy**. The `version`
794 attribute is omitted, so it takes its default value of "1.0".

795 [a08] specifies the algorithm that will be used to resolve the results of the various **rules** that may be
796 in the **policy**. The *deny-overrides rule-combining algorithm* specified here says that, if any **rule**
797 evaluates to "Deny", then the **policy** must return "Deny". If all **rules** evaluate to "Permit", then the
798 **policy** must return "Permit". The *rule-combining algorithm*, which is fully described in Appendix
799 C, also says what to do if an error were to occur when evaluating any **rule**, and what to do with
800 **rules** that do not apply to a particular **decision request**.

801 [a09] - [a11] provide a text description of the policy. This description is optional.

802 [a12] describes the **decision requests** to which this **policy** applies. If the **subject, resource,**
803 **action** and **environment** in a **decision request** do not match the values specified in the **policy**
804 **target**, then the remainder of the **policy** does not need to be evaluated. This **target** section is
805 useful for creating an index to a set of **policies**. In this simple example, the **target** section says the
806 **policy** is applicable to any **decision request**.

807 [a13] introduces the one and only **rule** in this simple **policy**.

808 [a14] specifies the identifier for this **rule**. Just as for a **policy**, each **rule** must have a unique
809 identifier (at least unique for any **PDP** that will be using the **policy**).

810 [a15] says what **effect** this **rule** has if the **rule** evaluates to "True". **Rules** can have an **effect** of
811 either "Permit" or "Deny". In this case, if the **rule** is satisfied, it will evaluate to "Permit", meaning
812 that, as far as this one **rule** is concerned, the requested **access** should be permitted. If a **rule**
813 evaluates to "False", then it returns a result of "NotApplicable". If an error occurs when evaluating
814 the **rule**, then the **rule** returns a result of "Indeterminate". As mentioned above, the *rule-*
815 *combining algorithm* for the **policy** specifies how various **rule** values are combined into a single
816 **policy** value.

817 [a16] - [a19] provide a text description of this **rule**. This description is optional.

818 [a20] introduces the **target** of the **rule**. As described above for the **target** of a policy, the **target** of
819 a **rule** describes the **decision requests** to which this **rule** applies. If the **subject, resource,**
820 **action** and **environment** in a **decision request** do not match the values specified in the **rule**
821 **target**, then the remainder of the **rule** does not need to be evaluated, and a value of
822 "NotApplicable" is returned to the **rule** evaluation.

823 The **rule target** is similar to the **target** of the **policy** itself, but with one important difference. [a23]-
824 [a32] spells out a specific value that the **subject** in the **decision request** must match. The
825 <SubjectMatch> element specifies a matching function in the `MatchId` attribute, a literal value of
826 "med.example.com" and a pointer to a specific **subject attribute** in the request **context** by means
827 of the <SubjectAttributeDesignator> element. The matching function will be used to
828 compare the literal value with the value of the **subject attribute**. Only if the match returns "True"
829 will this **rule** apply to a particular **decision request**. If the match returns "False", then this **rule** will
830 return a value of "NotApplicable".

831 [a36] closes the **rule**. In this **rule**, all the *work* is done in the <Target> element. In more complex
832 **rules**, the <Target> may have been followed by a <Condition> element (which could also be a
833 set of **conditions** to be **ANDed** or **ORed** together).

834 [a37] closes the **policy**. As mentioned above, this **policy** has only one **rule**, but more complex
835 **policies** may have any number of **rules**.

836 4.1.2 Example request context

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

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

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

```
844 [a38] <?xml version="1.0" encoding="UTF-8"?>
845 [a39] <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
846 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
847 [a40] xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
848 http://docs.oasis-open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd">
849 [a41] <Subject>
850 [a42] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
851 DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
852 [a43] <AttributeValue>
853 [a44] bs@simpsons.com
854 [a45] </AttributeValue>
855 [a46] </Attribute>
856 [a47] </Subject>
857 [a48] <Resource>
858 [a49] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-
859 id" DataType="http://www.w3.org/2001/XMLSchema#anyURI">
860 [a50] <AttributeValue>
861 [a51] file://example/med/record/patient/BartSimpson
862 [a52] </AttributeValue>
863 [a53] </Attribute>
864 [a54] </Resource>
865 [a55] <Action>
866 [a56] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
867 DataType="http://www.w3.org/2001/XMLSchema#string">
868 [a57] <AttributeValue>
869 [a58] read
870 [a59] </AttributeValue>
871 [a60] </Attribute>
872 [a61] </Action>
873 [a62] <Environment/>
874 [a63] </Request>
```

875 [a38] - [a40] contain the header information for the **request context**, and are used the same way
876 as the header for the **policy** explained above.

877 The <Subject> element contains one or more **attributes** of the entity making the **access** request.
878 There can be multiple **subjects**, and each **subject** can have multiple **attributes**. In this case, in
879 [a41] - [a47], there is only one **subject**, and the **subject** has only one **attribute**: the **subject's**
880 identity, expressed as an e-mail name, is "bs@simpsons.com". In this example, the `subject-`
881 category attribute is omitted. Therefore, it adopts its default value of "access-subject".

882 The <Resource> element contains one or more **attributes** of the **resource** to which the **subject** (or
883 **subjects**) has requested **access**. There can be only one <Resource> per **decision request**¹.
884 Lines [a48] - [a54] contain the one **attribute** of the **resource** to which Bart Simpson has requested
885 **access**: the **resource** identified by its file URI, which is
886 "file://medico/record/patient/BartSimpson".

887 The <Action> element contains one or more **attributes** of the **action** that the **subject** (or
888 **subjects**) wishes to take on the **resource**. There can be only one **action** per **decision request**.
889 [a55] - [a61] describe the identity of the **action** Bart Simpson wishes to take, which is "read".

890 The <Environment> element, [a62], is empty.

891 [a63] closes the **request context**. A more complex **request context** may have contained some
892 **attributes** not associated with the **subject**, the **resource** or the **action**. These would have been
893 placed in an optional <Environment> element following the <Action> element.

894 The **PDP** processing this request **context** locates the **policy** in its policy repository. It compares
895 the **subject**, **resource**, **action** and **environment** in the request **context** with the **subjects**,
896 **resources**, **actions** and **environments** in the **policy target**. Since the **policy target** is empty, the
897 **policy** matches this **context**.

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

902 4.1.3 Example response context

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

```
906 [a64] <?xml version="1.0" encoding="UTF-8"?>  
907 [a65] <Response xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"  
908 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
909 xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os  
910 http://docs.oasis-open.org/xacml/xacml-core-2.0-context-schema-os.xsd">  
911 [a66] <Result>  
912 [a67] <Decision>NotApplicable</Decision>  
913 [a68] </Result>  
914 [a69] </Response>
```

915 [a64] - [a65] contain the same sort of header information for the response as was described above
916 for a **policy**.

917 The <Result> element in lines [a66] - [a68] contains the result of evaluating the **decision request**
918 against the **policy**. In this case, the result is "NotApplicable". A **policy** can return "Permit", "Deny",
919 "NotApplicable" or "Indeterminate". Therefore, the **PEP** is required to deny **access**.

920 [a69] closes the response **context**.

¹ Some exceptions are described in the XACML Profile for Multiple Resources [MULT].

921

4.2. Example two

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

925

4.2.1 Example medical record instance

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

```
929 [a70] <?xml version="1.0" encoding="UTF-8"?>
930 [a71] <record xmlns="urn:example:med:schemas:record"
931 [a72] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
932 [a73] <patient>
933 [a74] <patientName>
934 [a75] <first>Bartholomew</first>
935 [a76] <last>Simpson</last>
936 [a77] </patientName>
937 [a78] <patientContact>
938 [a79] <street>27 Shelbyville Road</street>
939 [a80] <city>Springfield</city>
940 [a81] <state>MA</state>
941 [a82] <zip>12345</zip>
942 [a83] <phone>555.123.4567</phone>
943 [a84] <fax/>
944 [a85] <email/>
945 [a86] </patientContact>
946 [a87] <patientDoB>1992-03-21</patientDoB>
947 [a88] <patientGender>male</patientGender>
948 [a89] <patient-number>555555</patient-number>
949 [a90] </patient>
950 [a91] <parentGuardian>
951 [a92] <parentGuardianId>HS001</parentGuardianId>
952 [a93] <parentGuardianName>
953 [a94] <first>Homer</first>
954 [a95] <last>Simpson</last>
955 [a96] </parentGuardianName>
956 [a97] <parentGuardianContact>
957 [a98] <street>27 Shelbyville Road</street>
958 [a99] <city>Springfield</city>
959 [a100] <state>MA</state>
960 [a101] <zip>12345</zip>
961 [a102] <phone>555.123.4567</phone>
962 [a103] <fax/>
963 [a104] <email>homers@aol.com</email>
964 [a105] </parentGuardianContact>
965 [a106] </parentGuardian>
966 [a107] <primaryCarePhysician>
967 [a108] <physicianName>
968 [a109] <first>Julius</first>
969 [a110] <last>Hibbert</last>
970 [a111] </physicianName>
971 [a112] <physicianContact>
972 [a113] <street>1 First St</street>
973 [a114] <city>Springfield</city>
974 [a115] <state>MA</state>
975 [a116] <zip>12345</zip>
976 [a117] <phone>555.123.9012</phone>
977 [a118] <fax>555.123.9013</fax>
```

```

978 [a119] <email/>
979 [a120] </physicianContact>
980 [a121] <registrationID>ABC123</registrationID>
981 [a122] </primaryCarePhysician>
982 [a123] <insurer>
983 [a124] <name>Blue Cross</name>
984 [a125] <street>1234 Main St</street>
985 [a126] <city>Springfield</city>
986 [a127] <state>MA</state>
987 [a128] <zip>12345</zip>
988 [a129] <phone>555.123.5678</phone>
989 [a130] <fax>555.123.5679</fax>
990 [a131] <email/>
991 [a132] </insurer>
992 [a133] <medical>
993 [a134] <treatment>
994 [a135] <drug>
995 [a136] <name>methylphenidate hydrochloride</name>
996 [a137] <dailyDosage>30mgs</dailyDosage>
997 [a138] <startDate>1999-01-12</startDate>
998 [a139] </drug>
999 [a140] <comment>
1000 [a141] patient exhibits side-effects of skin coloration and carpal
1001 degeneration
1002 [a142] </comment>
1003 [a143] </treatment>
1004 [a144] <result>
1005 [a145] <test>blood pressure</test>
1006 [a146] <value>120/80</value>
1007 [a147] <date>2001-06-09</date>
1008 [a148] <performedBy>Nurse Betty</performedBy>
1009 [a149] </result>
1010 [a150] </medical>
1011 [a151] </record>

```

1012 4.2.2 Example request context

1013 The following example illustrates a request *context* to which the example *rules* may be applicable.
1014 It represents a request by the physician Julius Hibbert to read the patient date of birth in the record
1015 of Bartholomew Simpson.

```

1016 [a152] <?xml version="1.0" encoding="UTF-8"?>
1017 [a153] <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
1018 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
1019 urn:oasis:names:tc:xacml:2.0:context:schema:os http://docs.oasis-
1020 open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd">
1021 [a154] <Subject>
1022 [a155] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject-category"
1023 DataType="http://www.w3.org/2001/XMLSchema#anyURI">
1024 [a156] <AttributeValue>urn:oasis:names:tc:xacml:1.0:subject-category:access-
1025 subject</AttributeValue>
1026 [a157] </Attribute>
1027 [a158] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
1028 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="med.example.com">
1029 [a159] <AttributeValue>CN=Julius Hibbert</AttributeValue>
1030 [a160] </Attribute>
1031 [a161] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:name-
1032 format" DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1033 Issuer="med.example.com">
1034 [a162] <AttributeValue>
1035 [a163] urn:oasis:names:tc:xacml:1.0:datatype:x500name
1036 [a164] </AttributeValue>

```

```

1037 [a165] </Attribute>
1038 [a166] <Attribute
1039 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:role"
1040 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="med.example.com">
1041 [a167] <AttributeValue>physician</AttributeValue>
1042 [a168] </Attribute>
1043 [a169] <Attribute
1044 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:physician-id"
1045 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="med.example.com">
1046 [a170] <AttributeValue>jh1234</AttributeValue>
1047 [a171] </Attribute>
1048 [a172] </Subject>
1049 [a173] <Resource>
1050 [a174] <ResourceContent>
1051 [a175] <md:record xmlns:md="urn:example:med:schemas:record"
1052 xsi:schemaLocation="urn:example:med:schemas:record
1053 http:www.med.example.com/schemas/record.xsd">
1054 [a176] <md:patient>
1055 [a177] <md:patientDoB>1992-03-21</md:patientDoB>
1056 [a178] <md:patient-number>555555</md:patient-number>
1057 [a179] </md:patient>
1058 [a180] </md:record>
1059 [a181] </ResourceContent>
1060 [a182] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-
1061 id" DataType="http://www.w3.org/2001/XMLSchema#string">
1062 [a183] <AttributeValue>
1063 [a184] //med.example.com/records/bart-simpson.xml#
1064 [a185] xmlns(md=:Resource/ResourceContent/xpointer
1065 [a186] (/md:record/md:patient/md:patientDoB)
1066 [a187] </AttributeValue>
1067 [a188] </Attribute>
1068 [a189] </Resource>
1069 [a190] <Action>
1070 [a191] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1071 DataType="http://www.w3.org/2001/XMLSchema#string">
1072 [a192] <AttributeValue>read</AttributeValue>
1073 [a193] </Attribute>
1074 [a194] </Action>
1075 [a195] <Environment/>
1076 [a196] </Request>

```

1077 [a152] - [a153] Standard namespace declarations.

1078 [a154] - [a172] **Subject** attributes are placed in the <Subject> element of the <Request>
1079 element. Each **attribute** consists of the **attribute** meta-data and the **attribute** value. There is only
1080 one subject involved in this **request**.

1081 [a155] - [a157] Each <Subject> element has a SubjectCategory attribute. The value of this
1082 attribute describes the role that the related **subject** plays in making the **decision request**. The
1083 value of "access-subject" denotes the identity for which the request was issued.

1084 [a158] - [a160] **Subject** subject-id **attribute**.

1085 [a161] - [a165] The format of the subject-id.

1086 [a166] - [a168] **Subject** role **attribute**.

1087 [a169] - [a171] **Subject** physician-id **attribute**.

1088 [a173] - [a189] **Resource attributes** are placed in the <Resource> element of the <Request>
1089 element. Each **attribute** consists of **attribute** meta-data and an **attribute** value.

- 1090 [a174] - [a181] **Resource** content. The XML resource instance, access to all or part of which may
1091 be requested, is placed here.
- 1092 [a182] - [a188] The identifier of the **Resource** instance for which access is requested, which is an
1093 XPath expression into the <ResourceContent> element that selects the data to be accessed.
- 1094 [a190] - [a194] **Action attributes** are placed in the <Action> element of the <Request> element.
1095 [a192] **Action** identifier.
- 1096 [a195] The empty <Environment> element.

1097 4.2.3 Example plain-language rules

1098 The following plain-language rules are to be enforced:

- 1099 Rule 1: A person, identified by his or her patient number, may read any record for which he
1100 or she is the designated patient.
- 1101 Rule 2: A person may read any record for which he or she is the designated parent or
1102 guardian, and for which the patient is under 16 years of age.
- 1103 Rule 3: A physician may write to any medical element for which he or she is the designated
1104 primary care physician, provided an email is sent to the patient.
- 1105 Rule 4: An administrator shall not be permitted to read or write to medical elements of a
1106 patient record.

1107 These **rules** may be written by different **PAPs** operating independently, or by a single **PAP**.

1108 4.2.4 Example XACML rule instances

1109 4.2.4.1. Rule 1

1110 Rule 1 illustrates a simple **rule** with a single <Condition> element. It also illustrates the use of
1111 the <VariableDefinition> element to define a function that may be used throughout the
1112 **policy**. The following XACML <Rule> instance expresses Rule 1:

```
1113 [a197] <?xml version="1.0" encoding="UTF-8"?>
1114 [a198] <Policy
1115 [a199] xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" xmlns:xacml-
1116 context="urn:oasis:names:tc:xacml:2.0:context:schema:os"
1117 [a200] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
1118 urn:oasis:names:tc:xacml:2.0:policy:schema:os http://docs.oasis-
1119 open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd"
1120 [a201] xmlns:md="http://www.med.example.com/schemas/record.xsd"
1121 [a202] PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:1"
1122 [a203] RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1123 algorithm:deny-overrides">
1124 [a204] <PolicyDefaults>
1125 [a205] <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-
1126 19991116</XPathVersion>
1127 [a206] </PolicyDefaults>
1128 [a207] <Target/>
1129 [a208] <VariableDefinition VariableId="17590034">
1130 [a209] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1131 [a210] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1132 and-only">
```

```

1133 [a211] <SubjectAttributeDesignator
1134 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:patient-number"
1135 [a212]   DataType="http://www.w3.org/2001/XMLSchema#string" />
1136 [a213] </Apply>
1137 [a214] <Apply
1138 [a215]   FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1139 [a216]   <AttributeSelector
1140 [a217]     RequestContextPath="//xacml-context:Resource/xacml-
1141 context:ResourceContent/md:record/md:patient/md:patient-number/text()"
1142 [a218]     DataType="http://www.w3.org/2001/XMLSchema#string" />
1143 [a219]   </Apply>
1144 [a220] </Apply>
1145 [a221] </VariableDefinition>
1146 [a222] <Rule
1147 [a223]   RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:1"
1148 [a224]   Effect="Permit">
1149 [a225]   <Description>
1150 [a226]     A person may read any medical record in the
1151 [a227]     http://www.med.example.com/schemas/record.xsd namespace
1152 [a228]     for which he or she is the designated patient
1153 [a229]   </Description>
1154 [a230]   <Target>
1155 [a231]     <Resources>
1156 [a232]       <Resource>
1157 [a233]         <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1158 equal">
1159 [a234]           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1160 [a235]             urn:example:med:schemas:record
1161 [a236]           </AttributeValue>
1162 [a237]           <ResourceAttributeDesignator AttributeId=
1163 [a238]             "urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1164 [a239]             DataType="http://www.w3.org/2001/XMLSchema#string" />
1165 [a240]           </ResourceMatch>
1166 [a241]         <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-
1167 node-match">
1168 [a242]           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1169 [a243]             /md:record
1170 [a244]           </AttributeValue>
1171 [a245]           <ResourceAttributeDesignator
1172 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
1173 [a246]             DataType="http://www.w3.org/2001/XMLSchema#string" />
1174 [a247]           </ResourceMatch>
1175 [a248]         </Resource>
1176 [a249]       </Resources>
1177 [a250]     <Actions>
1178 [a251]       <Action>
1179 [a252]         <ActionMatch
1180 [a253]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1181 [a254]           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1182 [a255]             read
1183 [a256]           </AttributeValue>
1184 [a257]           <ActionAttributeDesignator
1185 [a258]             AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1186 [a259]             DataType="http://www.w3.org/2001/XMLSchema#string" />
1187 [a260]           </ActionMatch>
1188 [a261]         </Action>
1189 [a262]       </Actions>
1190 [a263]     </Target>
1191 [a264]   <Condition>
1192 [a265]     <VariableReference VariableId="17590034" />
1193 [a266]   </Condition>
1194 [a267] </Rule>

```

1195 [a268] </Policy>
1196 [a199] - [a201] XML namespace declarations.

1197 [a205] XPath expressions in the **policy** are to be interpreted according to the 1.0 version of the
1198 XPath specification.

1199 [a208] - [a221] A <VariableDefinition> element. It defines a function that evaluates the truth
1200 of the statement: the `patient-number` **subject attribute** is equal to the `patient-number` in the
1201 **resource**.

1202 [a209] The `FunctionId` attribute names the function to be used for comparison. In this case,
1203 comparison is done with the “urn:oasis:names:tc:xacml:1.0:function:string-equal” function; this
1204 function takes two arguments of type “http://www.w3.org/2001/XMLSchema#string”.

1205 [a210] The first argument of the variable definition is a function specified by the `FunctionId`
1206 attribute. Since `urn:oasis:names:tc:xacml:1.0:function:string-equal` takes
1207 arguments of type “http://www.w3.org/2001/XMLSchema#string” and
1208 `SubjectAttributeDesignator` selects a **bag** of type
1209 “http://www.w3.org/2001/XMLSchema#string”, “urn:oasis:names:tc:xacml:1.0:function:string-one-
1210 and-only” is used. This function guarantees that its argument evaluates to a **bag** containing exactly
1211 one value.

1212 [a211] The `SubjectAttributeDesignator` selects a **bag** of values for the `patient-number`
1213 **subject attribute** in the request **context**.

1214 [a215] The second argument of the variable definition is a function specified by the `FunctionId`
1215 attribute. Since “urn:oasis:names:tc:xacml:1.0:function:string-equal” takes arguments of type
1216 “http://www.w3.org/2001/XMLSchema#string” and the `AttributeSelector` selects a **bag** of type
1217 “http://www.w3.org/2001/XMLSchema#string”, “urn:oasis:names:tc:xacml:1.0:function:string-one-
1218 and-only” is used. This function guarantees that its argument evaluates to a **bag** containing exactly
1219 one value.

1220 [a216] The <AttributeSelector> element selects a **bag** of values from the request **context**
1221 using a free-form XPath expression. In this case, it selects the value of the `patient-number` in
1222 the **resource**. Note that the namespace prefixes in the XPath expression are resolved with the
1223 standard XML namespace declarations.

1224 [a223] **Rule** identifier.

1225 [a224] **Rule effect** declaration. When a **rule** evaluates to ‘True’ it emits the value of the `Effect`
1226 attribute. This value is then combined with the `Effect` values of other **rules** according to the **rule-**
1227 **combining algorithm**.

1228 [a225] - [a229] Free form description of the **rule**.

1229 [a230] - [a263] A **rule target** defines a set of **decision requests** that the **rule** is intended to
1230 evaluate. In this example, the <Subjects> and <Environments> elements are omitted.

1231 [a231] - [a249] The <Resources> element contains a **disjunctive sequence** of <Resource>
1232 elements. In this example, there is just one.

1233 [a232] - [a248] The <Resource> element encloses the **conjunctive sequence** of
1234 `ResourceMatch` elements. In this example, there are two.

1235 [a233] - [a240] The first <ResourceMatch> element compares its first and second child elements
1236 according to the matching function. A match is positive if the value of the first argument matches
1237 any of the values selected by the second argument. This match compares the target namespace of
1238 the requested document with the value of “urn:example:med:schemas:record”.

1239 [a233] The `MatchId` attribute names the matching function.

1240 [a235] Literal attribute value to match.

1241 [a237] - [a239] The `<ResourceAttributeDesignator>` element selects the target namespace
 1242 from the resource contained in the request **context**. The **attribute** name is specified by the
 1243 `AttributeId`.

1244 [a241] - [a247] The second `<ResourceMatch>` element. This match compares the results of two
 1245 XPath expressions. The second XPath expression is the location path to the requested XML
 1246 element and the first XPath expression is the literal value `"/md:record"`. The `"xpath-node-match"`
 1247 function evaluates to `"True"` if the requested XML element is below the `"/md:record"` element.

1248 [a250] - [a262] The `<Actions>` element contains a **disjunctive sequence** of `<Action>` elements.
 1249 In this case, there is just one `<Action>` element.

1250 [a251] - [a261] The `<Action>` element contains a **conjunctive sequence** of `<ActionMatch>`
 1251 elements. In this case, there is just one `<ActionMatch>` element.

1252 [a252] - [a260] The `<ActionMatch>` element compares its first and second child elements
 1253 according to the matching function. The match is positive if the value of the first argument matches
 1254 any of the values selected by the second argument. In this case, the value of the `action-id`
 1255 action attribute in the request **context** is compared with the literal value `"read"`.

1256 [a264] - [a266] The `<Condition>` element. A **condition** must evaluate to `"True"` for the **rule** to be
 1257 applicable. This **condition** contains a reference to a variable definition defined elsewhere in the
 1258 **policy**.

1259 4.2.4.2. Rule 2

1260 Rule 2 illustrates the use of a mathematical function, i.e. the `<Apply>` element with `functionId`
 1261 `"urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration"` to calculate the date of the
 1262 patient's sixteenth birthday. It also illustrates the use of **predicate** expressions, with the
 1263 `functionId` `"urn:oasis:names:tc:xacml:1.0:function:and"`. This example has one function
 1264 embedded in the `<Condition>` element and another one referenced in a
 1265 `<VariableDefinition>` element.

```

1266 [a269] <?xml version="1.0" encoding="UTF-8"?>
1267 [a270] <Policy
1268 [a271] xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" xmlns:xacml-
1269 context="urn:oasis:names:tc:xacml:2.0:context:schema:os"
1270 [a272] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1271 xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
1272 http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd"
1273 [a273] xmlns:xf="http://www.w3.org/TR/2002/WD-xquery-operators-20020816/#"
1274 [a274] xmlns:md="http://www.med.example.com/schemas/record.xsd"
1275 [a275] PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:2"
1276 RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
1277 overrides">
1278 [a276] <PolicyDefaults>
1279 [a277] <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-
1280 19991116</XPathVersion>
1281 [a278] </PolicyDefaults>
1282 [a279] <Target/>
1283 [a280] <VariableDefinition VariableId="17590035">
1284 [a281] <Apply FunctionId="urn:oasis:names:tc:xacml:2.0:function:date-less-or-
1285 equal">
1286 [a282] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-
1287 only">

```

```

1288 [a283] <EnvironmentAttributeDesignator
1289 [a284] AttributeId= "urn:oasis:names:tc:xacml:1.0:environment:current-date"
1290 [a285] DataType="http://www.w3.org/2001/XMLSchema#date" />
1291 [a286] </Apply>
1292 [a287] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-
1293 yearMonthDuration">
1294 [a288] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-
1295 only">
1296 [a289] <AttributeSelector RequestContextPath=
1297 [a290] "//md:record/md:patient/md:patientDoB/text()"
1298 [a291] DataType="http://www.w3.org/2001/XMLSchema#date" />
1299 [a292] </Apply>
1300 [a293] <AttributeValue
1301 [a294] DataType="http://www.w3.org/TR/2002/WD-xquery-operators-
1302 20020816#yearMonthDuration">
1303 [a295] <xf:dt-yearMonthDuration>
1304 [a296] P16Y
1305 [a297] </xf:dt-yearMonthDuration>
1306 [a298] </AttributeValue>
1307 [a299] </Apply>
1308 [a300] </Apply>
1309 [a301] </VariableDefinition>
1310 [a302] <Rule
1311 [a303] RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:2"
1312 [a304] Effect="Permit">
1313 [a305] <Description>
1314 [a306] A person may read any medical record in the
1315 [a307] http://www.med.example.com/records.xsd namespace
1316 [a308] for which he or she is the designated parent or guardian,
1317 [a309] and for which the patient is under 16 years of age
1318 [a310] </Description>
1319 [a311] <Target>
1320 [a312] <Resources>
1321 [a313] <Resource>
1322 [a314] <ResourceMatch
1323 [a315] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1324 [a316] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1325 [a317] http://www.med.example.com/schemas/record.xsd
1326 [a318] </AttributeValue>
1327 [a319] <ResourceAttributeDesignator AttributeId=
1328 "urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1329 [a320] DataType="http://www.w3.org/2001/XMLSchema#string" />
1330 [a321] </ResourceMatch>
1331 [a322] <ResourceMatch
1332 [a323] MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1333 [a324] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1334 [a325] /md:record
1335 [a326] </AttributeValue>
1336 [a327] <ResourceAttributeDesignator
1337 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
1338 [a328] DataType="http://www.w3.org/2001/XMLSchema#string" />
1339 [a329] </ResourceMatch>
1340 [a330] </Resource>
1341 [a331] </Resources>
1342 [a332] <Actions>
1343 [a333] <Action>
1344 [a334] <ActionMatch
1345 [a335] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1346 [a336] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1347 [a337] read
1348 [a338] </AttributeValue>

```

```

1349 [a339] <ActionAttributeDesignator
1350 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1351 [a340]   DataType="http://www.w3.org/2001/XMLSchema#string"/>
1352 [a341]   </ActionMatch>
1353 [a342]   </Action>
1354 [a343]   </Actions>
1355 [a344] </Target>
1356 [a345] <Condition>
1357 [a346] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
1358 [a347]   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1359 [a348]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1360 and-only">
1361 [a349]       <SubjectAttributeDesignator
1362 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:
1363 [a350] parent-guardian-id"
1364 [a351]   DataType="http://www.w3.org/2001/XMLSchema#string"/>
1365 [a352]   </Apply>
1366 [a353]   <Apply
1367 [a354]     FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-
1368 only">
1369 [a355]     <AttributeSelector
1370 [a356]       RequestContextPath="//xacml-context:Resource/xacml-
1371 context:ResourceContent/md:record/md:parentGuardian/md:parentGuardianId/text()"
1372 [a357]       DataType="http://www.w3.org/2001/XMLSchema#string"/>
1373 [a358]     </Apply>
1374 [a359]   </Apply>
1375 [a360]   <VariableReference VariableId="17590035"/>
1376 [a361] </Apply>
1377 [a362] </Condition>
1378 [a363] </Rule>
1379 [a364] </Policy>

```

1380 [a280] - [a301] The <VariableDefinition> element contains part of the **condition** (i.e. is the
1381 patient under 16 years of age?). The patient is under 16 years of age if the current date is less than
1382 the date computed by adding 16 to the patient's date of birth.

1383 [a281] - [a300] "urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal" is used to compute the
1384 difference of two date arguments.

1385 [a282] - [a286] The first date argument uses "urn:oasis:names:tc:xacml:1.0:function:date-one-and-
1386 only" to ensure that the **bag** of values selected by its argument contains exactly one value of type
1387 "http://www.w3.org/2001/XMLSchema#date".

1388 [a284] The current date is evaluated by selecting the
1389 "urn:oasis:names:tc:xacml:1.0:environment:current-date" **environment attribute**.

1390 [a287] - [a299] The second date argument uses "urn:oasis:names:tc:xacml:1.0:function:date-add-
1391 yearMonthDuration" to compute the date of the patient's sixteenth birthday by adding 16 years to
1392 the patient's date of birth. The first of its arguments is of type
1393 "http://www.w3.org/2001/XMLSchema#date" and the second is of type
1394 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration".

1395 [a289] The <AttributeSelector> element selects the patient's date of birth by taking the XPath
1396 expression over the **resource** content.

1397 [a293] - [a298] Year Month Duration of 16 years.

1398 [a311] - [a344] **Rule** declaration and **rule target**. See Rule 1 in Section 4.2.4.1 for the detailed
1399 explanation of these elements.

1400 [a345] - [a362] The <Condition> element. The **condition** must evaluate to "True" for the **rule** to
1401 be applicable. This **condition** evaluates the truth of the statement: the requestor is the designated

1402 parent or guardian and the patient is under 16 years of age. It contains one embedded <Apply>
1403 element and one referenced <VariableDefinition> element.

1404 [a346] The **condition** uses the “urn:oasis:names:tc:xacml:1.0:function:and” function. This is a
1405 Boolean function that takes one or more Boolean arguments (2 in this case) and performs the
1406 logical “AND” operation to compute the truth value of the expression.

1407 [a347] - [a359] The first part of the **condition** is evaluated (i.e. is the requestor the designated
1408 parent or guardian?). The function is “urn:oasis:names:tc:xacml:1.0:function:string-equal” and it
1409 takes two arguments of type “http://www.w3.org/2001/XMLSchema#string”.

1410 [a348] designates the first argument. Since “urn:oasis:names:tc:xacml:1.0:function:string-equal”
1411 takes arguments of type “http://www.w3.org/2001/XMLSchema#string”,
1412 “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used to ensure that the **subject**
1413 **attribute** “urn:oasis:names:tc:xacml:2.0:example:attribute:parent-guardian-id” in the request
1414 **context** contains exactly one value.

1415 [a353] designates the second argument. The value of the **subject attribute**
1416 “urn:oasis:names:tc:xacml:2.0:example:attribute:parent-guardian-id” is selected from the request
1417 **context** using the <SubjectAttributeDesignator> element.

1418 [a354] As above, the “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used to ensure
1419 that the **bag** of values selected by it’s argument contains exactly one value of type
1420 “http://www.w3.org/2001/XMLSchema#string”.

1421 [a355] The second argument selects the value of the <md:parentGuardianId> element from the
1422 **resource** content using the <AttributeSelector> element. This element contains a free-form
1423 XPath expression, pointing into the request **context**. Note that all namespace prefixes in the XPath
1424 expression are resolved with standard namespace declarations. The AttributeSelector
1425 evaluates to the **bag** of values of type “http://www.w3.org/2001/XMLSchema#string”.

1426 [a360] references the <VariableDefinition> element, where the second part of the **condition**
1427 is defined.

1428 **4.2.4.3. Rule 3**

1429 Rule 3 illustrates the use of an **obligation**. The XACML <Rule> element syntax does not include
1430 an element suitable for carrying an **obligation**, therefore Rule 3 has to be formatted as a
1431 <Policy> element.

```
1432 [a365] <?xml version="1.0" encoding="UTF-8"?>  
1433 [a366] <Policy  
1434 [a367]   xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" xmlns:xacml-  
1435 context="urn:oasis:names:tc:xacml:2.0:context:schema:os"  
1436 [a368]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
1437 [a369]   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os  
1438 http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd"  
1439 [a370]   xmlns:md="http://www.med.example.com/schemas/record.xsd"  
1440 [a371]   PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:3"  
1441 [a372]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-  
1442 algorithm:deny-overrides">  
1443 [a373]   <Description>  
1444 [a374]     Policy for any medical record in the  
1445 [a375]     http://www.med.example.com/schemas/record.xsd namespace  
1446 [a376]   </Description>  
1447 [a377]   <PolicyDefaults>  
1448 [a378]     <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-  
1449 19991116</XPathVersion>  
1450 [a379]   </PolicyDefaults>
```

```

1451 [a380] <Target>
1452 [a381]   <Resources>
1453 [a382]     <Resource>
1454 [a383]       <ResourceMatch
1455 [a384]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1456 [a385]           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1457 [a386]             urn:example:med:schemas:record
1458 [a387]           </AttributeValue>
1459 [a388]           <ResourceAttributeDesignator AttributeId=
1460 [a389]             "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1461 [a390]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1462 [a391]           </ResourceMatch>
1463 [a392]         </Resource>
1464 [a393]     </Resources>
1465 [a394] </Target>
1466 [a395] <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:3"
1467 [a396] Effect="Permit">
1468 [a397]   <Description>
1469 [a398]     A physician may write any medical element in a record
1470 [a399]     for which he or she is the designated primary care
1471 [a400]     physician, provided an email is sent to the patient
1472 [a401]   </Description>
1473 [a402] <Target>
1474 [a403]   <Subjects>
1475 [a404]     <Subject>
1476 [a405]       <SubjectMatch
1477 [a406]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1478 [a407]           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1479 [a408]             physician
1480 [a409]           </AttributeValue>
1481 [a410]           <SubjectAttributeDesignator AttributeId=
1482 [a411]             "urn:oasis:names:tc:xacml:2.0:example:attribute:role"
1483 [a412]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1484 [a413]           </SubjectMatch>
1485 [a414]         </Subject>
1486 [a415]       </Subjects>
1487 [a416]     <Resources>
1488 [a417]       <Resource>
1489 [a418]         <ResourceMatch
1490 [a419]           MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1491 [a420]             <AttributeValue
1492 [a421]               DataType="http://www.w3.org/2001/XMLSchema#string">
1493 [a422]                 /md:record/md:medical
1494 [a423]             </AttributeValue>
1495 [a424]             <ResourceAttributeDesignator
1496 [a425]               AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
1497 [a426]               DataType="http://www.w3.org/2001/XMLSchema#string"/>
1498 [a427]             </ResourceMatch>
1499 [a428]           </Resource>
1500 [a429]         </Resources>
1501 [a430]     <Actions>
1502 [a431]       <Action>
1503 [a432]         <ActionMatch
1504 [a433]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1505 [a434]             <AttributeValue
1506 [a435]               DataType="http://www.w3.org/2001/XMLSchema#string">
1507 [a436]                 write
1508 [a437]             </AttributeValue>
1509 [a438]             <ActionAttributeDesignator
1510 [a439]               AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1511 [a440]               DataType="http://www.w3.org/2001/XMLSchema#string"/>
1512 [a441]             </ActionMatch>

```

```

1513 [a439] </Action>
1514 [a440] </Actions>
1515 [a441] </Target>
1516 [a442] <Condition>
1517 [a443] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1518 [a444] <Apply
1519 [a445] FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1520 [a446] <SubjectAttributeDesignator
1521 [a447] AttributeId="urn:oasis:names:tc:xacml:2.0:example:
1522 attribute:physician-id"
1523 [a448] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1524 [a449] </Apply>
1525 [a450] <Apply
1526 [a451] FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1527 [a452] <AttributeSelector RequestContextPath=
1528 [a453] "//xacml-context:Resource/xacml-
1529 context:ResourceContent/md:record/md:primaryCarePhysician/md:registrationID/text(
1530 )"
1531 [a454] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1532 [a455] </Apply>
1533 [a456] </Apply>
1534 [a457] </Condition>
1535 [a458] </Rule>
1536 [a459] <Obligations>
1537 [a460] <Obligation
1538 ObligationId="urn:oasis:names:tc:xacml:example:obligation:email"
1539 [a461] FulfillOn="Permit">
1540 [a462] <AttributeAssignment
1541 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:mailto"
1542 [a463] DataType="http://www.w3.org/2001/XMLSchema#string">
1543 [a464] <AttributeSelector RequestContextPath=
1544 [a465] "//md:/record/md:patient/md:patientContact/md:email"
1545 [a466] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1546 [a467] </AttributeAssignment>
1547 [a468] <AttributeAssignment
1548 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
1549 [a469] DataType="http://www.w3.org/2001/XMLSchema#string">
1550 [a470] Your medical record has been accessed by:
1551 [a471] </AttributeAssignment>
1552 [a472] <AttributeAssignment
1553 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
1554 [a473] DataType="http://www.w3.org/2001/XMLSchema#string">
1555 [a474] <SubjectAttributeDesignator
1556 AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
1557 [a475] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1558 [a476] </AttributeAssignment>
1559 [a477] </Obligation>
1560 [a478] </Obligations>
1561 [a479] </Policy>

```

1562 [a366] - [a372] The <Policy> element includes standard namespace declarations as well as policy
1563 specific parameters, such as PolicyId and RuleCombiningAlgId.

1564 [a371] **Policy** identifier. This parameter allows the **policy** to be referenced by a **policy set**.

1565 [a372] The **Rule combining algorithm** identifies the algorithm for combining the outcomes of **rule**
1566 evaluation.

1567 [a373] - [a376] Free-form description of the **policy**.

1568 [a379] - [a394] **Policy target**. The **policy target** defines a set of applicable decision requests. The
1569 structure of the <Target> element in the <Policy> is identical to the structure of the <Target>

1570 element in the <Rule>. In this case, the **policy target** is the set of all XML resources that conform
1571 to the namespace "urn:example:med:schemas:record".

1572 [a395] The only <Rule> element included in this <Policy>. Two parameters are specified in the
1573 **rule** header: RuleId and Effect.

1574 [a402] - [a441] The **rule target** further constrains the **policy target**.

1575 [a405] - [a412] The <SubjectMatch> element targets the **rule** at **subjects** whose
1576 "urn:oasis:names:tc:xacml:2.0:example:attribute:role" **subject attribute** is equal to "physician".

1577 [a417] - [a425] The <ResourceMatch> element targets the **rule** at **resources** that match the
1578 XPath expression "/md:record/md:medical".

1579 [a430] - [a438] The <ActionMatch> element targets the **rule** at **actions** whose
1580 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** is equal to "write".

1581 [a442] - [a457] The <Condition> element. For the **rule** to be applicable to the **decision request**,
1582 the **condition** must evaluate to "True". This **condition** compares the value of the
1583 "urn:oasis:names:tc:xacml:2.0:example:attribute:physician-id" **subject attribute** with the value of
1584 the <registrationId> element in the medical record that is being accessed.

1585 [a459] - [a478] The <Obligations> element. **Obligations** are a set of operations that must be
1586 performed by the **PEP** in conjunction with an **authorization decision**. An **obligation** may be
1587 associated with a "Permit" or "Deny" **authorization decision**. The element contains a single
1588 **obligation**.

1589 [a460] - [a477] The <Obligation> element consists of the ObligationId attribute, the
1590 **authorization decision** value for which it must be fulfilled, and a set of **attribute** assignments. The
1591 **PDP** does not resolve the attribute assignments. This is the job of the **PEP**.

1592 [a460] The ObligationId attribute identifies the **obligation**. In this case, the **PEP** is required to
1593 send email.

1594 [a461] The FulfillOn attribute defines the **authorization decision** value for which this
1595 **obligation** must be fulfilled. In this case, when access is permitted.

1596 [a462] - [a467] The first parameter indicates where the **PEP** will find the email address in the
1597 resource.

1598 [a468] - [a471] The second parameter contains literal text for the email body.

1599 [a472] - [a476] The third parameter indicates where the **PEP** will find further text for the email body
1600 in the resource.

1601 **4.2.4.4. Rule 4**

1602 Rule 4 illustrates the use of the "Deny" Effect value, and a <Rule> with no <Condition>
1603 element.

```

1604 [a480] <?xml version="1.0" encoding="UTF-8"?>
1605 [a481] <Policy
1606 [a482] xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
1607 [a483] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1608 xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
1609 http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd"
1610 [a484] xmlns:md="http://www.med.example.com/schemas/record.xsd"
1611 [a485] PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:4"

```

```

1612 [a486] RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1613 algorithm:deny-overrides">
1614 [a487] <PolicyDefaults>
1615 [a488] <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-
1616 19991116</XPathVersion>
1617 [a489] </PolicyDefaults>
1618 [a490] <Target/>
1619 [a491] <Rule
1620 [a492] RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:4"
1621 [a493] Effect="Deny">
1622 [a494] <Description>
1623 [a495] An Administrator shall not be permitted to read or write
1624 [a496] medical elements of a patient record in the
1625 [a497] http://www.med.example.com/records.xsd namespace.
1626 [a498] </Description>
1627 [a499] <Target>
1628 [a500] <Subjects>
1629 [a501] <Subject>
1630 [a502] <SubjectMatch
1631 [a503] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1632 [a504] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1633 [a505] administrator
1634 [a506] </AttributeValue>
1635 [a507] <SubjectAttributeDesignator AttributeId=
1636 [a508] "urn:oasis:names:tc:xacml:2.0:example:attribute:role"
1637 [a509] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1638 [a510] </SubjectMatch>
1639 [a511] </Subject>
1640 [a512] </Subjects>
1641 [a513] <Resources>
1642 [a514] <Resource>
1643 [a515] <ResourceMatch
1644 [a516] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1645 [a517] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1646 [a518] urn:example:med:schemas:record
1647 [a519] </AttributeValue>
1648 [a520] <ResourceAttributeDesignator
1649 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1650 [a521] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1651 [a522] </ResourceMatch>
1652 [a523] <ResourceMatch
1653 [a524] MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1654 [a525] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1655 [a526] /md:record/md:medical
1656 [a527] </AttributeValue>
1657 [a528] <ResourceAttributeDesignator
1658 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
1659 [a529] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1660 [a530] </ResourceMatch>
1661 [a531] </Resource>
1662 [a532] </Resources>
1663 [a533] <Actions>
1664 [a534] <Action>
1665 [a535] <ActionMatch
1666 [a536] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1667 [a537] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1668 [a538] read
1669 [a539] </AttributeValue>
1670 [a540] <ActionAttributeDesignator
1671 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1672 [a541] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1673 [a542] </ActionMatch>

```



```

1674 [a543] </Action>
1675 [a544] <Action>
1676 [a545] <ActionMatch
1677 [a546] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1678 [a547] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1679 [a548] write
1680 [a549] </AttributeValue>
1681 [a550] <ActionAttributeDesignator
1682 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1683 [a551] DataType="http://www.w3.org/2001/XMLSchema#string" />
1684 [a552] </ActionMatch>
1685 [a553] </Action>
1686 [a554] </Actions>
1687 [a555] </Target>
1688 [a556] </Rule>
1689 [a557] </Policy>

```

1690 [a492] - [a493] The <Rule> element declaration.

1691 [a493] **Rule Effect**. Every **rule** that evaluates to “True” emits the **rule effect** as its value. This
1692 **rule Effect** is “Deny” meaning that according to this **rule**, access must be denied when it
1693 evaluates to “True”.

1694 [a494] - [a498] Free form description of the **rule**.

1695 [a499] - [a555] **Rule target**. The **Rule target** defines the set of **decision requests** that are
1696 applicable to the **rule**.

1697 [a502] - [a510] The <SubjectMatch> element targets the **rule** at **subjects** whose
1698 “urn:oasis:names:tc:xacml:2.0:example:attribute:role” **subject attribute** is equal to
1699 “administrator”.

1700 [a513] - [a532] The <Resources> element contains one <Resource> element, which (in turn)
1701 contains two <ResourceMatch> elements. The **target** matches if the **resource** identified by the
1702 request **context** matches both **resource** match criteria.

1703 [a515]-[a522] The first <ResourceMatch> element targets the **rule** at **resources**
1704 whose “urn:oasis:names:tc:xacml:2.0:resource:target-namespace” **resource attribute**
1705 is equal to “urn:example:med:schemas:record”.

1706 [a523] - [a530] The second <ResourceMatch> element targets the **rule** at XML elements that
1707 match the XPath expression “/md:record/md:medical”.

1708 [a533] - [a554] The <Actions> element contains two <Action> elements, each of which contains
1709 one <ActionMatch> element. The **target** matches if the **action** identified in the request **context**
1710 matches either of the **action** match criteria.

1711 [a535] - [a552] The <ActionMatch> elements target the **rule** at **actions** whose
1712 “urn:oasis:names:tc:xacml:1.0:action:action-id” **action attribute** is equal to “read” or “write”.

1713 This **rule** does not have a <Condition> element.

1714 4.2.4.5. Example PolicySet

1715 This section uses the examples of the previous sections to illustrate the process of combining
1716 **policies**. The policy governing read access to medical elements of a record is formed from each of
1717 the four **rules** described in Section 4.2.3. In plain language, the combined rule is:

- 1718 • Either the requestor is the patient; or
- 1719 • the requestor is the parent or guardian and the patient is under 16; or

- 1720 • the requestor is the primary care physician and a notification is sent to the patient; and
- 1721 • the requestor is not an administrator.

1722 The following **policy set** illustrates the combined **policies**. **Policy 3** is included by reference and
 1723 **policy 2** is explicitly included.

```

1724 [a558] <?xml version="1.0" encoding="UTF-8"?>
1725 [a559] <PolicySet
1726 [a560]   xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
1727 [a561]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1728 [a562]   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
1729 [a563]   http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd"
1730 [a564]   PolicySetId=
1731 [a565]   "urn:oasis:names:tc:xacml:2.0:example:policysetid:1"
1732 [a566]   PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
1733 [a567]   policy-combining-algorithm:deny-overrides">
1734 [a568]   <Description>
1735 [a569]     Example policy set.
1736 [a570]   </Description>
1737 [a571]   <Target>
1738 [a572]     <Resources>
1739 [a573]       <Resource>
1740 [a574]         <ResourceMatch
1741 [a575]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1742 [a576]             <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1743 [a577]               urn:example:med:schema:records
1744 [a578]             </AttributeValue>
1745 [a579]           <ResourceAttributeDesignator AttributeId=
1746 [a580]             "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1747 [a581]             DataType="http://www.w3.org/2001/XMLSchema#string" />
1748 [a582]           </ResourceMatch>
1749 [a583]         </Resource>
1750 [a584]       </Resources>
1751 [a585]     </Target>
1752 [a586]   <PolicyIdReference>
1753 [a587]     urn:oasis:names:tc:xacml:2.0:example:policyid:3
1754 [a588]   </PolicyIdReference>
1755 [a589] <Policy
1756 [a590]   PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:2"
1757 [a591]   RuleCombiningAlgId=
1758 [a592]   "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
1759 [a593]   <Target/>
1760 [a594]   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:1"
1761 [a595]     Effect="Permit">
1762 [a596]   </Rule>
1763 [a597]   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:2"
1764 [a598]     Effect="Permit">
1765 [a599]   </Rule>
1766 [a600]   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:4"
1767 [a601]     Effect="Deny">
1768 [a602]   </Rule>
1769 [a603] </Policy>
1770 [a604] </PolicySet>
1771

```

1772 [a559] - [a565] The <PolicySet> element declaration. Standard XML namespace declarations
 1773 are included.

1774 [a562] The PolicySetId attribute is used for identifying this **policy set** for possible inclusion in
 1775 another **policy set**.

1776 [a564] The **policy combining algorithm** identifier. **Policies** and **policy sets** in this **policy set** are
1777 combined according to the specified **policy combining algorithm** when the **authorization**
1778 **decision** is computed.

1779 [a566] - [a568] Free form description of the **policy set**.

1780 [a569] - [a583] The **policy set** <Target> element defines the set of **decision requests** that are
1781 applicable to this <PolicySet> element.

1782 [a584] PolicyIdReference includes a **policy** by id.

1783 [a588] Policy 2 is explicitly included in this **policy set**. The **rules** in Policy 2 are omitted for
1784 clarity.

1785 5. Policy syntax (normative, with the exception of 1786 the schema fragments)

1787 5.1. Element <PolicySet>

1788 The <PolicySet> element is a top-level element in the XACML policy schema. <PolicySet> is
1789 an aggregation of other **policy sets** and **policies**. **Policy sets** MAY be included in an enclosing
1790 <PolicySet> element either directly using the <PolicySet> element or indirectly using the
1791 <PolicySetIdReference> element. **Policies** MAY be included in an enclosing <PolicySet>
1792 element either directly using the <Policy> element or indirectly using the
1793 <PolicyIdReference> element.

1794 A <PolicySet> element MAY be evaluated, in which case the evaluation procedure defined in
1795 Section 7.11 SHALL be used.

1796 If a <PolicySet> element contains references to other **policy sets** or **policies** in the form of
1797 URLs, then these references MAY be resolvable.

1798 **Policy sets** and **policies** included in a <PolicySet> element MUST be combined using the
1799 algorithm identified by the PolicyCombiningAlgId attribute. <PolicySet> is treated exactly
1800 like a <Policy> in all **policy combining algorithms**.

1801 The <Target> element defines the applicability of the <PolicySet> element to a set of **decision**
1802 **requests**. If the <Target> element within the <PolicySet> element matches the **request**
1803 **context**, then the <PolicySet> element MAY be used by the **PDP** in making its **authorization**
1804 **decision**. See Section 7.11.

1805 The <Obligations> element contains a set of **obligations** that MUST be fulfilled by the **PEP** in
1806 conjunction with the **authorization decision**. If the **PEP** does not understand, or cannot fulfill, any
1807 of the **obligations**, then it MUST act as if the **PDP** had returned a "Deny" **authorization decision**
1808 value. See Section 7.14.

```
1809 <xs:element name="PolicySet" type="xacml:PolicySetType"/>
1810 <xs:complexType name="PolicySetType">
1811   <xs:sequence>
1812     <xs:element ref="xacml:Description" minOccurs="0"/>
1813     <xs:element ref="xacml:PolicySetDefaults" minOccurs="0"/>
1814     <xs:element ref="xacml:Target"/>
1815     <xs:choice minOccurs="0" maxOccurs="unbounded">
1816       <xs:element ref="xacml:PolicySet"/>
```

```

1817     <xs:element ref="xacml:Policy"/>
1818     <xs:element ref="xacml:PolicySetIdReference"/>
1819     <xs:element ref="xacml:PolicyIdReference"/>
1820     <xs:element ref="xacml:CombinerParameters"/>
1821     <xs:element ref="xacml:PolicyCombinerParameters"/>
1822     <xs:element ref="xacml:PolicySetCombinerParameters"/>
1823   </xs:choice>
1824   <xs:element ref="xacml:Obligations" minOccurs="0"/>
1825 </xs:sequence>
1826 <xs:attribute name="PolicySetId" type="xs:anyURI" use="required"/>
1827 <xs:attribute name="Version" type="xacml:VersionType" default="1.0"/>
1828 <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI" use="required"/>
1829 </xs:complexType>

```

1830 The <PolicySet> element is of **PolicySetType** complex type.

1831 The <PolicySet> element contains the following attributes and elements:

1832 PolicySetId [Required]

1833 **Policy set** identifier. It is the responsibility of the **PAP** to ensure that no two **policies**
1834 visible to the **PDP** have the same identifier. This MAY be achieved by following a
1835 predefined URN or URI scheme. If the **policy set** identifier is in the form of a URL, then it
1836 MAY be resolvable.

1837 Version [Default 1.0]

1838 The version number of the **PolicySet**.

1839 PolicyCombiningAlgId [Required]

1840 The identifier of the **policy-combining algorithm** by which the <PolicySet>, <CombinerParameters>, <PolicyCombinerParameters> and <PolicySetCombinerParameters> components MUST be combined. Standard **policy-combining algorithms** are listed in Appendix C. Standard **policy-combining algorithm** identifiers are listed in Section B.10.

1845 <Description> [Optional]

1846 A free-form description of the **policy set**.

1847 <PolicySetDefaults> [Optional]

1848 A set of default values applicable to the **policy set**. The scope of the <PolicySetDefaults> element SHALL be the enclosing **policy set**.

1850 <Target> [Required]

1851 The <Target> element defines the applicability of a **policy set** to a set of **decision requests**.

1853 The <Target> element MAY be declared by the creator of the <PolicySet> or it MAY be computed from the <Target> elements of the referenced <Policy> elements, either as an intersection or as a union.

1856 <PolicySet> [Any Number]

1857 A **policy set** that is included in this **policy set**.

1858 <Policy> [Any Number]

1859 A **policy** that is included in this **policy set**.

1860 <PolicySetIdReference> [Any Number]

1861 A reference to a **policy set**. that MUST be included in this **policy set**. If
 1862 <PolicySetIdReference> is a URL, then it MAY be resolvable.

1863 <PolicyIdReference> [Any Number]

1864 A reference to a **policy** that MUST be included in this **policy set**. If the
 1865 <PolicyIdReference> is a URL, then it MAY be resolvable.

1866 <Obligations> [Optional]

1867 Contains the set of <Obligation> elements. See Section 7.14 for a description of how
 1868 the set of **obligations** to be returned by the **PDP** shall be determined.

1869 <CombinerParameters> [Optional]

1870 Contains a sequence of <CombinerParameter> elements.

1871 <PolicyCombinerParameters> [Optional]

1872 Contains a sequence of <CombinerParameter> elements that are associated with a
 1873 particular <Policy> or <PolicyIdReference> element within the <PolicySet>.

1874 <PolicySetCombinerParameters> [Optional]

1875 Contains a sequence of <CombinerParameter> elements that are associated with a
 1876 particular <PolicySet> or <PolicySetIdReference> element within the
 1877 <PolicySet>.

1878 5.2. Element <Description>

1879 The <Description> element contains a free-form description of the <PolicySet>, <Policy>
 1880 or <Rule> element. The <Description> element is of **xs:string** simple type.

```
1881 <xs:element name="Description" type="xs:string"/>
```

1882 5.3. Element <PolicySetDefaults>

1883 The <PolicySetDefaults> element SHALL specify default values that apply to the
 1884 <PolicySet> element.

```
1885 <xs:element name="PolicySetDefaults" type="xacml:DefaultsType"/>
1886 <xs:complexType name="DefaultsType">
1887   <xs:sequence>
1888     <xs:choice>
1889       <xs:element ref="xacml:XPathVersion" minOccurs="0"/>
1890     </xs:choice>
1891   </xs:sequence>
1892 </xs:complexType>
```

1893 <PolicySetDefaults> element is of **DefaultsType** complex type.

1894 The <PolicySetDefaults> element contains the following elements:

1895 <XPathVersion> [Optional]

1896 Default XPath version.

1897

5.4. Element <XPathVersion>

1898 The <XPathVersion> element SHALL specify the version of the XPath specification to be used by
1899 <AttributeSelector> elements and XPath-based functions in the **policy set** or **policy**.

```
1900 <xs:element name="XPathVersion" type="xs:anyURI" />
```

1901 The URI for the XPath 1.0 specification is "http://www.w3.org/TR/1999/Rec-xpath-19991116". The
1902 <XPathVersion> element is REQUIRED if the XACML enclosing **policy set** or **policy** contains
1903 <AttributeSelector> elements or XPath-based functions.

1904

5.5. Element <Target>

1905 The <Target> element identifies the set of **decision requests** that the parent element is intended
1906 to evaluate. The <Target> element SHALL appear as a child of a <PolicySet> and <Policy>
1907 element and MAY appear as a child of a <Rule> element. It contains definitions for **subjects**,
1908 **resources**, **actions** and **environments**.

1909 The <Target> element SHALL contain a **conjunctive sequence** of <Subjects>, <Resources>
1910 <Actions> and <Environments> elements. For the parent of the <Target> element to be
1911 applicable to the **decision request**, there MUST be at least one positive match between each
1912 section of the <Target> element and the corresponding section of the <xacml-
1913 context:Request> element.

```
1914 <xs:element name="Target" type="xacml:TargetType" />  
1915 <xs:complexType name="TargetType" >  
1916   <xs:sequence>  
1917     <xs:element ref="xacml:Subjects" minOccurs="0" />  
1918     <xs:element ref="xacml:Resources" minOccurs="0" />  
1919     <xs:element ref="xacml:Actions" minOccurs="0" />  
1920     <xs:element ref="xacml:Environments" minOccurs="0" />  
1921   </xs:sequence>  
1922 </xs:complexType>
```

1923 The <Target> element is of **TargetType** complex type.

1924 The <Target> element contains the following elements:

1925 <Subjects> [Optional]

1926 Matching specification for the **subject attributes** in the **context**. If this element is missing,
1927 then the **target** SHALL match all **subjects**.

1928 <Resources> [Optional]

1929 Matching specification for the **resource attributes** in the **context**. If this element is
1930 missing, then the **target** SHALL match all **resources**.

1931 <Actions> [Optional]

1932 Matching specification for the **action attributes** in the **context**. If this element is missing,
1933 then the **target** SHALL match all **actions**.

1934 <Environments> [Optional]

1935 Matching specification for the **environment attributes** in the **context**. If this element is
1936 missing, then the **target** SHALL match all **environments**.

1937

5.6. Element <Subjects>

1938 The <Subjects> element SHALL contain a **disjunctive sequence** of <Subject> elements.

```
1939 <xs:element name="Subjects" type="xacml:SubjectsType"/>
1940 <xs:complexType name="SubjectsType">
1941   <xs:sequence>
1942     <xs:element ref="xacml:Subject" maxOccurs="unbounded"/>
1943   </xs:sequence>
1944 </xs:complexType>
```

1945 The <Subjects> element is of **SubjectsType** complex type.

1946 The <Subjects> element contains the following elements:

1947 <Subject> [One to Many, Required]

1948 See Section 5.7.

5.7. Element <Subject>

1950 The <Subject> element SHALL contain a **conjunctive sequence** of <SubjectMatch> elements.

```
1952 <xs:element name="Subject" type="xacml:SubjectType"/>
1953 <xs:complexType name="SubjectType">
1954   <xs:sequence>
1955     <xs:element ref="xacml:SubjectMatch" maxOccurs="unbounded"/>
1956   </xs:sequence>
1957 </xs:complexType>
```

1958 The <Subject> element is of **SubjectType** complex type.

1959 The <Subject> element contains the following elements:

1960 <SubjectMatch> [One to Many]

1961 A **conjunctive sequence** of individual matches of the **subject attributes** in the request
1962 **context** and the embedded **attribute** values. See Section 5.8.

5.8. Element <SubjectMatch>

1964 The <SubjectMatch> element SHALL identify a set of **subject**-related entities by matching
1965 **attribute** values in a <xacml-context:Subject> element of the request **context** with the
1966 embedded **attribute** value.

```
1967 <xs:element name="SubjectMatch" type="xacml:SubjectMatchType"/>
1968 <xs:complexType name="SubjectMatchType">
1969   <xs:sequence>
1970     <xs:element ref="xacml:AttributeValue"/>
1971     <xs:choice>
1972       <xs:element ref="xacml:SubjectAttributeDesignator"/>
1973       <xs:element ref="xacml:AttributeSelector"/>
1974     </xs:choice>
1975   </xs:sequence>
1976   <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
1977 </xs:complexType>
```

1978 The <SubjectMatch> element is of **SubjectMatchType** complex type.

1979 The <SubjectMatch> element contains the following attributes and elements:

- 1980 MatchId [Required]
- 1981 Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI** with
1982 legal values documented in Section 7.5.
- 1983 <xacml:AttributeValue> [Required]
- 1984 Embedded attribute value.
- 1985 <SubjectAttributeDesignator> [Required choice]
- 1986 MAY be used to identify one or more **attribute** values in a <Subject> element of the
1987 request **context**.
- 1988 <AttributeSelector> [Required choice]
- 1989 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
1990 expression SHOULD resolve to an **attribute** in a <Subject> element of the request
1991 **context**.

1992 **5.9. Element <Resources>**

1993 The <Resources> element SHALL contain a **disjunctive sequence** of <Resource> elements.

```
1994 <xs:element name="Resources" type="xacml:ResourcesType" />
1995 <xs:complexType name="ResourcesType">
1996   <xs:sequence>
1997     <xs:element ref="xacml:Resource" maxOccurs="unbounded" />
1998   </xs:sequence>
1999 </xs:complexType>
```

2000 The <Resources> element is of **ResourcesType** complex type.

2001 The <Resources> element contains the following elements:

2002 <Resource> [One to Many, Required]

2003 See Section 5.10.

2004 **5.10. Element <Resource>**

2005 The <Resource> element SHALL contain a **conjunctive sequence** of <ResourceMatch>
2006 elements.

```
2007 <xs:element name="Resource" type="xacml:ResourceType" />
2008 <xs:complexType name="ResourceType">
2009   <xs:sequence>
2010     <xs:element ref="xacml:ResourceMatch" maxOccurs="unbounded" />
2011   </xs:sequence>
2012 </xs:complexType>
```

2013 The <Resource> element is of **ResourceType** complex type.

2014 The <Resource> element contains the following elements:

2015 <ResourceMatch> [One to Many]

2016 A **conjunctive sequence** of individual matches of the **resource attributes** in the request
2017 **context** and the embedded **attribute** values. See Section 5.11.

2018

5.11. Element <ResourceMatch>

2019 The <ResourceMatch> element SHALL identify a set of **resource**-related entities by matching
2020 **attribute** values in the <xacml-context:Resource> element of the request **context** with the
2021 embedded **attribute** value.

```
2022 <xs:element name="ResourceMatch" type="xacml:ResourceMatchType" />  
2023 <xs:complexType name="ResourceMatchType">  
2024   <xs:sequence>  
2025     <xs:element ref="xacml:AttributeValue" />  
2026     <xs:choice>  
2027       <xs:element ref="xacml:ResourceAttributeDesignator" />  
2028       <xs:element ref="xacml:AttributeSelector" />  
2029     </xs:choice>  
2030   </xs:sequence>  
2031   <xs:attribute name="MatchId" type="xs:anyURI" use="required" />  
2032 </xs:complexType>
```

2033 The <ResourceMatch> element is of **ResourceMatchType** complex type.

2034 The <ResourceMatch> element contains the following attributes and elements:

2035 MatchId [Required]

2036 Specifies a matching function. Values of this attribute MUST be of type **xs:anyURI**, with
2037 legal values documented in Section 7.5.

2038 <xacml:AttributeValue> [Required]

2039 Embedded attribute value.

2040 <ResourceAttributeDesignator> [Required Choice]

2041 MAY be used to identify one or more **attribute** values in the <Resource> element of the
2042 request **context**.

2043 <AttributeSelector> [Required Choice]

2044 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
2045 expression SHOULD resolve to an **attribute** in the <Resource> element of the request
2046 **context**.

2047 5.12. Element <Actions>

2048 The <Actions> element SHALL contain a **disjunctive sequence** of <Action> elements.

```
2049 <xs:element name="Actions" type="xacml:ActionsType" />  
2050 <xs:complexType name="ActionsType">  
2051   <xs:sequence>  
2052     <xs:element ref="xacml:Action" maxOccurs="unbounded" />  
2053   </xs:sequence>  
2054 </xs:complexType>
```

2055 The <Actions> element is of **ActionsType** complex type.

2056 The <Actions> element contains the following elements:

2057 <Action> [One to Many, Required]

2058 See Section 5.13.

2059

5.13. Element <Action>

2060 The <Action> element SHALL contain a **conjunctive sequence** of <ActionMatch> elements.

```
2061 <xs:element name="Action" type="xacml:ActionType"/>
2062 <xs:complexType name="ActionType">
2063   <xs:sequence>
2064     <xs:element ref="xacml:ActionMatch" maxOccurs="unbounded"/>
2065   </xs:sequence>
2066 </xs:complexType>
```

2067 The <Action> element is of **ActionType** complex type.

2068 The <Action> element contains the following elements:

2069 <ActionMatch> [One to Many]

2070 A **conjunctive sequence** of individual matches of the **action attributes** in the request
2071 **context** and the embedded **attribute** values. See Section 5.14.

2072

5.14. Element <ActionMatch>

2073 The <ActionMatch> element SHALL identify a set of **action**-related entities by matching **attribute**
2074 values in the <xacml-context:Action> element of the request **context** with the embedded
2075 **attribute** value.

```
2076 <xs:element name="ActionMatch" type="xacml:ActionMatchType"/>
2077 <xs:complexType name="ActionMatchType">
2078   <xs:sequence>
2079     <xs:element ref="xacml:AttributeValue"/>
2080     <xs:choice>
2081       <xs:element ref="xacml:ActionAttributeDesignator"/>
2082       <xs:element ref="xacml:AttributeSelector"/>
2083     </xs:choice>
2084   </xs:sequence>
2085   <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
2086 </xs:complexType>
```

2087 The <ActionMatch> element is of **ActionMatchType** complex type.

2088 The <ActionMatch> element contains the following attributes and elements:

2089 MatchId [Required]

2090 Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI**, with
2091 legal values documented in Section 7.5.

2092 <xacml:AttributeValue> [Required]

2093 Embedded attribute value.

2094 <ActionAttributeDesignator> [Required Choice]

2095 MAY be used to identify one or more **attribute** values in the <Action> element of the
2096 request **context**.

2097 <AttributeSelector> [Required Choice]

2098 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
2099 expression SHOULD resolve to an **attribute** in the <Action> element of the **context**.

2100 **5.15. Element <Environments>**

2101 The <Environments> element SHALL contain a **disjunctive sequence** of <Environment>
2102 elements.

```
2103 <xs:element name="Environments" type="xacml:EnvironmentsType"/>
2104 <xs:complexType name="EnvironmentsType">
2105   <xs:sequence>
2106     <xs:element ref="xacml:Environment" maxOccurs="unbounded"/>
2107   </xs:sequence>
2108 </xs:complexType>
```

2109 The <Environments> element is of **EnvironmentsType** complex type.

2110 The <Environments> element contains the following elements:

2111 <Environment> [One to Many, Required]

2112 See Section 5.16.

2113 **5.16. Element <Environment>**

2114 The <Environment> element SHALL contain a **conjunctive sequence** of
2115 <EnvironmentMatch> elements.

```
2116 <xs:element name="Environment" type="xacml:EnvironmentType"/>
2117 <xs:complexType name="EnvironmentType">
2118   <xs:sequence>
2119     <xs:element ref="xacml:EnvironmentMatch" maxOccurs="unbounded"/>
2120   </xs:sequence>
2121 </xs:complexType>
```

2122 The <Environment> element is of **EnvironmentType** complex type.

2123 The <Environment> element contains the following elements:

2124 <EnvironmentMatch> [One to Many]

2125 A **conjunctive sequence** of individual matches of the **environment** attributes in the
2126 request **context** and the embedded **attribute** values.

2127 **5.17. Element <EnvironmentMatch>**

2128 The <EnvironmentMatch> element SHALL identify an environment by matching **attribute** values
2129 in the <xacml-context:Environment> element of the request **context** with the embedded
2130 **attribute** value.

```
2131 <xs:element name="EnvironmentMatch" type="xacml:EnvironmentMatchType"/>
2132 <xs:complexType name="EnvironmentMatchType">
2133   <xs:sequence>
2134     <xs:element ref="xacml:AttributeValue"/>
2135     <xs:choice>
2136       <xs:element ref="xacml:EnvironmentAttributeDesignator"/>
2137       <xs:element ref="xacml:AttributeSelector"/>
2138     </xs:choice>
2139   </xs:sequence>
2140   <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
2141 </xs:complexType>
```

2142 The <EnvironmentMatch> element is of **EnvironmentMatchType** complex type.

2143 The <EnvironmentMatch> element contains the following attributes and elements:

2144 MatchId [Required]

2145 Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI**, with
2146 legal values documented in Section A.3.

2147 <xacml:AttributeValue> [Required]

2148 Embedded attribute value.

2149 <EnvironmentAttributeDesignator> [Required Choice]

2150 MAY be used to identify one or more **attribute** values in the <Environment> element of
2151 the request **context**.

2152 <AttributeSelector> [Required Choice]

2153 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
2154 expression SHOULD resolve to an **attribute** in the <Environment> element of the
2155 request **context**.

2156 **5.18. Element <PolicySetIdReference>**

2157 The <PolicySetIdReference> element SHALL be used to reference a <PolicySet> element
2158 by id. If <PolicySetIdReference> is a URL, then it MAY be resolvable to the <PolicySet>
2159 element. However, the mechanism for resolving a **policy set** reference to the corresponding
2160 **policy set** is outside the scope of this specification.

```
2161 <xs:element name="PolicySetIdReference" type="xacml:IdReferenceType" />
2162 xs:complexType name="IdReferenceType">
2163   <xs:simpleContent>
2164     <xs:extension base="xs:anyURI">
2165       <xs:attribute name="xacml:Version" type="xacml:VersionMatchType"
2166 use="optional" />
2167       <xs:attribute name="xacml:EarliestVersion" type="xacml:VersionMatchType"
2168 use="optional" />
2169       <xs:attribute name="xacml:LatestVersion" type="xacml:VersionMatchType"
2170 use="optional" />
2171     </xs:extension>
2172   </xs:simpleContent>
2173 </xs:complexType>
```

2174 Element <PolicySetIdReference> is of **xacml:IdReferenceType** complex type.

2175 **IdReferenceType** extends the **xs:anyURI** type with the following attributes:

2176 Version [Optional]

2177 Specifies a matching expression for the version of the **policy set** referenced.

2178 EarliestVersion [Optional]

2179 Specifies a matching expression for the earliest acceptable version of the **policy set**
2180 referenced.

2181 LatestVersion [Optional]

2182 Specifies a matching expression for the latest acceptable version of the **policy set**
2183 referenced.

2184 The matching operation is defined in Section 5.21. Any combination of these attributes MAY be
2185 present in a <PolicySetIdReference>. The referenced **policy set** MUST match all
2186 expressions. If none of these attributes is present, then any version of the **policy set** is acceptable.
2187 In the case that more than one matching version can be obtained, then the most recent one
2188 SHOULD be used.

2189 **5.19. Element <PolicyIdReference>**

2190 The <xacml:PolicyIdReference> element SHALL be used to reference a <Policy> element
2191 by id. If <PolicyIdReference> is a URL, then it MAY be resolvable to the <Policy> element.
2192 However, the mechanism for resolving a **policy** reference to the corresponding **policy** is outside
2193 the scope of this specification.

```
2194 <xs:element name="PolicyIdReference" type="xacml:IdReferenceType"/>
```

2195 Element <PolicyIdReference> is of **xacml:IdReferenceType** complex type (see Section 5.18) .

2196 **5.20. Simple type VersionType**

2197 Elements of this type SHALL contain the version number of the **policy** or **policy set**.

```
2198 <xs:simpleType name="VersionType">  
2199   <xs:restriction base="xs:string">  
2200     <xs:pattern value="(\d+\.)*\d+"/>  
2201   </xs:restriction>  
2202 </xs:simpleType>
```

2203 The version number is expressed as a sequence of decimal numbers, each separated by a period
2204 (.). 'd+' represents a sequence of one or more decimal digits.

2205 **5.21. Simple type VersionMatchType**

2206 Elements of this type SHALL contain a restricted regular expression matching a version number
2207 (see Section 5.20). The expression SHALL match versions of a referenced **policy** or **policy set**
2208 that are acceptable for inclusion in the referencing **policy** or **policy set**.

```
2209 <xs:simpleType name="VersionMatchType">  
2210   <xs:restriction base="xs:string">  
2211     <xs:pattern value="((\d+|\*)\.)*(\d+|\*|\+)" />  
2212   </xs:restriction>  
2213 </xs:simpleType>
```

2214 A version match is '.'-separated, like a version string. A number represents a direct numeric match.
2215 A '*' means that any single number is valid. A '+' means that any number, and any subsequent
2216 numbers, are valid. In this manner, the following four patterns would all match the version string
2217 '1.2.3': '1.2.3', '1.*.3', '1.2.*' and '1.+'

2218 **5.22. Element <Policy>**

2219 The <Policy> element is the smallest entity that SHALL be presented to the **PDP** for evaluation.

2220 A <Policy> element MAY be evaluated, in which case the evaluation procedure defined in
2221 Section 7.10 SHALL be used.

2222 The main components of this element are the <Target>, <Rule>, <CombinerParameters>,
2223 <RuleCombinerParameters> and <Obligations> elements and the RuleCombiningAlgId
2224 attribute.

2225 The <Target> element defines the applicability of the <Policy> element to a set of **decision**
2226 **requests**. If the <Target> element within the <Policy> element matches the **request context**,
2227 then the <Policy> element MAY be used by the **PDP** in making its **authorization decision**. See
2228 Section 7.10.

2229 The <Policy> element includes a sequence of choices between <VariableDefinition> and
2230 <Rule> elements.

2231 **Rules** included in the <Policy> element MUST be combined by the algorithm specified by the
2232 RuleCombiningAlgId attribute.

2233 The <Obligations> element contains a set of **obligations** that MUST be fulfilled by the **PEP** in
2234 conjunction with the **authorization decision**.

```
2235 <xs:element name="Policy" type="xacml:PolicyType"/>
2236 <xs:complexType name="PolicyType">
2237   <xs:sequence>
2238     <xs:element ref="xacml:Description" minOccurs="0"/>
2239     <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>
2240     <xs:element ref="xacml:CombinerParameters" minOccurs="0"/>
2241     <xs:element ref="xacml:Target"/>
2242     <xs:choice maxOccurs="unbounded">
2243       <xs:element ref="xacml:CombinerParameters" minOccurs="0"/>
2244       <xs:element ref="xacml:RuleCombinerParameters" minOccurs="0"/>
2245       <xs:element ref="xacml:VariableDefinition"/>
2246       <xs:element ref="xacml:Rule"/>
2247     </xs:choice>
2248     <xs:element ref="xacml:Obligations" minOccurs="0"/>
2249   </xs:sequence>
2250   <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
2251   <xs:attribute name="Version" type="xacml:VersionType" default="1.0"/>
2252   <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>
2253 </xs:complexType>
```

2254 The <Policy> element is of **PolicyType** complex type.

2255 The <Policy> element contains the following attributes and elements:

2256 PolicyId [Required]

2257 **Policy** identifier. It is the responsibility of the **PAP** to ensure that no two **policies** visible to
2258 the **PDP** have the same identifier. This MAY be achieved by following a predefined URN or
2259 URI scheme. If the **policy** identifier is in the form of a URL, then it MAY be resolvable.

2260 Version [Default 1.0]

2261 The version number of the **Policy**.

2262 RuleCombiningAlgId [Required]

2263 The identifier of the **rule-combining algorithm** by which the <Policy>, ,
2264 <CombinerParameters> and <RuleCombinerParameters> components MUST be
2265 combined. Standard **rule-combining algorithms** are listed in Appendix C. Standard **rule-**
2266 **combining algorithm** identifiers are listed in Section B.10.

2267 <Description> [Optional]

2268 A free-form description of the **policy**. See Section 5.2.

2269 <PolicyDefaults> [Optional]

- 2270 Defines a set of default values applicable to the **policy**. The scope of the
2271 <PolicyDefaults> element SHALL be the enclosing **policy**.
- 2272 <CombinerParameters> [Optional]
- 2273 A sequence of parameters to be used by the **rule-combining algorithm**.
- 2274 <RuleCombinerParameters> [Optional]
- 2275 A sequence of parameters to be used by the **rule-combining algorithm**.
- 2276 <Target> [Required]
- 2277 The <Target> element defines the applicability of a <Policy> to a set of **decision requests**.
- 2278 The <Target> element MAY be declared by the creator of the <Policy> element, or it
2279 MAY be computed from the <Target> elements of the referenced <Rule> elements either
2280 as an intersection or as a union.
- 2281 <VariableDefinition> [Any Number]
- 2282 Common function definitions that can be referenced from anywhere in a **rule** where an
2283 expression can be found.
- 2284 <Rule> [Any Number]
- 2285 A sequence of **rules** that MUST be combined according to the RuleCombiningAlgId
2286 attribute. **Rules** whose <Target> elements match the **decision request** MUST be
2287 considered. **Rules** whose <Target> elements do not match the **decision request** SHALL
2288 be ignored.
- 2289 <Obligations> [Optional]
- 2290 A **conjunctive sequence** of **obligations** that MUST be fulfilled by the **PEP** in conjunction
2291 with the **authorization decision**. See Section 7.14 for a description of how the set of
2292 **obligations** to be returned by the **PDP** SHALL be determined.

2293 5.23. Element <PolicyDefaults>

2294 The <PolicyDefaults> element SHALL specify default values that apply to the <Policy>
2295 element.

```
2296 <xs:element name="PolicyDefaults" type="xacml:DefaultsType"/>
2297 <xs:complexType name="DefaultsType">
2298   <xs:sequence>
2299     <xs:choice>
2300       <xs:element ref="xacml:XPathVersion" minOccurs="0"/>
2301     </xs:choice>
2302   </xs:sequence>
2303 </xs:complexType>
```

2304 <PolicyDefaults> element is of **DefaultsType** complex type.

2305 The <PolicyDefaults> element contains the following elements:

2306 <XPathVersion> [Optional]

2307 Default XPath version.

2308

5.24. Element <CombinerParameters>

2309 The <CombinerParameters> element conveys parameters for a *policy-* or *rule-combining*
2310 *algorithm*.

2311 If multiple <CombinerParameters> elements occur within the same *policy* or *policy set*, they
2312 SHALL be considered equal to one <CombinerParameters> element containing the
2313 concatenation of all the sequences of <CombinerParameters> contained in all the aforementioned
2314 <CombinerParameters> elements, such that the order of occurrence of the
2315 <CominberParameters> elements is preserved in the concatenation of the
2316 <CombinerParameter> elements.

2317 Note that none of the *combining algorithms* specified in XACML 2.0 is parameterized.

```
2318 <xs:element name="CombinerParameters" type="xacml:CombinerParametersType" />  
2319 <xs:complexType name="CombinerParametersType">  
2320   <xs:sequence>  
2321     <xs:element ref="xacml:CombinerParameter" minOccurs="0"  
2322     maxOccurs="unbounded" />  
2323   </xs:sequence>  
2324 </xs:complexType>
```

2325 The <CombinerParameters> element is of **CombinerParametersType** complex type.

2326 The <CombinerParameters> element contains the following elements:

2327 <CombinerParameter> [Any Number]

2328 A single parameter. See Section 5.25.

2329 Support for the <CombinerParameters> element is optional.

2330 5.25. Element <CombinerParameter>

2331 The <CombinerParameter> element conveys a single parameter for a *policy-* or *rule-*
2332 *combining algorithm*.

```
2333 <xs:element name="CombinerParameter" type="xacml:CombinerParameterType" />  
2334 <xs:complexType name="CombinerParameterType">  
2335   <xs:sequence>  
2336     <xs:element ref="xacml:AttributeValue" />  
2337   </xs:sequence>  
2338   <xs:attribute name="ParameterName" type="xs:string" use="required" />  
2339 </xs:complexType>
```

2340 The <CombinerParameter> element is of **CombinerParameterType** complex type.

2341 The <CombinerParameter> element contains the following attribute:

2342 ParameterName [Required]

2343 The identifier of the parameter.

2344 AttributeValue [Required]

2345 The value of the parameter.

2346 Support for the <CombinerParameter> element is optional.

2347

5.26. Element <RuleCombinerParameters>

2348 The <RuleCombinerParameters> element conveys **parameters** associated with a particular
2349 **rule** within a **policy** for a **rule-combining algorithm**.

2350 Each <RuleCombinerParameters> element MUST be associated with a **rule** contained within
2351 the same **policy**. If multiple <RuleCombinerParameters> elements reference the same **rule**,
2352 they SHALL be considered equal to one <RuleCombinerParameters> element containing the
2353 concatenation of all the sequences of <CombinerParameters> contained in all the aforementioned
2354 <RuleCombinerParameters> elements, such that the order of occurrence of the
2355 <RuleCominberParameters> elements is preserved in the concatenation of the
2356 <CombinerParameter> elements.

2357 Note that none of the **rule-combining algorithms** specified in XACML 2.0 is parameterized.

```
2358 <xs:element name="RuleCombinerParameters"  
2359 type="xacml:RuleCombinerParametersType"/>  
2360 <xs:complexType name="RuleCombinerParametersType">  
2361   <xs:complexContent>  
2362     <xs:extension base="xacml:CombinerParametersType">  
2363       <xs:attribute name="RuleIdRef" type="xs:string" use="required"/>  
2364     </xs:extension>  
2365   </xs:complexContent>  
2366 </xs:complexType>
```

2367 The <RuleCombinerParameters> element contains the following elements:

2368 RuleIdRef [Required]

2369 The identifier of the <Rule> contained in the **policy**.

2370 Support for the <RuleCombinerParameters> element is optional, only if support for **combiner**
2371 **parameters** is optional.

5.27. Element <PolicyCombinerParameters>

2373 The <PolicyCombinerParameters> element conveys **parameters** associated with a particular
2374 **policy** within a **policy set** for a **policy-combining algorithm**.

2375 Each <PolicyCombinerParameters> element MUST be associated with a **policy** contained
2376 within the same **policy set**. If multiple <PolicyCombinerParameters> elements reference the
2377 same **policy**, they SHALL be considered equal to one <PolicyCombinerParameters> element
2378 containing the concatenation of all the sequences of <CombinerParameters> contained in all the
2379 aforementioned <PolicyCombinerParameters> elements, such that the order of occurrence of
2380 the <PolicyCominberParameters> elements is preserved in the concatenation of the
2381 <CombinerParameter> elements.

2382 Note that none of the **policy-combining algorithms** specified in XACML 2.0 is parameterized.

```
2383 <xs:element name="PolicyCombinerParameters"  
2384 type="xacml:PolicyCombinerParametersType"/>  
2385 <xs:complexType name="PolicyCombinerParametersType">  
2386   <xs:complexContent>  
2387     <xs:extension base="xacml:CombinerParametersType">  
2388       <xs:attribute name="PolicyIdRef" type="xs:anyURI" use="required"/>  
2389     </xs:extension>  
2390   </xs:complexContent>  
2391 </xs:complexType>
```

2392 The <PolicyCombinerParameters> element is of **PolicyCombinerParametersType** complex
2393 type.

2394 The <PolicyCombinerParameters> element contains the following elements:

2395 PolicyIdRef [Required]

2396 The identifier of a <Policy> or the value of a <PolicyIdReference> contained in the
2397 **policy set**.

2398 Support for the <PolicyCombinerParameters> element is optional, only if support for
2399 **combiner parameters** is optional.

2400 **5.28. Element <PolicySetCombinerParameters>**

2401 The <PolicySetCombinerParameters> element conveys **parameters** associated with a
2402 particular **policy set** within a **policy set** for a **policy-combining algorithm**.

2403 Each <PolicySetCombinerParameters> element MUST be associated with a **policy set**
2404 contained within the same **policy set**. If multiple <PolicySetCombinerParameters> elements
2405 reference the same **policy set**, they SHALL be considered equal to one
2406 <PolicySetCombinerParameters> element containing the concatenation of all the sequences
2407 of <CombinerParameters> contained in all the aforementioned
2408 <PolicySetCombinerParameters> elements, such that the order of occurrence of the
2409 <PolicySetCominberParameters> elements is preserved in the concatenation of the
2410 <CombinerParameter> elements.

2411 Note that none of the **policy-combining algorithms** specified in XACML 2.0 is parameterized.

```
2412 <xs:element name="PolicySetCombinerParameters"  
2413 type="xacml:PolicySetCombinerParametersType" />  
2414 <xs:complexType name="PolicySetCombinerParametersType">  
2415   <xs:complexContent>  
2416     <xs:extension base="xacml:CombinerParametersType">  
2417       <xs:attribute name="PolicySetIdRef" type="xs:anyURI" use="required" />  
2418     </xs:extension>  
2419   </xs:complexContent>  
2420 </xs:complexType>
```

2421 The <PolicySetCombinerParameters> element is of **PolicySetCombinerParametersType**
2422 complex type.

2423 The <PolicySetCombinerParameters> element contains the following elements:

2424 PolicySetIdRef [Required]

2425 The identifier of a <PolicySet> or the value of a <PolicySetIdReference> contained
2426 in the **policy set**.

2427 Support for the <PolicySetCombinerParameters> element is optional, only if support for
2428 **combiner parameters** is optional.

2429 **5.29. Element <Rule>**

2430 The <Rule> element SHALL define the individual **rules** in the **policy**. The main components of
2431 this element are the <Target> and <Condition> elements and the **Effect** attribute.

2432 A <Rule> element MAY be evaluated, in which case the evaluation procedure defined in Section
2433 7.9 SHALL be used.

```
2434 <xs:element name="Rule" type="xacml:RuleType"/>
2435 <xs:complexType name="RuleType">
2436   <xs:sequence>
2437     <xs:element ref="xacml:Description" minOccurs="0"/>
2438     <xs:element ref="xacml:Target" minOccurs="0"/>
2439     <xs:element ref="xacml:Condition" minOccurs="0"/>
2440   </xs:sequence>
2441   <xs:attribute name="RuleId" type="xs:string" use="required"/>
2442   <xs:attribute name="Effect" type="xacml:EffectType" use="required"/>
2443 </xs:complexType>
```

2444 The <Rule> element is of **RuleType** complex type.

2445 The <Rule> element contains the following attributes and elements:

2446 RuleId [Required]

2447 A string identifying this *rule*.

2448 Effect [Required]

2449 **Rule effect.** The value of this attribute is either "Permit" or "Deny".

2450 <Description> [Optional]

2451 A free-form description of the *rule*.

2452 <Target> [Optional]

2453 Identifies the set of **decision requests** that the <Rule> element is intended to evaluate. If
2454 this element is omitted, then the **target** for the <Rule> SHALL be defined by the
2455 <Target> element of the enclosing <Policy> element. See Section 7.6 for details.

2456 <Condition> [Optional]

2457 A **predicate** that MUST be satisfied for the *rule* to be assigned its Effect value.

2458 **5.30. Simple type EffectType**

2459 The **EffectType** simple type defines the values allowed for the Effect attribute of the <Rule>
2460 element and for the FulfillOn attribute of the <Obligation> element.

```
2461 <xs:simpleType name="EffectType">
2462   <xs:restriction base="xs:string">
2463     <xs:enumeration value="Permit"/>
2464     <xs:enumeration value="Deny"/>
2465   </xs:restriction>
2466 </xs:simpleType>
```

2467 **5.31. Element <VariableDefinition>**

2468 The <VariableDefinition> element SHALL be used to define a value that can be referenced
2469 by a <VariableReference> element. The name supplied for its VariableId attribute SHALL
2470 NOT occur in the VariableId attribute of any other <VariableDefinition> element within the
2471 encompassing **policy**. The <VariableDefinition> element MAY contain undefined
2472 <VariableReference> element, but if it does, a corresponding <VariableDefinition> element
2473 MUST be defined later in the encompassing **policy**. <VariableDefinition> elements MAY be

2474 grouped together or MAY be placed close to the reference in the encompassing *policy*. There
2475 MAY be zero or more references to each <VariableDefinition> element.

```
2476 <xs:element name="VariableDefinition" type="xacml:VariableDefinitionType" />
2477 <xs:complexType name="VariableDefinitionType">
2478   <xs:sequence>
2479     <xs:element ref="xacml:Expression" />
2480   </xs:sequence>
2481   <xs:attribute name="VariableId" type="xs:string" use="required" />
2482 </xs:complexType>
```

2483 The <VariableDefinition> element is of **VariableDefinitionType** complex type. The
2484 <VariableDefinition> element has the following elements and attributes:

2485 <Expression> [Required]

2486 Any element of **ExpressionType** complex type.

2487 VariableId [Required]

2488 The name of the variable definition.

2489 **5.32. Element <VariableReference>**

2490 The <VariableReference> element is used to reference a value defined within the same
2491 encompassing <Policy> element. The <VariableReference> element SHALL refer to the
2492 <VariableDefinition> element by string equality on the value of their respective VariableId
2493 attributes. There SHALL exist one and only one <VariableDefinition> within the same
2494 encompassing <Policy> element to which the <VariableReference> refers. There MAY be
2495 zero or more <VariableReference> elements that refer to the same <VariableDefinition>
2496 element.

```
2497 <xs:element name="VariableReference" type="xacml:VariableReferenceType"
2498 substitutionGroup="xacml:Expression" />
2499 <xs:complexType name="VariableReferenceType">
2500   <xs:complexContent>
2501     <xs:extension base="xacml:ExpressionType">
2502       <xs:attribute name="VariableId" type="xs:string" use="required" />
2503     </xs:extension>
2504   </xs:complexContent>
2505 </xs:complexType>
```

2506 The <VariableReference> element is of the **VariableReferenceType** complex type, which is of
2507 the **ExpressionType** complex type and is a member of the <Expression> element substitution
2508 group. The <VariableReference> element MAY appear any place where an <Expression>
2509 element occurs in the schema.

2510 The <VariableReference> element has the following attributes:

2511 VariableId [Required]

2512 The name used to refer to the value defined in a <VariableDefinition> element.

2513 **5.33. Element <Expression>**

2514 The <Expression> element is not used directly in a *policy*. The <Expression> element
2515 signifies that an element that extends the **ExpressionType** and is a member of the
2516 <Expression> element substitution group SHALL appear in its place.

```
2517 <xs:element name="Expression" type="xacml:ExpressionType" abstract="true" />
```

2518 `<xs:complexType name="ExpressionType" abstract="true"/>`

2519 The following elements are in the `<Expression>` element substitution group:

2520 `<Apply>`, `<AttributeSelector>`, `<AttributeValue>`, `<Function>`,
2521 `<VariableReference>`, `<ActionAttributeDesignator>`,
2522 `<ResourceAttributeDesignator>`, `<SubjectAttributeDesignator>` and
2523 `<EnvironmentAttributeDesignator>`.

2524 **5.34. Element `<Condition>`**

2525 The `<Condition>` element is a Boolean function over ***subject***, ***resource***, ***action*** and
2526 ***environment attributes*** or functions of ***attributes***.

```
2527 <xs:element name="Condition" type="xacml:ConditionType"/>
2528 <xs:complexType name="ConditionType">
2529   <xs:sequence>
2530     <xs:element ref="xacml:Expression"/>
2531   </xs:sequence>
2532 </xs:complexType>
```

2533 The `<Condition>` contains one `<Expression>` element, with the restriction that the
2534 `<Expression>` return data-type MUST be "http://www.w3.org/2001/XMLSchema#boolean".
2535 Evaluation of the `<Condition>` element is described in Section 7.8.

2536 **5.35. Element `<Apply>`**

2537 The `<Apply>` element denotes application of a function to its arguments, thus encoding a function
2538 call. The `<Apply>` element can be applied to any combination of the members of the
2539 `<Expression>` element substitution group. See Section 5.33.

```
2540 <xs:element name="Apply" type="xacml:ApplyType"
2541 substitutionGroup="xacml:Expression"/>
2542 <xs:complexType name="ApplyType">
2543   <xs:complexContent>
2544     <xs:extension base="xacml:ExpressionType">
2545       <xs:sequence>
2546         <xs:element ref="xacml:Expression" minOccurs="0"
2547 maxOccurs="unbounded"/>
2548       </xs:sequence>
2549       <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>
2550     </xs:extension>
2551   </xs:complexContent>
2552 </xs:complexType>
```

2553 The `<Apply>` element is of **`ApplyType`** complex type.

2554 The `<Apply>` element contains the following attributes and elements:

2555 `FunctionId` [Required]

2556 The identifier of the function to be applied to the arguments. XACML-defined functions are
2557 described in Appendix A.

2558 `<Expression>` [Optional]

2559 Arguments to the function, which may include other functions.

2560

5.36. Element <Function>

2561 The <Function> element SHALL be used to name a function as an argument to the function
2562 defined by the parent <Apply> element. In the case where the parent <Apply> element is a
2563 higher-order **bag** function, the named function is applied to every element of the **bag** or **bags**
2564 identified in the other arguments of the parent element. The higher-order **bag** functions are
2565 described in Section A3A.3.12.

```
2566 <xs:element name="Function" type="xacml:FunctionType"  
2567 substitutionGroup="xacml:Expression"/>  
2568 <xs:complexType name="FunctionType">  
2569   <xs:complexContent>  
2570     <xs:extension base="xacml:ExpressionType">  
2571       <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>  
2572     </xs:extension>  
2573   </xs:complexContent>  
2574 </xs:complexType>
```

2575 The Function element is of **FunctionType** complex type.

2576 The Function element contains the following attributes:

2577 FunctionId [Required]

2578 The identifier of the function.

2579 5.37. Complex type AttributeDesignatorType

2580 The **AttributeDesignatorType** complex type is the type for elements that identify **attributes** by
2581 name. It contains the information required to match **attributes** in the request **context**. See Section
2582 7.2.4.

2583 It also contains information to control behaviour in the event that no matching **attribute** is present in
2584 the **context**.

2585 Elements of this type SHALL NOT alter the match semantics of **named attributes**, but MAY narrow
2586 the search space.

```
2587 <xs:complexType name="AttributeDesignatorType">  
2588   <xs:complexContent>  
2589     <xs:extension base="xacml:ExpressionType">  
2590       <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2591       <xs:attribute name="DataType" type="xs:anyURI" use="required"/>  
2592       <xs:attribute name="Issuer" type="xs:string" use="optional"/>  
2593       <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"  
2594 default="false"/>  
2595     </xs:extension>  
2596   </xs:complexContent>  
2597 </xs:complexType>
```

2598 A **named attribute** SHALL match an **attribute** if the values of their respective **AttributeId**,
2599 **DataType** and **Issuer** attributes match. The **attribute** designator's **AttributeId** MUST match,
2600 by URI equality, the **AttributeId** of the **attribute**. The **attribute** designator's **DataType** MUST
2601 match, by URI equality, the **DataType** of the same **attribute**.

2602 If the **Issuer** attribute is present in the **attribute** designator, then it MUST match, using the
2603 "urn:oasis:names:tc:xacml:1.0:function:string-equal" function, the **Issuer** of the same **attribute**. If
2604 the **Issuer** is not present in the **attribute** designator, then the matching of the **attribute** to the
2605 **named attribute** SHALL be governed by **AttributeId** and **DataType** attributes alone.

2606 The <AttributeDesignatorType> contains the following attributes:

2607 AttributeId [Required]

2608 This attribute SHALL specify the AttributeId with which to match the **attribute**.

2609 DataType [Required]

2610 The bag returned by the <AttributeDesignator> element SHALL contain values of this
2611 data-type.

2612 Issuer [Optional]

2613 This attribute, if supplied, SHALL specify the Issuer with which to match the **attribute**.

2614 MustBePresent [Optional]

2615 This attribute governs whether the element returns "Indeterminate" or an empty **bag** in the
2616 event the **named attribute** is absent from the request **context**. See Section 7.2.5. Also
2617 see Sections 7.15.2 and 7.15.3.

2618 **5.38. Element <SubjectAttributeDesignator>**

2619 The <SubjectAttributeDesignator> element retrieves a **bag** of values for a **named**
2620 categorized **subject attribute** from the request **context**. A **subject attribute** is an **attribute**
2621 contained within a <Subject> element of the request **context**. A categorized **subject** is a **subject**
2622 that is identified by a particular **subject-category** attribute. A **named categorized subject attribute**
2623 is a **named subject attribute** for a particular **categorized subject**.

2624 The <SubjectAttributeDesignator> element SHALL return a **bag** containing all the **subject**
2625 **attribute** values that are matched by the **named categorized subject attribute**. In the event that
2626 no matching attribute is present in the context, the MustBePresent attribute governs whether this
2627 element returns an empty **bag** or "Indeterminate". See Section 7.2.5.

2628 The **SubjectAttributeDesignatorType** extends the match semantics of the
2629 **AttributeDesignatorType** (See Section 5.37) such that it narrows the **attribute** search space to
2630 the specific **categorized subject** such that the value of this element's SubjectCategory attribute
2631 matches, by URI equality, the value of the request **context's** <Subject> element's
2632 SubjectCategory attribute.

2633 If the request context contains multiple **subjects** with the same SubjectCategory XML attribute,
2634 then they SHALL be treated as if they were one **categorized subject**.

2635 The <SubjectAttributeDesignator> MAY appear in the <SubjectMatch> element and
2636 MAY be passed to the <Apply> element as an argument.

```
2637 <xs:element name="SubjectAttributeDesignator"
2638 type="xacml:SubjectAttributeDesignatorType"
2639 substitutionGroup="xacml:Expression" />
2640 <xs:complexType name="SubjectAttributeDesignatorType">
2641   <xs:complexContent>
2642     <xs:extension base="xacml:AttributeDesignatorType">
2643       <xs:attribute name="SubjectCategory" type="xs:anyURI" use="optional"
2644 default="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"/>
2645     </xs:extension>
2646   </xs:complexContent>
2647 </xs:complexType>
```

2648 The <SubjectAttributeDesignator> element is of type **SubjectAttributeDesignatorType**.
2649 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType**
2650 complex type with a SubjectCategory attribute.

2651 SubjectCategory [Optional]

2652 This attribute SHALL specify the *categorized subject* from which to match *named subject*
2653 *attributes*. If SubjectCategory is not present, then its default value of
2654 "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" SHALL be used. Standard
2655 values of the SubjectCategory are listed in Section B.2.

2656 **5.39. Element <ResourceAttributeDesignator>**

2657 The <ResourceAttributeDesignator> element retrieves a **bag** of values for a *named*
2658 **resource attribute** from the request **context**. A **resource attribute** is an **attribute** contained
2659 within the <Resource> element of the request **context**. A *named resource attribute* is a **named**
2660 **attribute** that matches a **resource attribute**. A *named resource attribute* SHALL be considered
2661 *present* if there is at least one **resource attribute** that matches the criteria set out below. A
2662 **resource attribute** value is an **attribute** value that is contained within a **resource attribute**.

2663 The <ResourceAttributeDesignator> element SHALL return a **bag** containing all the
2664 **resource attribute** values that are matched by the *named resource attribute*. In the event that no
2665 matching attribute is present in the context, the MustBePresent attribute governs whether this
2666 element returns an empty **bag** or "Indeterminate". See Section 7.2.5.

2667 A *named resource attribute* SHALL match a **resource attribute** as per the match semantics
2668 specified in the **AttributeDesignatorType** complex type. See Section 5.37.

2669 The <ResourceAttributeDesignator> MAY appear in the <ResourceMatch> element and
2670 MAY be passed to the <Apply> element as an argument.

```
2671 <xs:element name="ResourceAttributeDesignator"  
2672 type="xacml:AttributeDesignatorType" substitutionGroup="xacml:Expression" />
```

2673 The <ResourceAttributeDesignator> element is of the **AttributeDesignatorType** complex
2674 type.

2675 **5.40. Element <ActionAttributeDesignator>**

2676 The <ActionAttributeDesignator> element retrieves a **bag** of values for a *named action*
2677 **attribute** from the request **context**. An **action attribute** is an **attribute** contained within the
2678 <Action> element of the request **context**. A *named action attribute* has specific criteria
2679 (described below) with which to match an **action attribute**. A *named action attribute* SHALL be
2680 considered *present*, if there is at least one **action attribute** that matches the criteria. An **action**
2681 **attribute value** is an **attribute value** that is contained within an **action attribute**.

2682 The <ActionAttributeDesignator> element SHALL return a **bag** of all the **action attribute**
2683 values that are matched by the *named action attribute*. In the event that no matching attribute is
2684 present in the context, the MustBePresent attribute governs whether this element returns an
2685 empty **bag** or "Indeterminate". See Section 7.2.5.

2686 A *named action attribute* SHALL match an **action attribute** as per the match semantics specified
2687 in the **AttributeDesignatorType** complex type. See Section 5.37.

2688 The <ActionAttributeDesignator> MAY appear in the <ActionMatch> element and MAY
2689 be passed to the <Apply> element as an argument.

2690 <xs:element name="ActionAttributeDesignator" type="xacml:AttributeDesignatorType"
2691 substitutionGroup="xacml:Expression" />

2692 The <ActionAttributeDesignator> element is of the **AttributeDesignatorType** complex
2693 type.

2694 **5.41. Element <EnvironmentAttributeDesignator>**

2695 The <EnvironmentAttributeDesignator> element retrieves a **bag** of values for a *named*
2696 **environment attribute** from the request **context**. An **environment attribute** is an **attribute**
2697 contained within the <Environment> element of request **context**. A *named environment*
2698 **attribute** has specific criteria (described below) with which to match an **environment attribute**. A
2699 *named environment attribute* SHALL be considered *present*, if there is at least one **environment**
2700 **attribute** that matches the criteria. An **environment attribute value** is an **attribute** value that is
2701 contained within an **environment attribute**.

2702 The <EnvironmentAttributeDesignator> element SHALL evaluate to a **bag** of all the
2703 **environment attribute** values that are matched by the *named environment attribute*. In the
2704 event that no matching attribute is present in the context, the `MustBePresent` attribute governs
2705 whether this element returns an empty **bag** or "Indeterminate". See Section 7.2.5.

2706 A *named environment attribute* SHALL match an **environment attribute** as per the match
2707 semantics specified in the **AttributeDesignatorType** complex type. See Section 5.37.

2708 The <EnvironmentAttributeDesignator> MAY be passed to the <Apply> element as an
2709 argument.

2710 <xs:element name="EnvironmentAttributeDesignator"
2711 type="xacml:AttributeDesignatorType" substitutionGroup="xacml:Expression" />

2712 The <EnvironmentAttributeDesignator> element is of the **AttributeDesignatorType**
2713 complex type.

2714 **5.42. Element <AttributeSelector>**

2715 The <AttributeSelector> element identifies attributes by their location in the request **context**.
2716 Support for the <AttributeSelector> element is OPTIONAL.

2717 The <AttributeSelector> element's `RequestContextPath` XML attribute SHALL contain a
2718 legal XPath expression whose context node is the <xacml-context:Request> element. The
2719 **AttributeSelector** element SHALL evaluate to a **bag** of values whose data-type is specified by
2720 the element's `DataType` attribute. If the `DataType` specified in the **AttributeSelector** is a
2721 primitive data type defined in [XF] or [XS], then the value returned by the XPath expression SHALL
2722 be converted to the `DataType` specified in the <AttributeSelector> using the constructor
2723 function below [XF Section 4] that corresponds to the `DataType`. If an error results from using the
2724 constructor function, then the value of the <AttributeSelector> SHALL be "Indeterminate".

2725
2726 `xs:string()`
2727 `xs:boolean()`
2728 `xs:integer()`
2729 `xs:double()`
2730 `xs:dateTime()`
2731 `xs:date()`
2732 `xs:time()`
2733 `xs:hexBinary()`
2734 `xs:base64Binary()`
2735 `xs:anyURI()`

2736 xf:yearMonthDuration()
2737 xf:dayTimeDuration()

2738

2739 If the `DataType` specified in the `AttributeSelector` is not one of the preceding primitive
2740 DataTypes, then the `AttributeSelector` SHALL return a **bag** of instances of the specified
2741 DataType. If an error occurs when converting the values returned by the XPath expression to the
2742 specified `DataType`, then the result of the `AttributeSelector` SHALL be "Indeterminate".

2743

2744 Each node selected by the specified XPath expression MUST be either a text node, an attribute
2745 node, a processing instruction node or a comment node. The string representation of the value of
2746 each node MUST be converted to an **attribute** value of the specified data-type, and the result of
2747 the `AttributeSelector` is the **bag** of the **attribute** values generated from all the selected
2748 nodes.

2749

2750 If the node selected by the specified XPath expression is not one of those listed above (i.e. a text
2751 node, an attribute node, a processing instruction node or a comment node), then the result of the
2752 enclosing **policy** SHALL be "Indeterminate" with a `Status Code` value of
2753 "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

2754

```
2755 <xs:element name="AttributeSelector" type="xacml:AttributeSelectorType"  
2756 substitutionGroup="xacml:Expression" />  
2757 <xs:complexType name="AttributeSelectorType">  
2758   <xs:complexContent>  
2759     <xs:extension base="xacml:ExpressionType">  
2760       <xs:attribute name="RequestContextPath" type="xs:string" use="required" />  
2761       <xs:attribute name="DataType" type="xs:anyURI" use="required" />  
2762       <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"  
2763       default="false" />  
2764     </xs:extension>  
2765   </xs:complexContent>  
2766 </xs:complexType>
```

2767 The `<AttributeSelector>` element is of **AttributeSelectorType** complex type.

2768 The `<AttributeSelector>` element has the following attributes:

2769 RequestContextPath [Required]

2770 An XPath expression whose context node is the `<xacml-context:Request>` element.
2771 There SHALL be no restriction on the XPath syntax. See also Section 5.4.

2772 DataType [Required]

2773 The **bag** returned by the `<AttributeSelector>` element SHALL contain values of this
2774 data-type.

2775 MustBePresent [Optional]

2776 This attribute governs whether the element returns "Indeterminate" or an empty **bag** in the
2777 event the XPath expression selects no node. See Section 7.2.5. Also see Sections 7.15.2
2778 and 7.15.3.

2779 5.43. Element `<AttributeValue>`

2780 The `<xacml:AttributeValue>` element SHALL contain a literal **attribute** value.

```
2781 <xs:element name="AttributeValue" type="xacml:AttributeValueType"  
2782 substitutionGroup="xacml:Expression" />  
2783 <xs:complexType name="AttributeValueType" mixed="true">
```

```

2784 <xs:complexContent>
2785   <xs:extension base="xacml:ExpressionType">
2786     <xs:sequence>
2787       <xs:any namespace="##any" processContents="lax" minOccurs="0"
2788 maxOccurs="unbounded" />
2789     </xs:sequence>
2790     <xs:attribute name="DataType" type="xs:anyURI" use="required" />
2791     <xs:anyAttribute namespace="##any" processContents="lax" />
2792   </xs:extension>
2793 </xs:complexContent>
2794 </xs:complexType>

```

2795 The <xacml:AttributeValue> element is of **AttributeValueType** complex type.

2796 The <xacml:AttributeValue> element has the following attributes:

2797 **DataType** [Required]

2798 The data-type of the *attribute* value.

2799 **5.44. Element <Obligations>**

2800 The <Obligations> element SHALL contain a set of <Obligation> elements.

2801 Support for the <Obligations> element is OPTIONAL.

```

2802 <xs:element name="Obligations" type="xacml:ObligationsType" />
2803 <xs:complexType name="ObligationsType">
2804   <xs:sequence>
2805     <xs:element ref="xacml:Obligation" maxOccurs="unbounded" />
2806   </xs:sequence>
2807 </xs:complexType>

```

2808 The <Obligations> element is of **ObligationsType** complexType.

2809 The <Obligations> element contains the following element:

2810 <Obligation> [One to Many]

2811 A sequence of *obligations*. See Section 5.45.

2812 **5.45. Element <Obligation>**

2813 The <Obligation> element SHALL contain an identifier for the *obligation* and a set of *attributes* that form arguments of the action defined by the *obligation*. The FulfillOn attribute SHALL indicate the *effect* for which this *obligation* must be fulfilled by the *PEP*.

```

2816 <xs:element name="Obligation" type="xacml:ObligationType" />
2817 <xs:complexType name="ObligationType">
2818   <xs:sequence>
2819     <xs:element ref="xacml:AttributeAssignment" minOccurs="0"
2820 maxOccurs="unbounded" />
2821   </xs:sequence>
2822   <xs:attribute name="ObligationId" type="xs:anyURI" use="required" />
2823   <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required" />
2824 </xs:complexType>

```

2825 The <Obligation> element is of **ObligationType** complexType. See Section 7.14 for a description of how the set of *obligations* to be returned by the *PDP* is determined.

2827 The <Obligation> element contains the following elements and attributes:

2828 ObligationId [Required]

2829 **Obligation** identifier. The value of the **obligation** identifier SHALL be interpreted by the
2830 **PEP**.

2831 FulfillOn [Required]

2832 The **effect** for which this **obligation** must be fulfilled by the **PEP**.

2833 <AttributeAssignment> [Optional]

2834 **Obligation** arguments assignment. The values of the **obligation** arguments SHALL be
2835 interpreted by the **PEP**.

2836 **5.46. Element <AttributeAssignment>**

2837 The <AttributeAssignment> element is used for including arguments in **obligations**. It SHALL
2838 contain an AttributeId and the corresponding **attribute** value, by extending the
2839 **AttributeValueType** type definition. The <AttributeAssignment> element MAY be used in
2840 any way that is consistent with the schema syntax, which is a sequence of <xs:any> elements.
2841 The value specified SHALL be understood by the **PEP**, but it is not further specified by XACML.
2842 See Section 7.14. Section 4.2.4.3 provides a number of examples of arguments included in
2843 **obligations**.

```
2844 <xs:element name="AttributeAssignment" type="xacml:AttributeAssignmentType"/>  
2845 <xs:complexType name="AttributeAssignmentType" mixed="true">  
2846     <xs:complexContent>  
2847         <xs:extension base="xacml:AttributeValueType">  
2848             <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2849         </xs:extension>  
2850     </xs:complexContent>  
2851 </xs:complexType>
```

2852 The <AttributeAssignment> element is of **AttributeAssignmentType** complex type.

2853 The <AttributeAssignment> element contains the following attributes:

2854 AttributeId [Required]

2855 The **attribute** Identifier.

2856 **6. Context syntax (normative with the exception of** 2857 **the schema fragments)**

2858 **6.1. Element <Request>**

2859 The <Request> element is a top-level element in the XACML **context** schema. The <Request>
2860 element is an abstraction layer used by the policy language. For simplicity of expression, this
2861 document describes **policy** evaluation in terms of operations on the **context**. However a
2862 conforming **PDP** is not required to actually instantiate the **context** in the form of an XML document.
2863 But, any system conforming to the XACML specification MUST produce exactly the same
2864 **authorization decisions** as if all the inputs had been transformed into the form of an <xacml-
2865 context:Request> element.

2866 The <Request> element contains <Subject>, <Resource>, <Action> and <Environment>
2867 elements. There may be multiple <Subject> elements and, under some conditions, multiple
2868 <Resource> elements². Each child element contains a sequence of <xacml-
2869 context:Attribute> elements associated with the **subject**, **resource**, **action** and
2870 **environment** respectively. These <Attribute> elements MAY form a part of **policy** evaluation.

```
2871 <xs:element name="Request" type="xacml-context:RequestType" />  
2872 <xs:complexType name="RequestType">  
2873   <xs:sequence>  
2874     <xs:element ref="xacml-context:Subject" maxOccurs="unbounded" />  
2875     <xs:element ref="xacml-context:Resource" maxOccurs="unbounded" />  
2876     <xs:element ref="xacml-context:Action" />  
2877     <xs:element ref="xacml-context:Environment" />  
2878   </xs:sequence>  
2879 </xs:complexType>
```

2880 The <Request> element is of **RequestType** complex type.

2881 The <Request> element contains the following elements:

2882 <Subject> [One to Many]

2883 Specifies information about a **subject** of the request **context** by listing a sequence of
2884 <Attribute> elements associated with the **subject**. One or more <Subject> elements
2885 are allowed. A **subject** is an entity associated with the **access** request. For example, one
2886 **subject** might represent the human user that initiated the application from which the
2887 request was issued; another **subject** might represent the application's executable code
2888 responsible for creating the request; another **subject** might represent the machine on
2889 which the application was executing; and another **subject** might represent the entity that is
2890 to be the recipient of the **resource**. Attributes of each of these entities MUST be enclosed
2891 in separate <Subject> elements.

2892 <Resource> [One to Many]

2893 Specifies information about the **resource** or **resources** for which **access** is being
2894 requested by listing a sequence of <Attribute> elements associated with the **resource**.
2895 It MAY include a <ResourceContent> element.

2896 <Action> [Required]

2897 Specifies the requested **action** to be performed on the **resource** by listing a set of
2898 <Attribute> elements associated with the **action**.

2899 <Environment> [Required]

2900 Contains a set of <Attribute> elements for the **environment**.

2901 **6.2. Element <Subject>**

2902 The <Subject> element specifies a **subject** by listing a sequence of <Attribute> elements
2903 associated with the **subject**.

```
2904 <xs:element name="Subject" type="xacml-context:SubjectType" />  
2905 <xs:complexType name="SubjectType">  
2906   <xs:sequence>
```

² The conditions under which multiple <Resource> elements are allowed are described in the XACML Profile for Multiple Resources [MULT].

```

2907     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2908 maxOccurs="unbounded" />
2909   </xs:sequence>
2910   <xs:attribute name="SubjectCategory" type="xs:anyURI"
2911 default="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" />
2912 </xs:complexType>

```

2913 The <Subject> element is of **SubjectType** complex type.

2914 The <Subject> element contains the following elements and attributes:

2915 SubjectCategory [Optional]

2916 This attribute indicates the role that the parent <Subject> played in the formation of the
 2917 **access** request. If this attribute is not present in a given <Subject> element, then the
 2918 default value of "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" SHALL be
 2919 used, indicating that the parent <Subject> element represents the entity ultimately
 2920 responsible for initiating the **access** request.

2921 If more than one <Subject> element contains a "urn:oasis:names:tc:xacml:2.0:subject-
 2922 category" attribute with the same value, then the PDP SHALL treat the contents of those
 2923 elements as if they were contained in the same <Subject> element.

2924 <Attribute> [Any Number]

2925 A sequence of **attributes** that apply to the subject.

2926 Typically, a <Subject> element will contain an <Attribute> with an AttributeId of
 2927 "urn:oasis:names:tc:xacml:1.0:subject:subject-id", containing the identity of the **subject**.

2928 A <Subject> element MAY contain additional <Attribute> elements.

2929 **6.3. Element <Resource>**

2930 The <Resource> element specifies information about the **resource** to which **access** is requested,
 2931 by listing a sequence of <Attribute> elements associated with the **resource**. It MAY include the
 2932 **resource** content.

```

2933 <xs:element name="Resource" type="xacml-context:ResourceType" />
2934 <xs:complexType name="ResourceType">
2935   <xs:sequence>
2936     <xs:element ref="xacml-context:ResourceContent" minOccurs="0" />
2937     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2938 maxOccurs="unbounded" />
2939   </xs:sequence>
2940 </xs:complexType>

```

2941 The <Resource> element is of **ResourceType** complex type.

2942 The <Resource> element contains the following elements:

2943 <ResourceContent> [Optional]

2944 The **resource** content.

2945 <Attribute> [Any Number]

2946 A sequence of **resource attributes**.

2947 The <Resource> element MAY contain one or more <Attribute> elements with an
 2948 AttributeId of "urn:oasis:names:tc:xacml:2.0:resource:resource-id". Each such

2949 <Attribute> SHALL be an absolute and fully-resolved representation of the identity of
2950 the single **resource** to which access is being requested. If there is more than one such
2951 absolute and fully-resolved representation, and if any <Attribute> with this
2952 AttributeId is specified, then an <Attribute> for each such distinct representation of
2953 the **resource** identity SHALL be specified. All such <Attribute> elements SHALL refer
2954 to the same single **resource** instance. A Profile for a particular **resource** MAY specify a
2955 single normative representation for instances of the **resource**; in this case, any
2956 <Attribute> with this AttributeId SHALL use only this one representation.

2957 A <Resource> element MAY contain additional <Attribute> elements.

2958 6.4. Element <ResourceContent>

2959 The <ResourceContent> element is a notional placeholder for the content of the **resource**. If an
2960 XACML **policy** references the contents of the **resource** by means of an <AttributeSelector>
2961 element, then the <ResourceContent> element MUST be included in the
2962 RequestContextPath string.

```
2963 <xs:complexType name="ResourceContentType" mixed="true">  
2964   <xs:sequence>  
2965     <xs:any namespace="##any" processContents="lax" minOccurs="0"  
2966     maxOccurs="unbounded" />  
2967   </xs:sequence>  
2968   <xs:anyAttribute namespace="##any" processContents="lax" />  
2969 </xs:complexType>
```

2970 The <ResourceContent> element is of **ResourceContentType** complex type.

2971 The <ResourceContent> element allows arbitrary elements and attributes.

2972 6.5. Element <Action>

2973 The <Action> element specifies the requested **action** on the **resource**, by listing a set of
2974 <Attribute> elements associated with the **action**.

```
2975 <xs:element name="Action" type="xacml-context:ActionType" />  
2976 <xs:complexType name="ActionType">  
2977   <xs:sequence>  
2978     <xs:element ref="xacml-context:Attribute" minOccurs="0"  
2979     maxOccurs="unbounded" />  
2980   </xs:sequence>  
2981 </xs:complexType>
```

2982 The <Action> element is of **ActionType** complex type.

2983 The <Action> element contains the following elements:

2984 <Attribute> [Any Number]

2985 List of **attributes** of the **action** to be performed on the **resource**.

2986 6.6. Element <Environment>

2987 The <Environment> element contains a set of **attributes** of the **environment**.

```
2988 <xs:element name="Environment" type="xacml-context:EnvironmentType" />  
2989 <xs:complexType name="EnvironmentType">  
2990   <xs:sequence>
```

```

2991     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2992     maxOccurs="unbounded" />
2993     </xs:sequence>
2994 </xs:complexType>

```

2995 The <Environment> element is of **EnvironmentType** complex type.

2996 The <Environment> element contains the following elements:

2997 <Attribute> [Any Number]

2998 A list of **environment attributes**. Environment **attributes** are **attributes** that are not
2999 associated with either the **resource**, the **action** or any of the **subjects** of the **access**
3000 request.

3001 6.7. Element <Attribute>

3002 The <Attribute> element is the central abstraction of the request **context**. It contains **attribute**
3003 meta-data and one or more **attribute** values. The **attribute** meta-data comprises the **attribute**
3004 identifier and the **attribute** issuer. <AttributeDesignator> and <AttributeSelector>
3005 elements in the **policy** MAY refer to **attributes** by means of this meta-data.

```

3006 <xs:element name="Attribute" type="xacml-context:AttributeType" />
3007 <xs:complexType name="AttributeType">
3008   <xs:sequence>
3009     <xs:element ref="xacml-context:AttributeValue" maxOccurs="unbounded" />
3010   </xs:sequence>
3011   <xs:attribute name="AttributeId" type="xs:anyURI" use="required" />
3012   <xs:attribute name="DataType" type="xs:anyURI" use="required" />
3013   <xs:attribute name="Issuer" type="xs:string" use="optional" />
3014 </xs:complexType>

```

3015 The <Attribute> element is of **AttributeType** complex type.

3016 The <Attribute> element contains the following attributes and elements:

3017 AttributeId [Required]

3018 The **Attribute** identifier. A number of identifiers are reserved by XACML to denote
3019 commonly used **attributes**. See Appendix B.

3020 DataType [Required]

3021 The data-type of the contents of the <xacml-context:AttributeValue> element.
3022 This SHALL be either a primitive type defined by the XACML 2.0 specification or a type
3023 (primitive or structured) defined in a namespace declared in the <xacml-context>
3024 element.

3025 Issuer [Optional]

3026 The **Attribute** issuer. For example, this attribute value MAY be an x500Name that binds to
3027 a public key, or it may be some other identifier exchanged out-of-band by issuing and
3028 relying parties.

3029 <xacml-context:AttributeValue> [One to Many]

3030 One or more **attribute** values. Each **attribute** value MAY have contents that are empty,
3031 occur once or occur multiple times.

3032

6.8. Element <AttributeValue>

3033 The <xacml-context:AttributeValue> element contains the value of an *attribute*.

```
3034 <xs:element name="AttributeValue" type="xacml-context:AttributeValueType"/>
3035 <xs:complexType name="AttributeValueType" mixed="true">
3036   <xs:sequence>
3037     <xs:any namespace="##any" processContents="lax" minOccurs="0"
3038     maxOccurs="unbounded"/>
3039   </xs:sequence>
3040   <xs:anyAttribute namespace="##any" processContents="lax"/>
3041 </xs:complexType>
```

3042 The <xacml-context:AttributeValue> element is of **AttributeValueType** complex type.

3043 The data-type of the <xacml-context:AttributeValue> SHALL be specified by using the
3044 DataType attribute of the parent <Attribute> element.

6.9. Element <Response>

3046 The <Response> element is a top-level element in the XACML *context* schema. The
3047 <Response> element is an abstraction layer used by the *policy* language. Any proprietary
3048 system using the XACML specification MUST transform an XACML *context* <Response> element
3049 into the form of its *authorization decision*.

3050 The <Response> element encapsulates the *authorization decision* produced by the *PDP*. It includes
3051 a sequence of one or more results, with one <Result> element per requested *resource*. Multiple
3052 results MAY be returned by some implementations, in particular those that support the XACML
3053 Profile for Requests for Multiple Resources [MULT]. Support for multiple results is OPTIONAL.

```
3054 <xs:element name="Response" type="xacml-context:ResponseType"/>
3055 <xs:complexType name="ResponseType">
3056   <xs:sequence>
3057     <xs:element ref="xacml-context:Result" maxOccurs="unbounded"/>
3058   </xs:sequence>
3059 </xs:complexType>
```

3060 The <Response> element is of **ResponseType** complex type.

3061 The <Response> element contains the following elements:

3062 <Result> [One to Many]

3063 An authorization decision result. See Section 6.10.

6.10. Element <Result>

3065 The <Result> element represents an *authorization decision* result for the *resource* specified by
3066 the ResourceId *attribute*. It MAY include a set of *obligations* that MUST be fulfilled by the *PEP*.
3067 If the *PEP* does not understand or cannot fulfill an *obligation*, then it MUST act as if the *PDP* had
3068 denied *access* to the requested *resource*.

3069

```
3070 <xs:complexType name="ResultType">
3071   <xs:sequence>
3072     <xs:element ref="xacml-context:Decision"/>
3073     <xs:element ref="xacml-context:Status" minOccurs="0"/>
3074     <xs:element ref="xacml:Obligations" minOccurs="0"/>
3075   </xs:sequence>
```

```
3076 <xs:attribute name="ResourceId" type="xs:string" use="optional"/>
3077 </xs:complexType>
```

3078 The <Result> element is of **ResultType** complex type.

3079 The <Result> element contains the following attributes and elements:

3080 ResourceId [Optional]

3081 The identifier of the requested **resource**. If this attribute is omitted, then the **resource**
3082 identity is that specified by the “urn:oasis:names:tc:xacml:1.0:resource:resource-id”
3083 **resource attribute** in the corresponding <Request> element.

3084 <Decision> [Required]

3085 The **authorization decision**: “Permit”, “Deny”, “Indeterminate” or “NotApplicable”.

3086 <Status> [Optional]

3087 Indicates whether errors occurred during evaluation of the **decision request**, and
3088 optionally, information about those errors. If the <Response> element contains <Result>
3089 elements whose <Status> elements are all identical, and the <Response> element is
3090 contained in a protocol wrapper that can convey status information, then the common
3091 status information MAY be placed in the protocol wrapper and this <Status> element
3092 MAY be omitted from all <Result> elements.

3093 <Obligations> [Optional]

3094 A list of **obligations** that MUST be fulfilled by the **PEP**. If the **PEP** does not understand or
3095 cannot fulfill an **obligation**, then it MUST act as if the **PDP** had denied **access** to the
3096 requested **resource**. See Section 7.14 for a description of how the set of **obligations** to
3097 be returned by the PDP is determined.

3098 6.11. Element <Decision>

3099 The <Decision> element contains the result of **policy** evaluation.

```
3100 <xs:element name="Decision" type="xacml-context:DecisionType"/>
3101 <xs:simpleType name="DecisionType">
3102   <xs:restriction base="xs:string">
3103     <xs:enumeration value="Permit"/>
3104     <xs:enumeration value="Deny"/>
3105     <xs:enumeration value="Indeterminate"/>
3106     <xs:enumeration value="NotApplicable"/>
3107   </xs:restriction>
3108 </xs:simpleType>
```

3109 The <Decision> element is of **DecisionType** simple type.

3110 The values of the <Decision> element have the following meanings:

3111 “Permit”: the requested **access** is permitted.

3112 “Deny”: the requested **access** is denied.

3113 “Indeterminate”: the PDP is unable to evaluate the requested **access**. Reasons for such
3114 inability include: missing **attributes**, network errors while retrieving **policies**, division by
3115 zero during **policy** evaluation, syntax errors in the **decision request** or in the **policy**, etc..

3116 “NotApplicable”: the **PDP** does not have any **policy** that applies to this **decision request**.

3117

6.12. Element <Status>

3118 The <Status> element represents the status of the *authorization decision* result.

```
3119 <xs:element name="Status" type="xacml-context:StatusType" />
3120 <xs:complexType name="StatusType">
3121   <xs:sequence>
3122     <xs:element ref="xacml-context:StatusCode" />
3123     <xs:element ref="xacml-context:StatusMessage" minOccurs="0" />
3124     <xs:element ref="xacml-context:StatusDetail" minOccurs="0" />
3125   </xs:sequence>
3126 </xs:complexType>
```

3127 The <Status> element is of **StatusType** complex type.

3128 The <Status> element contains the following elements:

3129 <StatusCode> [Required]

3130 Status code.

3131 <StatusMessage> [Optional]

3132 A status message describing the status code.

3133 <StatusDetail> [Optional]

3134 Additional status information.

6.13. Element <StatusCode>

3136 The <StatusCode> element contains a major status code value and an optional sequence of
3137 minor status codes.

```
3138 <xs:element name="StatusCode" type="xacml-context:StatusCodeType" />
3139 <xs:complexType name="StatusCodeType">
3140   <xs:sequence>
3141     <xs:element ref="xacml-context:StatusCode" minOccurs="0" />
3142   </xs:sequence>
3143   <xs:attribute name="Value" type="xs:anyURI" use="required" />
3144 </xs:complexType>
```

3145 The <StatusCode> element is of **StatusCodeType** complex type.

3146 The <StatusCode> element contains the following attributes and elements:

3147 Value [Required]

3148 See Section B.9 for a list of values.

3149 <StatusCode> [Any Number]

3150 Minor status code. This status code qualifies its parent status code.

6.14. Element <StatusMessage>

3152 The <StatusMessage> element is a free-form description of the status code.

```
3153 <xs:element name="StatusMessage" type="xs:string" />
```

3154 The <StatusMessage> element is of **xs:string** type.

3155

6.15. Element <StatusDetail>

3156 The <StatusDetail> element qualifies the <Status> element with additional information.

```
3157 <xs:element name="StatusDetail" type="xacml-context:StatusDetailType"/>
3158 <xs:complexType name="StatusDetailType">
3159   <xs:sequence>
3160     <xs:any namespace="##any" processContents="lax" minOccurs="0"
3161     maxOccurs="unbounded"/>
3162   </xs:sequence>
3163 </xs:complexType>
```

3164 The <StatusDetail> element is of **StatusDetailType** complex type.

3165 The <StatusDetail> element allows arbitrary XML content.

3166 Inclusion of a <StatusDetail> element is optional. However, if a **PDP** returns one of the
3167 following XACML-defined <StatusCode> values and includes a <StatusDetail> element, then
3168 the following rules apply.

3169 urn:oasis:names:tc:xacml:1.0:status:ok

3170 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “ok” status value.

3171 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

3172 A **PDP** MAY choose not to return any <StatusDetail> information or MAY choose to return a
3173 <StatusDetail> element containing one or more <xacml-context:
3174 MissingAttributeDetail> elements.

3175 urn:oasis:names:tc:xacml:1.0:status:syntax-error

3176 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “syntax-error” status
3177 value. A syntax error may represent either a problem with the **policy** being used or with the
3178 request **context**. The **PDP** MAY return a <StatusMessage> describing the problem.

3179 urn:oasis:names:tc:xacml:1.0:status:processing-error

3180 A **PDP** MUST NOT return <StatusDetail> element in conjunction with the “processing-error”
3181 status value. This status code indicates an internal problem in the **PDP**. For security reasons, the
3182 **PDP** MAY choose to return no further information to the **PEP**. In the case of a divide-by-zero error
3183 or other computational error, the **PDP** MAY return a <StatusMessage> describing the nature of
3184 the error.

3185 6.16. Element <MissingAttributeDetail>

3186 The <MissingAttributeDetail> element conveys information about **attributes** required for
3187 **policy** evaluation that were missing from the request **context**.

```
3188 <xs:element name="MissingAttributeDetail" type="xacml-
3189 context:MissingAttributeDetailType"/>
3190 <xs:complexType name="MissingAttributeDetailType">
3191   <xs:sequence>
3192     <xs:element ref="xacml-context:AttributeValue" minOccurs="0"
3193     maxOccurs="unbounded"/>
3194   </xs:sequence>
3195   <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
3196   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
3197   <xs:attribute name="Issuer" type="xs:string" use="optional"/>
3198 </xs:complexType>
```

3199 The <MissingAttributeDetail> element is of **MissingAttributeDetailType** complex type.

3200 The <MissingAttributeDetail> element contains the following attributes and elements:

3201 AttributeValue [Optional]

3202 The required value of the missing **attribute**.

3203 <AttributeId> [Required]

3204 The identifier of the missing **attribute**.

3205 <DataType> [Required]

3206 The data-type of the missing **attribute**.

3207 Issuer [Optional]

3208 This attribute, if supplied, SHALL specify the required **Issuer** of the missing **attribute**.

3209 If the PDP includes <xacml-context:AttributeValue> elements in the <MissingAttributeDetail>

3210 element, then this indicates the acceptable values for that attribute. If no <xacml-

3211 context:AttributeValue> elements are included, then this indicates the names of attributes that the

3212 PDP failed to resolve during its evaluation. The list of attributes may be partial or complete. There

3213 is no guarantee by the PDP that supplying the missing values or attributes will be sufficient to

3214 satisfy the policy.

3215 7. Functional requirements (normative)

3216 This section specifies certain functional requirements that are not directly associated with the

3217 production or consumption of a particular XACML element.

3218 7.1. Policy enforcement point

3219 This section describes the requirements for the **PEP**.

3220 An application functions in the role of the **PEP** if it guards access to a set of **resources** and asks

3221 the **PDP** for an **authorization decision**. The **PEP** MUST abide by the **authorization decision** as

3222 described in one of the following sub-sections

3223 7.1.1. Base PEP

3224 If the **decision** is "Permit", then the **PEP** SHALL permit access. If **obligations** accompany the

3225 **decision**, then the **PEP** SHALL permit **access** only if it understands and it can and will discharge

3226 those **obligations**.

3227 If the **decision** is "Deny", then the **PEP** SHALL deny **access**. If **obligations** accompany the

3228 **decision**, then the **PEP** shall deny **access** only if it understands, and it can and will discharge

3229 those **obligations**.

3230 If the **decision** is "Not Applicable", then the **PEP's** behavior is undefined.

3231 If the **decision** is "Indeterminate", then the **PEP's** behavior is undefined.

3232

7.1.2. Deny-biased PEP

3233 If the **decision** is "Permit", then the **PEP** SHALL permit **access**. If **obligations** accompany the
3234 **decision**, then the **PEP** SHALL permit **access** only if it understands and it can and will discharge
3235 those **obligations**.

3236 All other **decisions** SHALL result in the denial of **access**.

3237 Note: other actions, e.g. consultation of additional **PDPs**, reformulation/resubmission of the
3238 **decision request**, etc., are not prohibited.

3239

7.1.3. Permit-biased PEP

3240 If the **decision** is "Deny", then the **PEP** SHALL deny **access**. If **obligations** accompany the
3241 **decision**, then the **PEP** shall deny **access** only if it understands, and it can and will discharge
3242 those **obligations**.

3243 All other **decisions** SHALL result in the permission of **access**.

3244 Note: other actions, e.g. consultation of additional **PDPs**, reformulation/resubmission of the
3245 **decision request**, etc., are not prohibited.

3246

7.2. Attribute evaluation

3247 **Attributes** are represented in the request **context** by the **context handler**, regardless of whether
3248 or not they appeared in the original **decision request**, and are referred to in the **policy** by **subject**,
3249 **resource**, **action** and **environment attribute** designators and **attribute** selectors. A **named**
3250 **attribute** is the term used for the criteria that the specific **subject**, **resource**, **action** and
3251 **environment attribute** designators and selectors use to refer to particular **attributes** in the
3252 **subject**, **resource**, **action** and **environment** elements of the request **context**, respectively.

3253

7.2.1. Structured attributes

3254 <xacml:AttributeValue> and <xacml-context:AttributeValue> elements MAY contain
3255 an instance of a structured XML data-type, for example <ds:KeyInfo>. XACML 2.0 supports
3256 several ways for comparing the contents of such elements.

3257 1. In some cases, such elements MAY be compared using one of the XACML string functions,
3258 such as "string-regexp-match", described below. This requires that the element be given
3259 the data-type "<http://www.w3.org/2001/XMLSchema#string>". For example, a structured
3260 data-type that is actually a ds:KeyInfo/KeyName would appear in the Context as:

```
3261 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3262   <ds:KeyName>jhibbert-key</ds:KeyName>  
3263 </AttributeValue>
```

3264 In general, this method will not be adequate unless the structured data-type is quite simple.

3265 2. An <AttributeSelector> element MAY be used to select the contents of a leaf sub-
3266 element of the structured data-type by means of an XPath expression. That value MAY
3267 then be compared using one of the supported XACML functions appropriate for its primitive
3268 data-type. This method requires support by the **PDP** for the optional XPath expressions
3269 feature.

3270 3. An <AttributeSelector> element MAY be used to select any node in the structured
3271 data-type by means of an XPath expression. This node MAY then be compared using one

3272 of the XPath-based functions described in Section A.3. This method requires support by
3273 the **PDP** for the optional XPath expressions and XPath functions features.

3274 See also Section 8.2.

3275 7.2.2. Attribute bags

3276 XACML defines implicit collections of its data-types. XACML refers to a collection of values that are
3277 of a single data-type as a **bag**. **Bags** of data-types are needed because selections of nodes from
3278 an XML **resource** or XACML request **context** may return more than one value.

3279 The <AttributeSelector> element uses an XPath expression to specify the selection of data
3280 from an XML **resource**. The result of an XPath expression is termed a *node-set*, which contains all
3281 the leaf nodes from the XML **resource** that match the predicate in the XPath expression. Based on
3282 the various indexing functions provided in the XPath specification, it SHALL be implied that a
3283 resultant node-set is the collection of the matching nodes. XACML also defines the
3284 <AttributeDesignator> **element** to have the same matching methodology for **attributes** in the
3285 XACML request **context**.

3286 The values in a **bag** are not ordered, and some of the values may be duplicates. There SHALL be
3287 no notion of a **bag** containing **bags**, or a **bag** containing values of differing types. I.e. a **bag** in
3288 XACML SHALL contain only values that are of the same data-type.

3289 7.2.3. Multivalued attributes

3290 If a single <Attribute> element in a request **context** contains multiple <xacml-
3291 context:AttributeValue> child elements, then the **bag** of values resulting from evaluation of
3292 the <Attribute> element MUST be identical to the **bag** of values that results from evaluating a
3293 **context** in which each <xacml-context:AttributeValue> element appears in a separate
3294 <Attribute> element, each carrying identical meta-data.

3295 7.2.4. Attribute Matching

3296 A **named attribute** includes specific criteria with which to match **attributes** in the **context**. An
3297 **attribute** specifies an **AttributeId** and **DataType**, and a **named attribute** also specifies the
3298 **Issuer**. A **named attribute** SHALL match an **attribute** if the values of their respective
3299 **AttributeId**, **DataType** and optional **Issuer** attributes match within their particular element -
3300 **subject**, **resource**, **action** or **environment** - of the **context**. The **AttributeId** of the **named**
3301 **attribute** MUST match, by URI equality, the **AttributeId** of the corresponding **context attribute**.
3302 The **DataType** of the **named attribute** MUST match, by URI equality, the **DataType** of the
3303 corresponding **context attribute**. If **Issuer** is supplied in the **named attribute**, then it MUST
3304 match, using the `urn:oasis:names:tc:xacml:1.0:function:string-equal` function, the
3305 **Issuer** of the corresponding **context attribute**. If **Issuer** is not supplied in the **named attribute**,
3306 then the matching of the **context attribute** to the **named attribute** SHALL be governed by
3307 **AttributeId** and **DataType** alone, regardless of the presence, absence, or actual value of
3308 **Issuer** in the corresponding **context attribute**. In the case of an **attribute** selector, the matching
3309 of the **attribute** to the **named attribute** SHALL be governed by the XPath expression and
3310 **DataType**.

3311

7.2.5. Attribute Retrieval

3312 The **PDP** SHALL request the values of **attributes** in the request **context** from the **context handler**.
3313 The **PDP** SHALL reference the **attributes** as if they were in a physical request **context** document,
3314 but the **context handler** is responsible for obtaining and supplying the requested values by
3315 whatever means it deems appropriate. The **context handler** SHALL return the values of
3316 **attributes** that match the **attribute** designator or **attribute** selector and form them into a **bag** of
3317 values with the specified data-type. If no **attributes** from the request **context** match, then the
3318 **attribute** SHALL be considered missing. If the **attribute** is missing, then `MustBePresent`
3319 governs whether the **attribute** designator or **attribute** selector returns an empty **bag** or an
3320 "Indeterminate" result. If `MustBePresent` is "False" (default value), then a missing **attribute**
3321 SHALL result in an empty **bag**. If `MustBePresent` is "True", then a missing **attribute** SHALL
3322 result in "Indeterminate". This "Indeterminate" result SHALL be handled in accordance with the
3323 specification of the encompassing expressions, **rules**, **policies** and **policy sets**. If the result is
3324 "Indeterminate", then the `AttributeId`, `Data Type` and `Issuer` of the **attribute** MAY be listed in
3325 the **authorization decision** as described in Section 7.13. However, a **PDP** MAY choose not to
3326 return such information for security reasons.

3327

7.2.6. Environment Attributes

3328 Standard **environment attributes** are listed in Section B.8. If a value for one of these **attributes** is
3329 supplied in the **decision request**, then the **context handler** SHALL use that value. Otherwise, the
3330 **context handler** SHALL supply a value. In the case of date and time **attributes**, the supplied
3331 value SHALL have the semantics of the "date and time that apply to the **decision request**".

3332

7.3. Expression evaluation

3333 XACML specifies expressions in terms of the elements listed below, of which the `<Apply>` and
3334 `<Condition>` elements recursively compose greater expressions. Valid expressions SHALL be
3335 type correct, which means that the types of each of the elements contained within `<Apply>` and
3336 `<Condition>` elements SHALL agree with the respective argument types of the function that is
3337 named by the `FunctionId` attribute. The resultant type of the `<Apply>` or `<Condition>`
3338 element SHALL be the resultant type of the function, which MAY be narrowed to a primitive data-
3339 type, or a **bag** of a primitive data-type, by type-unification. XACML defines an evaluation result of
3340 "Indeterminate", which is said to be the result of an invalid expression, or an operational error
3341 occurring during the evaluation of the expression.

3342 XACML defines these elements to be in the substitution group of the `<Expression>` element:

- 3343 • `<xacml:AttributeValue>`
- 3344 • `<xacml:SubjectAttributeDesignator>`
- 3345 • `<xacml:ResourceAttributeDesignator>`
- 3346 • `<xacml:ActionAttributeDesignator>`
- 3347 • `<xacml:EnvironmentAttributeDesignator>`
- 3348 • `<xacml:AttributeSelector>`
- 3349 • `<xacml:Apply>`
- 3350 • `<xacml:Condition>`

- 3351 • <xacml:Function>
- 3352 • <xacml:VariableReference>

3353 7.4. Arithmetic evaluation

3354 IEEE 754 [IEEE 754] specifies how to evaluate arithmetic functions in a context, which specifies
3355 defaults for precision, rounding, etc. XACML SHALL use this specification for the evaluation of all
3356 integer and double functions relying on the *Extended Default Context*, enhanced with double
3357 precision:

3358 flags - all set to 0

3359 trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the “division-by-zero” trap
3360 enabler, which SHALL be set to 1

3361 precision - is set to the designated double precision

3362 rounding - is set to round-half-even (IEEE 854 §4.1)

3363 7.5. Match evaluation

3364 **Attribute** matching elements appear in the <Target> element of **rules**, **policies** and **policy sets**.
3365 They are the following:

3366 <SubjectMatch>

3367 <ResourceMatch>

3368 <ActionMatch>

3369 <EnvironmentMatch>

3370 These elements represent Boolean expressions over **attributes** of the **subject**, **resource**, **action**
3371 and **environment**, respectively. A matching element contains a MatchId attribute that specifies
3372 the function to be used in performing the match evaluation, an <xacml:AttributeValue> and an
3373 <AttributeDesignator> or <AttributeSelector> element that specifies the **attribute** in the
3374 **context** that is to be matched against the specified value.

3375 The MatchId attribute SHALL specify a function that compares two arguments, returning a result
3376 type of "http://www.w3.org/2001/XMLSchema#boolean". The **attribute** value specified in the
3377 matching element SHALL be supplied to the MatchId function as its first argument. An element of
3378 the **bag** returned by the <AttributeDesignator> or <AttributeSelector> element SHALL
3379 be supplied to the MatchId function as its second argument, as explained below. The DataType
3380 of the <xacml:AttributeValue> SHALL match the data-type of the first argument expected by
3381 the MatchId function. The DataType of the <AttributeDesignator> or
3382 <AttributeSelector> element SHALL match the data-type of the second argument expected
3383 by the MatchId function.

3384 The XACML standard functions that meet the requirements for use as a MatchId attribute value
3385 are:

3386 urn:oasis:names:tc:xacml:2.0:function:-type-equal

3387 urn:oasis:names:tc:xacml:2.0:function:-type-greater-than

3388 urn:oasis:names:tc:xacml:2.0:function:-type-greater-than-or-equal

3389 urn:oasis:names:tc:xacml:2.0:function:-type-less-than
3390 urn:oasis:names:tc:xacml:2.0:function:-type-less-than-or-equal
3391 urn:oasis:names:tc:xacml:2.0:function:-type-match

3392 In addition, functions that are strictly within an extension to XACML MAY appear as a value for the
3393 MatchId attribute, and those functions MAY use data-types that are also extensions, so long as
3394 the extension function returns a Boolean result and takes two single base types as its inputs. The
3395 function used as the value for the MatchId attribute SHOULD be easily indexable. Use of non-
3396 indexable or complex functions may prevent efficient evaluation of **decision requests**.

3397 The evaluation semantics for a matching element is as follows. If an operational error were to
3398 occur while evaluating the <AttributeDesignator> or <AttributeSelector> element, then
3399 the result of the entire expression SHALL be "Indeterminate". If the <AttributeDesignator> or
3400 <AttributeSelector> element were to evaluate to an empty **bag**, then the result of the
3401 expression SHALL be "False". Otherwise, the MatchId function SHALL be applied between the
3402 <xacml:AttributeValue> and each element of the **bag** returned from the
3403 <AttributeDesignator> or <AttributeSelector> element. If at least one of those function
3404 applications were to evaluate to "True", then the result of the entire expression SHALL be "True".
3405 Otherwise, if at least one of the function applications results in "Indeterminate", then the result
3406 SHALL be "Indeterminate". Finally, if all function applications evaluate to "False", then the result of
3407 the entire expression SHALL be "False".

3408 It is also possible to express the semantics of a **target** matching element in a **condition**. For
3409 instance, the **target** match expression that compares a "subject-name" starting with the name
3410 "John" can be expressed as follows:

```
3411 <SubjectMatch  
3412 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">  
3413   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3414     John.*  
3415   </AttributeValue>  
3416   <SubjectAttributeDesignator  
3417     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3418     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3419 </SubjectMatch>
```

3420 Alternatively, the same match semantics can be expressed as an <Apply> element in a **condition**
3421 by using the "urn:oasis:names:tc:xacml:1.0:function:any-of" function, as follows:

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

3432 7.6. Target evaluation

3433 The **target** value SHALL be "Match" if the **subjects**, **resources**, **actions** and **environments**
3434 specified in the **target** all match values in the request **context**. If any one of the **subjects**,
3435 **resources**, **actions** and **environments** specified in the **target** are "Indeterminate", then the **target**
3436 SHALL be "Indeterminate". Otherwise, the **target** SHALL be "No match". The **target** match table is
3437 shown in Table 1.
3438

Subjects value	Resources value	Actions value	Environments value	Target value
"Match"	"Match"	"Match"	"Match"	"Match"
"No match"	"Match" or "No match"	"Match" or "No match"	"Match" or "No match"	"No match"
"Match" or "No match"	"No match"	"Match" or "No match"	"Match" or "No match"	"No match"
"Match" or "No match"	"Match" or "No match"	"No match"	"Match" or "No match"	"No match"
"Match" or "No match"	"Match" or "No match"	"Match" or "No match"	"No match"	"No match"
"Indeterminate"	Don't care	Don't care	Don't care	"Indeterminate"
Don't care	"Indeterminate"	Don't care	Don't care	"Indeterminate"
Don't care	Don't care	"Indeterminate"	Don't care	"Indeterminate"
Don't care	Don't care	Don't care	"Indeterminate"	"Indeterminate"

3439

Table 1 - Target match table

3440 The **subjects**, **resources**, **actions** and **environments** SHALL match values in the request **context**
 3441 if at least one of their <Subject>, <Resource>, <Action> or <Environment> elements,
 3442 respectively, matches a value in the request **context**. The **subjects** match table is shown in Table
 3443 2. The **resources**, **actions** and **environments** match tables are analogous.

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

3444

Table 2 - Subjects match table

3445 A **subject**, **resource**, **action** or **environment** SHALL match a value in the request **context** if the
 3446 value of all its <SubjectMatch>, <ResourceMatch>, <ActionMatch> or
 3447 <EnvironmentMatch> elements, respectively, are "True".

3448 The **subject** match table is shown in Table 3. The **resource**, **action** and **environment** match
 3449 tables are analogous.

3450

3451

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

3452

Table 3 - Subject match table

3453 7.7. VariableReference Evaluation

3454 The <VariableReference> element references a single <VariableDefinition> element
3455 contained within the same <Policy> element. A <VariableReference> that does not
3456 reference a particular <VariableDefinition> element within the encompassing <Policy>
3457 element is called an undefined reference. **Policies** with undefined references are invalid.

3458 In any place where a <VariableReference> occurs, it has the effect as if the text of the
3459 <Expression> element defined in the <VariableDefinition> element replaces the
3460 <VariableReference> element. Any evaluation scheme that preserves this semantic is
3461 acceptable. For instance, the expression in the <VariableDefinition> element may be
3462 evaluated to a particular value and cached for multiple references without consequence. (I.e. the
3463 value of an <Expression> element remains the same for the entire policy evaluation.) This
3464 characteristic is one of the benefits of XACML being a declarative language.

3465 7.8. Condition evaluation

3466 The **condition** value SHALL be "True" if the <Condition> element is absent, or if it evaluates to
3467 "True". Its value SHALL be "False" if the <Condition> element evaluates to "False". The
3468 **condition** value SHALL be "Indeterminate", if the expression contained in the <Condition>
3469 element evaluates to "Indeterminate."

3470 7.9. Rule evaluation

3471 A **rule** has a value that can be calculated by evaluating its contents. **Rule** evaluation involves
3472 separate evaluation of the **rule's target** and **condition**. The **rule** truth table is shown in Table 4.

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

3473 Table 4 - Rule truth table

3474 If the **target** value is "No-match" or "Indeterminate" then the **rule** value SHALL be "NotApplicable"
3475 or "Indeterminate", respectively, regardless of the value of the **condition**. For these cases,
3476 therefore, the **condition** need not be evaluated.

3477 If the **target** value is "Match" and the **condition** value is "True", then the **effect** specified in the
3478 enclosing <Rule> element SHALL determine the **rule's** value.

3479

7.10. Policy evaluation

3480
3481
3482

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

3483
3484
3485
3486
3487

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

3488

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

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

3489

Table 5 - Policy truth table

3490
3491
3492
3493
3494
3495
3496

A **rules** value of "At least one rule value is its Effect" means either that the <Rule> element is absent, or one or more of the **rules** contained in the **policy** is applicable to the **decision request** (i.e., it returns the value of its "Effect"; see Section 7.9). A **rules** value of "All rule values are 'NotApplicable'" SHALL be used if no **rule** contained in the **policy** is applicable to the request and if no **rule** contained in the **policy** returns a value of "Indeterminate". If no **rule** contained in the **policy** is applicable to the request, but one or more **rule** returns a value of "Indeterminate", then the **rules** SHALL evaluate to "At least one rule value is 'Indeterminate'".

3497
3498
3499

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

3500
3501
3502

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

3503
3504
3505
3506
3507
3508
3509

Note that none of the **rule-combining algorithms** defined by XACML 2.0 take parameters. However, non-standard **combining algorithms** MAY take parameters. In such a case, the values of these parameters associated with the **rules**, MUST be taken into account when evaluating the **policy**. The parameters and their types should be defined in the specification of the **combining algorithm**. If the implementation supports combiner parameters and if combiner parameters are present in a **policy**, then the parameter values MUST be supplied to the **combining algorithm** implementation.

3510

7.11. Policy Set evaluation

3511
3512
3513
3514

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

3515
3516
3517
3518
3519

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

3520

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

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

3521

Table 6 – Policy set truth table

3522
3523
3524
3525
3526
3527
3528
3529
3530
3531

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

3532
3533
3534

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

3535
3536
3537

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

3538
3539
3540
3541

Note that none of the **policy-combining algorithms** defined by XACML 2.0 take parameters. However, non-standard **combining algorithms** MAY take parameters. In such a case, the values of these parameters associated with the **policies**, MUST be taken into account when evaluating the **policy set**. The parameters and their types should be defined in the specification of the

3542 **combining algorithm.** If the implementation supports combiner parameters and if combiner
3543 parameters are present in a **policy**, then the parameter values MUST be supplied to the
3544 **combining algorithm** implementation.

3545 7.12. Hierarchical resources

3546 It is often the case that a **resource** is organized as a hierarchy (e.g. file system, XML document).
3547 XACML provides several optional mechanisms for supporting hierarchical resources. These are
3548 described in the XACML Profile for Hierarchical Resources [HIER] and in the XACML Profile for
3549 Requests for Multiple Resources [MULT].

3550 7.13. Authorization decision

3551 In relation to a particular **decision request**, the **PDP** is defined by a **policy-combining algorithm**
3552 and a set of **policies** and/or **policy sets**. The **PDP** SHALL return a response **context** as if it had
3553 evaluated a single **policy set** consisting of this **policy-combining algorithm** and the set of
3554 **policies** and/or **policy sets**.

3555 The **PDP** MUST evaluate the **policy set** as specified in Sections 5 and 7. The **PDP** MUST return a
3556 response **context**, with one <Decision> element of value "Permit", "Deny", "Indeterminate" or
3557 "NotApplicable".

3558 If the **PDP** cannot make a decision, then an "Indeterminate" <Decision> element SHALL be
3559 returned.

3560 7.14. Obligations

3561 A **policy** or **policy set** may contain one or more **obligations**. When such a **policy** or **policy set** is
3562 evaluated, an **obligation** SHALL be passed up to the next level of evaluation (the enclosing or
3563 referencing **policy**, **policy set** or **authorization decision**) only if the **effect** of the **policy** or **policy**
3564 **set** being evaluated matches the value of the FulfillOn attribute of the **obligation**.

3565
3566 As a consequence of this procedure, no **obligations** SHALL be returned to the **PEP** if the **policies**
3567 or **policy sets** from which they are drawn are not evaluated, or if their evaluated result is
3568 "Indeterminate" or "NotApplicable", or if the **decision** resulting from evaluating the **policy** or **policy**
3569 **set** does not match the **decision** resulting from evaluating an enclosing **policy set**.

3570
3571 If the **PDP's** evaluation is viewed as a tree of **policy sets** and **policies**, each of which returns
3572 "Permit" or "Deny", then the set of **obligations** returned by the **PDP** to the **PEP** will include only the
3573 **obligations** associated with those paths where the **effect** at each level of evaluation is the same as
3574 the **effect** being returned by the **PDP**. In situations where any lack of determinism is unacceptable,
3575 a deterministic combining algorithm, such as ordered-deny-overrides, should be used.

3576 Also, see Section 7.1.

3577 7.15. Exception handling

3578 XACML specifies behaviour for the **PDP** in the following situations.

3579 7.15.1. Unsupported functionality

3580 If the **PDP** attempts to evaluate a **policy set** or **policy** that contains an optional element type or
3581 function that the **PDP** does not support, then the **PDP** SHALL return a <Decision> value of

3582 "Indeterminate". If a <StatusCode> element is also returned, then its value SHALL be
3583 "urn:oasis:names:tc:xacml:1.0:status:syntax-error" in the case of an unsupported element type, and
3584 "urn:oasis:names:tc:xacml:1.0:status:processing-error" in the case of an unsupported function.

3585 **7.15.2. Syntax and type errors**

3586 If a **policy** that contains invalid syntax is evaluated by the XACML **PDP** at the time a **decision**
3587 **request** is received, then the result of that **policy** SHALL be "Indeterminate" with a StatusCode
3588 value of "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

3589 If a **policy** that contains invalid static data-types is evaluated by the XACML **PDP** at the time a
3590 **decision request** is received, then the result of that **policy** SHALL be "Indeterminate" with a
3591 StatusCode value of "urn:oasis:names:tc:xacml:1.0:status:processing-error".

3592 **7.15.3. Missing attributes**

3593 The absence of matching **attributes** in the request **context** for any of the **attribute** designators or
3594 selectors that are found in the **policy** SHALL result in a <Decision> element containing the
3595 "Indeterminate" value, as described in Sections 5.37 and 5.42. If, in this case, and a status code is
3596 supplied, then the value

3597 "urn:oasis:names:tc:xacml:1.0:status:missing-attribute"

3598 SHALL be used, to indicate that more information is needed in order for a definitive decision to be
3599 rendered. In this case, the <Status> element MAY list the names and data-types of any
3600 **attributes** of the **subjects**, **resource**, **action** or **environment** that are needed by the **PDP** to refine
3601 its decision (see Section 6.16). A **PEP** MAY resubmit a refined request **context** in response to a
3602 <Decision> element contents of "Indeterminate" with a status code of

3603 "urn:oasis:names:tc:xacml:1.0:missing-attribute"

3604 by adding **attribute** values for the **attribute** names that were listed in the previous response. When
3605 the **PDP** returns a <Decision> element contents of "Indeterminate", with a status code of

3606 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3607 it MUST NOT list the names and data-types of any **attribute** of the **subject**, **resource**, **action** or
3608 **environment** for which values were supplied in the original request. Note, this requirement forces
3609 the **PDP** to eventually return an **authorization decision** of "Permit", "Deny" or "Indeterminate" with
3610 some other status code, in response to successively-refined requests.

3611 **8. XACML extensibility points (non-normative)**

3612 This section describes the points within the XACML model and schema where extensions can be
3613 added

3614 **8.1. Extensible XML attribute types**

3615 The following XML attributes have values that are URIs. These may be extended by the creation of
3616 new URIs associated with new semantics for these attributes.

3617 AttributeId,

3618 DataType,
3619 FunctionId,
3620 MatchId,
3621 ObligationId,
3622 PolicyCombiningAlgId,
3623 RuleCombiningAlgId,
3624 StatusCode,
3625 SubjectCategory.
3626 See Section 5 for definitions of these attribute types.

3627 **8.2. Structured attributes**

3628 <xacml:AttributeValue> and <xacml-context:AttributeValue> elements MAY
3629 contain an instance of a structured XML data-type. Section 7.2.1 describes a number of standard
3630 techniques to identify data items within such a structured attribute. Listed here are some additional
3631 techniques that require XACML extensions.

- 3632 1. For a given structured data-type, a community of XACML users MAY define new attribute
3633 identifiers for each leaf sub-element of the structured data-type that has a type conformant
3634 with one of the XACML-defined primitive data-types. Using these new attribute identifiers,
3635 the **PEPs** or **context handlers** used by that community of users can flatten instances of
3636 the structured data-type into a sequence of individual <Attribute> elements. Each such
3637 <Attribute> element can be compared using the XACML-defined functions. Using this
3638 method, the structured data-type itself never appears in an <xacml-
3639 context:AttributeValue> element.
- 3640 2. A community of XACML users MAY define a new function that can be used to compare a
3641 value of the structured data-type against some other value. This method may only be used
3642 by **PDPs** that support the new function.

3643 **9. Security and privacy considerations (non- 3644 normative)**

3645 This section identifies possible security and privacy compromise scenarios that should be
3646 considered when implementing an XACML-based system. The section is informative only. It is left
3647 to the implementer to decide whether these compromise scenarios are practical in their
3648 environment and to select appropriate safeguards.

3649 **9.1. Threat model**

3650 We assume here that the adversary has access to the communication channel between the
3651 XACML actors and is able to interpret, insert, delete and modify messages or parts of messages.

3652 Additionally, an actor may use information from a former message maliciously in subsequent
3653 transactions. It is further assumed that **rules** and **policies** are only as reliable as the actors that
3654 create and use them. Thus it is incumbent on each actor to establish appropriate trust in the other

3655 actors upon which it relies. Mechanisms for trust establishment are outside the scope of this
3656 specification.

3657 The messages that are transmitted between the actors in the XACML model are susceptible to
3658 attack by malicious third parties. Other points of vulnerability include the **PEP**, the **PDP** and the
3659 **PAP**. While some of these entities are not strictly within the scope of this specification, their
3660 compromise could lead to the compromise of **access control** enforced by the **PEP**.

3661 It should be noted that there are other components of a distributed system that may be
3662 compromised, such as an operating system and the domain-name system (DNS) that are outside
3663 the scope of this discussion of threat models. Compromise in these components may also lead to a
3664 policy violation.

3665 The following sections detail specific compromise scenarios that may be relevant to an XACML
3666 system.

3667 **9.1.1. Unauthorized disclosure**

3668 XACML does not specify any inherent mechanisms to protect the confidentiality of the messages
3669 exchanged between actors. Therefore, an adversary could observe the messages in transit. Under
3670 certain security policies, disclosure of this information is a violation. Disclosure of **attributes** or the
3671 types of **decision requests** that a **subject** submits may be a breach of privacy policy. In the
3672 commercial sector, the consequences of unauthorized disclosure of personal data may range from
3673 embarrassment to the custodian to imprisonment and large fines in the case of medical or financial
3674 data.

3675 Unauthorized disclosure is addressed by confidentiality safeguards.

3676 **9.1.2. Message replay**

3677 A message replay attack is one in which the adversary records and replays legitimate messages
3678 between XACML actors. This attack may lead to denial of service, the use of out-of-date
3679 information or impersonation.

3680 Prevention of replay attacks requires the use of message freshness safeguards.

3681 Note that encryption of the message does not mitigate a replay attack since the message is simply
3682 replayed and does not have to be understood by the adversary.

3683 **9.1.3. Message insertion**

3684 A message insertion attack is one in which the adversary inserts messages in the sequence of
3685 messages between XACML actors.

3686 The solution to a message insertion attack is to use mutual authentication and message sequence
3687 integrity safeguards between the actors. It should be noted that just using SSL mutual
3688 authentication is not sufficient. This only proves that the other party is the one identified by the
3689 subject of the X.509 certificate. In order to be effective, it is necessary to confirm that the certificate
3690 subject is authorized to send the message.

3691 **9.1.4. Message deletion**

3692 A message deletion attack is one in which the adversary deletes messages in the sequence of
3693 messages between XACML actors. Message deletion may lead to denial of service. However, a

3694 properly designed XACML system should not render an incorrect authorization decision as a result
3695 of a message deletion attack.

3696 The solution to a message deletion attack is to use message sequence integrity safeguards
3697 between the actors.

3698 **9.1.5. Message modification**

3699 If an adversary can intercept a message and change its contents, then they may be able to alter an
3700 **authorization decision**. A message integrity safeguard can prevent a successful message
3701 modification attack.

3702 **9.1.6. NotApplicable results**

3703 A result of "NotApplicable" means that the **PDP** could not locate a **policy** whose **target** matched
3704 the information in the **decision request**. In general, it is highly recommended that a "Deny" **effect**
3705 **policy** be used, so that when a **PDP** would have returned "NotApplicable", a result of "Deny" is
3706 returned instead.

3707 In some security models, however, such as those found in many Web Servers, an **authorization**
3708 **decision** of "NotApplicable" is treated as equivalent to "Permit". There are particular security
3709 considerations that must be taken into account for this to be safe. These are explained in the
3710 following paragraphs.

3711 If "NotApplicable" is to be treated as "Permit", it is vital that the matching algorithms used by the
3712 **policy** to match elements in the **decision request** be closely aligned with the data syntax used by
3713 the applications that will be submitting the **decision request**. A failure to match will result in
3714 "NotApplicable" and be treated as "Permit". So an unintended failure to match may allow
3715 unintended **access**.

3716 Commercial http responders allow a variety of syntaxes to be treated equivalently. The "%" can be
3717 used to represent characters by hex value. The URL path "/../" provides multiple ways of specifying
3718 the same value. Multiple character sets may be permitted and, in some cases, the same printed
3719 character can be represented by different binary values. Unless the matching algorithm used by
3720 the policy is sophisticated enough to catch these variations, unintended access may be permitted.

3721 It may be safe to treat "NotApplicable" as "Permit" only in a closed environment where all
3722 applications that formulate a **decision request** can be guaranteed to use the exact syntax
3723 expected by the **policies**. In a more open environment, where **decision requests** may be received
3724 from applications that use any legal syntax, it is strongly recommended that "NotApplicable" NOT
3725 be treated as "Permit" unless matching rules have been very carefully designed to match all
3726 possible applicable inputs, regardless of syntax or type variations. Note, however, that according to
3727 Section 7.1, a **PEP** must deny **access** unless it receives an explicit "Permit" **authorization**
3728 **decision**.

3729 **9.1.7. Negative rules**

3730 A negative **rule** is one that is based on a **predicate** not being "True". If not used with care,
3731 negative **rules** can lead to a policy violation, therefore some authorities recommend that they not
3732 be used. However, negative **rules** can be extremely efficient in certain cases, so XACML has
3733 chosen to include them. Nevertheless, it is recommended that they be used with care and avoided
3734 if possible.

3735 A common use for negative **rules** is to deny **access** to an individual or subgroup when their
3736 membership in a larger group would otherwise permit them access. For example, we might want to
3737 write a **rule** that allows all Vice Presidents to see the unpublished financial data, except for Joe,

3738 who is only a Ceremonial Vice President and can be indiscreet in his communications. If we have
3739 complete control over the administration of **subject attributes**, a superior approach would be to
3740 define "Vice President" and "Ceremonial Vice President" as distinct groups and then define **rules**
3741 accordingly. However, in some environments this approach may not be feasible. (It is worth noting
3742 in passing that, generally speaking, referring to individuals in **rules** does not scale well. Generally,
3743 shared **attributes** are preferred.)

3744 If not used with care, negative **rules** can lead to policy violation in two common cases. They are:
3745 when **attributes** are suppressed and when the base group changes. An example of suppressed
3746 **attributes** would be if we have a policy that **access** should be permitted, *unless* the **subject** is a
3747 credit risk. If it is possible that the **attribute** of being a credit risk may be unknown to the **PDP** for
3748 some reason, then unauthorized **access** may result. In some environments, the **subject** may be
3749 able to suppress the publication of **attributes** by the application of privacy controls, or the server or
3750 repository that contains the information may be unavailable for accidental or intentional reasons.

3751 An example of a changing base group would be if there is a policy that everyone in the engineering
3752 department may change software source code, except for secretaries. Suppose now that the
3753 department was to merge with another engineering department and the intent is to maintain the
3754 same policy. However, the new department also includes individuals identified as administrative
3755 assistants, who ought to be treated in the same way as secretaries. Unless the policy is altered,
3756 they will unintentionally be permitted to change software source code. Problems of this type are
3757 easy to avoid when one individual administers all **policies**, but when administration is distributed,
3758 as XACML allows, this type of situation must be explicitly guarded against.

3759 9.2. Safeguards

3760 9.2.1. Authentication

3761 Authentication provides the means for one party in a transaction to determine the identity of the
3762 other party in the transaction. Authentication may be in one direction, or it may be bilateral.

3763 Given the sensitive nature of **access control** systems, it is important for a **PEP** to authenticate the
3764 identity of the **PDP** to which it sends **decision requests**. Otherwise, there is a risk that an
3765 adversary could provide false or invalid **authorization decisions**, leading to a policy violation.

3766 It is equally important for a **PDP** to authenticate the identity of the **PEP** and assess the level of trust
3767 to determine what, if any, sensitive data should be passed. One should keep in mind that even
3768 simple "Permit" or "Deny" responses could be exploited if an adversary were allowed to make
3769 unlimited requests to a **PDP**.

3770 Many different techniques may be used to provide authentication, such as co-located code, a
3771 private network, a VPN or digital signatures. Authentication may also be performed as part of the
3772 communication protocol used to exchange the **contexts**. In this case, authentication may be
3773 performed either at the message level or at the session level.

3774 9.2.2. Policy administration

3775 If the contents of **policies** are exposed outside of the **access control** system, potential **subjects**
3776 may use this information to determine how to gain unauthorized **access**.

3777 To prevent this threat, the repository used for the storage of **policies** may itself require **access**
3778 **control**. In addition, the <Status> element should be used to return values of missing **attributes**
3779 only when exposure of the identities of those **attributes** will not compromise security.

3780

9.2.3. Confidentiality

3781 Confidentiality mechanisms ensure that the contents of a message can be read only by the desired
3782 recipients and not by anyone else who encounters the message while it is in transit. There are two
3783 areas in which confidentiality should be considered: one is confidentiality during transmission; the
3784 other is confidentiality within a `<Policy>` element.

3785

9.2.3.1. Communication confidentiality

3786 In some environments it is deemed good practice to treat all data within an **access control** system
3787 as confidential. In other environments, **policies** may be made freely available for distribution,
3788 inspection and audit. The idea behind keeping **policy** information secret is to make it more difficult
3789 for an adversary to know what steps might be sufficient to obtain unauthorized **access**. Regardless
3790 of the approach chosen, the security of the **access control** system should not depend on the
3791 secrecy of the **policy**.

3792 Any security considerations related to transmitting or exchanging XACML `<Policy>` elements are
3793 outside the scope of the XACML standard. While it is often important to ensure that the integrity
3794 and confidentiality of `<Policy>` elements is maintained when they are exchanged between two
3795 parties, it is left to the implementers to determine the appropriate mechanisms for their
3796 environment.

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

3800

9.2.3.2. Statement level confidentiality

3801 In some cases, an implementation may want to encrypt only parts of an XACML `<Policy>`
3802 element.

3803 The XML Encryption Syntax and Processing Candidate Recommendation from W3C can be used
3804 to encrypt all or parts of an XML document. This specification is recommended for use with
3805 XACML.

3806 It should go without saying that if a repository is used to facilitate the communication of cleartext
3807 (i.e., unencrypted) **policy** between the **PAP** and **PDP**, then a secure repository should be used to
3808 store this sensitive data.

3809

9.2.4. Policy integrity

3810 The XACML **policy**, used by the **PDP** to evaluate the request **context**, is the heart of the system.
3811 Therefore, maintaining its integrity is essential. There are two aspects to maintaining the integrity of
3812 the **policy**. One is to ensure that `<Policy>` elements have not been altered since they were
3813 originally created by the **PAP**. The other is to ensure that `<Policy>` elements have not been
3814 inserted or deleted from the set of **policies**.

3815 In many cases, both aspects can be achieved by ensuring the integrity of the actors and
3816 implementing session-level mechanisms to secure the communication between actors. The
3817 selection of the appropriate mechanisms is left to the implementers. However, when **policy** is
3818 distributed between organizations to be acted on at a later time, or when the **policy** travels with the
3819 protected resource, it would be useful to sign the **policy**. In these cases, the XML Signature
3820 Syntax and Processing standard from W3C is recommended to be used with XACML.

3821 Digital signatures should only be used to ensure the integrity of the statements. Digital signatures
3822 should not be used as a method of selecting or evaluating **policy**. That is, the **PDP** should not
3823 request a **policy** based on who signed it or whether or not it has been signed (as such a basis for
3824 selection would, itself, be a matter of policy). However, the **PDP** must verify that the key used to
3825 sign the **policy** is one controlled by the purported issuer of the **policy**. The means to do this are
3826 dependent on the specific signature technology chosen and are outside the scope of this document.

3827 **9.2.5. Policy identifiers**

3828 Since **policies** can be referenced by their identifiers, it is the responsibility of the **PAP** to ensure
3829 that these are unique. Confusion between identifiers could lead to misidentification of the
3830 **applicable policy**. This specification is silent on whether a **PAP** must generate a new identifier
3831 when a **policy** is modified or may use the same identifier in the modified **policy**. This is a matter of
3832 administrative practice. However, care must be taken in either case. If the identifier is reused,
3833 there is a danger that other **policies** or **policy sets** that reference it may be adversely affected.
3834 Conversely, if a new identifier is used, these other **policies** may continue to use the prior **policy**,
3835 unless it is deleted. In either case the results may not be what the **policy** administrator intends.

3836 **9.2.6. Trust model**

3837 Discussions of authentication, integrity and confidentiality safeguards necessarily assume an
3838 underlying trust model: how can one actor come to believe that a given key is uniquely associated
3839 with a specific, identified actor so that the key can be used to encrypt data for that actor or verify
3840 signatures (or other integrity structures) from that actor? Many different types of trust model exist,
3841 including strict hierarchies, distributed authorities, the Web, the bridge and so on.

3842 It is worth considering the relationships between the various actors of the **access control** system in
3843 terms of the interdependencies that do and do not exist.

- 3844 • None of the entities of the authorization system are dependent on the **PEP**. They may
3845 collect data from it, for example authentication data, but are responsible for verifying it
3846 themselves.
- 3847 • The correct operation of the system depends on the ability of the **PEP** to actually enforce
3848 **policy** decisions.
- 3849 • The **PEP** depends on the **PDP** to correctly evaluate **policies**. This in turn implies that the
3850 **PDP** is supplied with the correct inputs. Other than that, the **PDP** does not depend on the
3851 **PEP**.
- 3852 • The **PDP** depends on the **PAP** to supply appropriate policies. The **PAP** is not dependent
3853 on other components.

3854 **9.2.7. Privacy**

3855 It is important to be aware that any transactions that occur with respect to **access control** may
3856 reveal private information about the actors. For example, if an XACML **policy** states that certain
3857 data may only be read by **subjects** with "Gold Card Member" status, then any transaction in which
3858 a **subject** is permitted **access** to that data leaks information to an adversary about the **subject's**
3859 status. Privacy considerations may therefore lead to encryption and/or to access control
3860 requirements surrounding the enforcement of XACML **policy** instances themselves: confidentiality-
3861 protected channels for the request/response protocol messages, protection of **subject attributes** in
3862 storage and in transit, and so on.

3863 Selection and use of privacy mechanisms appropriate to a given environment are outside the scope
 3864 of XACML. The decision regarding whether, how and when to deploy such mechanisms is left to
 3865 the implementers associated with the environment.

3866 10. Conformance (normative)

3867 10.1. Introduction

3868 The XACML specification addresses the following aspect of conformance:

3869 The XACML specification defines a number of functions, etc. that have somewhat special
 3870 application, therefore they are not required to be implemented in an implementation that claims to
 3871 conform with the OASIS standard.

3872 10.2. Conformance tables

3873 This section lists those portions of the specification that **MUST** be included in an implementation of
 3874 a **PDP** that claims to conform with XACML v2.0. A set of test cases has been created to assist in
 3875 this process. These test cases are hosted by Sun Microsystems and can be located from the
 3876 XACML Web page. The site hosting the test cases contains a full description of the test cases and
 3877 how to execute them.

3878 Note: "M" means mandatory-to-implement. "O" means optional.

3879 10.2.1. Schema elements

3880 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:MissingAttributeDetail	M
xacml-context:Obligations	O
xacml-context:Request	M
xacml-context:Resource	M
xacml-context:ResourceContent	O
xacml-context:Response	M
xacml-context:Result	M
xacml-context:Status	M
xacml-context:StatusCode	M
xacml-context:StatusDetail	O
xacml-context:StatusMessage	O
xacml-context:Subject	M
xacml:Action	M
xacml:ActionAttributeDesignator	M
xacml:ActionMatch	M
xacml:Actions	M
xacml:Apply	M
xacml:AttributeAssignment	O
xacml:AttributeSelector	O
xacml:AttributeValue	M

xacml:CombinerParameters	O
xacml:CombinerParameter	O
xacml:Condition	M
xacml:Description	M
xacml:Environment	M
xacml:EnvironmentMatch	M
xacml:EnvironmentAttributeDesignator	M
xacml:Environments	M
xacml:Expression	M
xacml:Function	M
xacml:Obligation	O
xacml:Obligations	O
xacml:Policy	M
xacml:PolicyCombinerParameters	O
xacml:PolicyDefaults	O
xacml:PolicyIdReference	M
xacml:PolicySet	M
xacml:PolicySetDefaults	O
xacml:PolicySetIdReference	M
xacml:Resource	M
xacml:ResourceAttributeDesignator	M
xacml:ResourceMatch	M
xacml:Resources	M
xacml:Rule	M
xacml:RuleCombinerParameters	O
xacml:Subject	M
xacml:SubjectMatch	M
xacml:Subjects	M
xacml:Target	M
xacml:VariableDefinition	M
xacml:VariableReference	M
xacml:XPathVersion	O

3881 **10.2.2. Identifier Prefixes**

3882 The following identifier prefixes are reserved by XACML.

Identifier
urn:oasis:names:tc:xacml:2.0
urn:oasis:names:tc:xacml:2.0:conformance-test
urn:oasis:names:tc:xacml:2.0:context
urn:oasis:names:tc:xacml:2.0:example
urn:oasis:names:tc:xacml:1.0:function
urn:oasis:names:tc:xacml:2.0:function
urn:oasis:names:tc:xacml:2.0:policy
urn:oasis:names:tc:xacml:1.0:subject
urn:oasis:names:tc:xacml:1.0:resource
urn:oasis:names:tc:xacml:1.0:action
urn:oasis:names:tc:xacml:1.0:environment
urn:oasis:names:tc:xacml:1.0:status

3883 **10.2.3. Algorithms**

3884 The implementation MUST include the rule- and policy-combining algorithms associated with the
 3885 following identifiers that are marked "M".

3886

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

3887

10.2.4. Status Codes

3888 Implementation support for the <StatusCode> element is optional, but if the element is supported,
3889 then the following status codes must be supported and must be used in the way XACML has
3890 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

3891

10.2.5. Attributes

3892 The implementation MUST support the **attributes** associated with the following identifiers as
3893 specified by XACML. If values for these **attributes** are not present in the **decision request**, then
3894 their values MUST be supplied by the **context handler**. So, unlike most other **attributes**, their
3895 semantics are not transparent to the **PDP**.

Identifier	M/O
urn:oasis:names:tc:xacml:1.0:environment:current-time	M
urn:oasis:names:tc:xacml:1.0:environment:current-date	M
urn:oasis:names:tc:xacml:1.0:environment:current-dateTime	M

3896

10.2.6. Identifiers

3897 The implementation MUST use the **attributes** associated with the following identifiers in the way
3898 XACML has defined. This requirement pertains primarily to implementations of a **PAP** or **PEP** that
3899 uses XACML, since the semantics of the attributes are transparent to the **PDP**.

3900

3901

3902

3903

Identifier	M/O
urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name	O
urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address	O
urn:oasis:names:tc:xacml:1.0:subject:authentication-method	O
urn:oasis:names:tc:xacml:1.0:subject:authentication-time	O
urn:oasis:names:tc:xacml:1.0:subject:key-info	O
urn:oasis:names:tc:xacml:1.0:subject:request-time	O
urn:oasis:names:tc:xacml:1.0:subject:session-start-time	O
urn:oasis:names:tc:xacml:1.0:subject:subject-id	O
urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier	O
urn:oasis:names:tc:xacml:1.0:subject-category:access-subject	M
urn:oasis:names:tc:xacml:1.0:subject-category:codebase	O
urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject	O
urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject	O
urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine	O
urn:oasis:names:tc:xacml:1.0:resource:resource-location	O
urn:oasis:names:tc:xacml:1.0:resource:resource-id	M
urn:oasis:names:tc:xacml:1.0:resource:simple-file-name	O
urn:oasis:names:tc:xacml:1.0:action:action-id	O
urn:oasis:names:tc:xacml:1.0:action:implied-action	O

3904 **10.2.7. Data-types**

3905 The implementation MUST support the data-types associated with the following identifiers marked
 3906 "M".

Data-type	M/O
http://www.w3.org/2001/XMLSchema#string	M
http://www.w3.org/2001/XMLSchema#boolean	M
http://www.w3.org/2001/XMLSchema#integer	M
http://www.w3.org/2001/XMLSchema#double	M
http://www.w3.org/2001/XMLSchema#time	M
http://www.w3.org/2001/XMLSchema#date	M
http://www.w3.org/2001/XMLSchema#dateTime	M
http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration	M
http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration	M
http://www.w3.org/2001/XMLSchema#anyURI	M
http://www.w3.org/2001/XMLSchema#hexBinary	M
http://www.w3.org/2001/XMLSchema#base64Binary	M
urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name	M
urn:oasis:names:tc:xacml:1.0:data-type:x500Name	M

3907 **10.2.8. Functions**

3908 The implementation MUST properly process those functions associated with the identifiers marked
 3909 with an "M".

Function	M/O
urn:oasis:names:tc:xacml:1.0:function:string-equal	M
urn:oasis:names:tc:xacml:1.0:function:boolean-equal	M
urn:oasis:names:tc:xacml:1.0:function:integer-equal	M
urn:oasis:names:tc:xacml:1.0:function:double-equal	M
urn:oasis:names:tc:xacml:1.0:function:date-equal	M
urn:oasis:names:tc:xacml:1.0:function:time-equal	M

urn:oasis:names:tc:xacml:1.0:function:dateTime-equal	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-equal	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-equal	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-equal	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-equal	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal	M
urn:oasis:names:tc:xacml:1.0:function:integer-add	M
urn:oasis:names:tc:xacml:1.0:function:double-add	M
urn:oasis:names:tc:xacml:1.0:function:integer-subtract	M
urn:oasis:names:tc:xacml:1.0:function:double-subtract	M
urn:oasis:names:tc:xacml:1.0:function:integer-multiply	M
urn:oasis:names:tc:xacml:1.0:function:double-multiply	M
urn:oasis:names:tc:xacml:1.0:function:integer-divide	M
urn:oasis:names:tc:xacml:1.0:function:double-divide	M
urn:oasis:names:tc:xacml:1.0:function:integer-mod	M
urn:oasis:names:tc:xacml:1.0:function:integer-abs	M
urn:oasis:names:tc:xacml:1.0:function:double-abs	M
urn:oasis:names:tc:xacml:1.0:function:round	M
urn:oasis:names:tc:xacml:1.0:function:floor	M
urn:oasis:names:tc:xacml:1.0:function:string-normalize-space	M
urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case	M
urn:oasis:names:tc:xacml:1.0:function:double-to-integer	M
urn:oasis:names:tc:xacml:1.0:function:integer-to-double	M
urn:oasis:names:tc:xacml:1.0:function:or	M
urn:oasis:names:tc:xacml:1.0:function:and	M
urn:oasis:names:tc:xacml:1.0:function:n-of	M
urn:oasis:names:tc:xacml:1.0:function:not	M
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:integer-less-than	M
urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:double-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:double-less-than	M
urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-dayTimeDuration	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:date-subtract-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:string-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:string-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:string-less-than	M
urn:oasis:names:tc:xacml:1.0:function:string-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:time-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:time-less-than	M
urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal	M
urn:oasis:names:tc:xacml:2.0:function:time-in-range	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than	M

urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:date-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:date-less-than	M
urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:string-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:string-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:string-is-in	M
urn:oasis:names:tc:xacml:1.0:function:string-bag	M
urn:oasis:names:tc:xacml:1.0:function:boolean-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:boolean-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:boolean-is-in	M
urn:oasis:names:tc:xacml:1.0:function:boolean-bag	M
urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:integer-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:integer-is-in	M
urn:oasis:names:tc:xacml:1.0:function:integer-bag	M
urn:oasis:names:tc:xacml:1.0:function:double-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:double-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:double-is-in	M
urn:oasis:names:tc:xacml:1.0:function:double-bag	M
urn:oasis:names:tc:xacml:1.0:function:time-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:time-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:time-is-in	M
urn:oasis:names:tc:xacml:1.0:function:time-bag	M
urn:oasis:names:tc:xacml:1.0:function:date-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:date-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:date-is-in	M
urn:oasis:names:tc:xacml:1.0:function:date-bag	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-is-in	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-bag	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-is-in	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-bag	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-is-in	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-bag	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-is-in	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-bag	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-is-in	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-bag	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-is-in	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-bag	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-is-in	M

urn:oasis:names:tc:xacml:1.0:function:x500Name-bag	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-is-in	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-bag	M
urn:oasis:names:tc:xacml:2.0:function:string-concatenate	M
urn:oasis:names:tc:xacml:2.0:function:uri-string-concatenate	M
urn:oasis:names:tc:xacml:1.0:function:any-of	M
urn:oasis:names:tc:xacml:1.0:function:all-of	M
urn:oasis:names:tc:xacml:1.0:function:any-of-any	M
urn:oasis:names:tc:xacml:1.0:function:all-of-any	M
urn:oasis:names:tc:xacml:1.0:function:any-of-all	M
urn:oasis:names:tc:xacml:1.0:function:all-of-all	M
urn:oasis:names:tc:xacml:1.0:function:map	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-match	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match	M
urn:oasis:names:tc:xacml:1.0:function:string-regexp-match	M
urn:oasis:names:tc:xacml:2.0:function:anyURI-regexp-match	M
urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match	M
urn:oasis:names:tc:xacml:2.0:function:dnsName-regexp-match	M
urn:oasis:names:tc:xacml:2.0:function:rfc822Name-regexp-match	M
urn:oasis:names:tc:xacml:2.0:function:x500Name-regexp-match	M
urn:oasis:names:tc:xacml:1.0:function:xpath-node-count	O
urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal	O
urn:oasis:names:tc:xacml:1.0:function:xpath-node-match	O
urn:oasis:names:tc:xacml:1.0:function:string-intersection	M
urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:string-union	M
urn:oasis:names:tc:xacml:1.0:function:string-subset	M
urn:oasis:names:tc:xacml:1.0:function:string-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:boolean-intersection	M
urn:oasis:names:tc:xacml:1.0:function:boolean-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:boolean-union	M
urn:oasis:names:tc:xacml:1.0:function:boolean-subset	M
urn:oasis:names:tc:xacml:1.0:function:boolean-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:integer-intersection	M
urn:oasis:names:tc:xacml:1.0:function:integer-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:integer-union	M
urn:oasis:names:tc:xacml:1.0:function:integer-subset	M
urn:oasis:names:tc:xacml:1.0:function:integer-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:double-intersection	M
urn:oasis:names:tc:xacml:1.0:function:double-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:double-union	M
urn:oasis:names:tc:xacml:1.0:function:double-subset	M
urn:oasis:names:tc:xacml:1.0:function:double-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:time-intersection	M
urn:oasis:names:tc:xacml:1.0:function:time-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:time-union	M
urn:oasis:names:tc:xacml:1.0:function:time-subset	M
urn:oasis:names:tc:xacml:1.0:function:time-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:date-intersection	M
urn:oasis:names:tc:xacml:1.0:function:date-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:date-union	M
urn:oasis:names:tc:xacml:1.0:function:date-subset	M
urn:oasis:names:tc:xacml:1.0:function:date-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-intersection	M

urn:oasis:names:tc:xacml:1.0:function:dateTime-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-union	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-subset	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-intersection	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-union	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-subset	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-intersection	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-union	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-subset	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-intersection	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-union	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-subset	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-intersection	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-union	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-subset	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-intersection	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-union	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-subset	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-intersection	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-union	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-subset	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-intersection	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-union	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-subset	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-set-equals	M

3910

3911

11. References

- 3912 **[DS]** D. Eastlake et al., *XML-Signature Syntax and Processing*,
3913 <http://www.w3.org/TR/xmlsig-core/>, World Wide Web Consortium.
- 3914 **[Hancock]** Hancock, "Polymorphic Type Checking", in Simon L. Peyton Jones,
3915 "Implementation of Functional Programming Languages", Section 8,
3916 Prentice-Hall International, 1987
- 3917 **[Haskell]** Haskell, a purely functional language. Available at
3918 <http://www.haskell.org/>

3919	[Hier]	Anderson, A., ed., "Hierarchical resource profile of XACML v2.0", OASIS Standard, 1 February 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-hier-profile-spec-os.pdf
3920		
3921		
3922	[Hinton94]	Hinton, H, M, Lee,, E, S, The Compatibility of Policies, Proceedings 2nd ACM Conference on Computer and Communications Security, Nov 1994, Fairfax, Virginia, USA.
3923		
3924		
3925	[IEEE754]	IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-7653-8, IEEE Product No. SH10116-TBR
3926		
3927	[ISO10181-3]	ISO/IEC 10181-3:1996 Information technology – Open Systems Interconnection -- Security frameworks for open systems: Access control framework.
3928		
3929		
3930	[Kudo00]	Kudo M and Hada S, XML document security based on provisional authorization, Proceedings of the Seventh ACM Conference on Computer and Communications Security, Nov 2000, Athens, Greece, pp 87-96.
3931		
3932		
3933	[LDAP-1]	RFC2256, A summary of the X500(96) User Schema for use with LDAPv3, Section 5, M Wahl, December 1997 http://www.ietf.org/rfc/rfc2798.txt
3934		
3935	[LDAP-2]	RFC2798, Definition of the inetOrgPerson, M. Smith, April 2000
3936		http://www.ietf.org/rfc/rfc2798.txt
3937	[MathML]	Mathematical Markup Language (MathML), Version 2.0, W3C Recommendation, 21 February 2001. Available at:
3938		http://www.w3.org/TR/MathML2/
3939		
3940	[Multi]	Anderson, A., ed., "Multiple resource profile of XACML v2.0", OASIS Standard, 1 February 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-mult-profile-spec-os.pdf
3941		
3942		
3943	[Perritt93]	Perritt, H. Knowbots, Permissions Headers and Contract Law, Conference on Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment, April 1993. Available at:
3944		http://www.ifla.org/documents/infopol/copyright/perh2.txt
3945		
3946		
3947	[RBAC]	Role-Based Access Controls, David Ferraiolo and Richard Kuhn, 15th National Computer Security Conference, 1992. Available at:
3948		http://csrc.nist.gov/rbac
3949		
3950	[RegEx]	XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, Appendix D. Available at: http://www.w3.org/TR/xmlschema-0/
3951		
3952	[RFC2119]	S. Bradner, <i>Key words for use in RFCs to Indicate Requirement Levels</i> , http://www.ietf.org/rfc/rfc2119.txt , IETF RFC 2119, March 1997
3953		
3954	[RFC2396]	Berners-Lee T, Fielding R, Masinter L, Uniform Resource Identifiers (URI): Generic Syntax. Available at: http://www.ietf.org/rfc/rfc2396.txt
3955		
3956	[RFC2732]	Hinden R, Carpenter B, Masinter L, Format for Literal IPv6 Addresses in URL's. Available at: http://www.ietf.org/rfc/rfc2732.txt
3957		
3958	[RFC3198]	IETF RFC 3198: Terminology for Policy-Based Management, November 2001. http://www.ietf.org/rfc/rfc3198.txt
3959		
3960	[SAML]	Security Assertion Markup Language available from http://www.oasis-open.org/committees/security/#documents
3961		
3962	[Sloman94]	Sloman, M. Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Volume 2, part 4. Plenum Press. 1994.
3963		
3964		
3965	[XACMLv1.0]	Extensible access control markup language (XACML) Version 1.0. OASIS Standard. 18 February 2003. Available at: http://www.oasis-
3966		

3967		open.org/apps/org/workgroup/xacml/download.php/940/oasis-xacml-1.0.pdf
3968		
3969	[XACMLv1.1]	Extensible access control markup language (XACML) Version 1.1. OASIS Committee Specification. 7 August 2003. Available at: http://www.oasis-open.org/apps/org/workgroup/xacml/download.php/4104/cs-xacml-specification-1.1.pdf
3970		
3971		
3972		
3973	[XF]	XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft 16 August 2002. Available at: http://www.w3.org/TR/2002/WD-xquery-operators-20020816
3974		
3975		
3976	[XS]	XML Schema, parts 1 and 2. Available at: http://www.w3.org/TR/xmlschema-1/ and http://www.w3.org/TR/xmlschema-2/
3977		
3978		
3979	[XPath]	XML Path Language (XPath), Version 1.0, W3C Recommendation 16 November 1999. Available at: http://www.w3.org/TR/xpath
3980		
3981	[XSLT]	XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999. Available at: http://www.w3.org/TR/xslt
3982		
3983		

3984 Appendix A. Data-types and functions (normative)

3985 A.1. Introduction

3986 This section specifies the data-types and functions used in XACML to create *predicates* for
3987 *conditions* and *target* matches.

3988 This specification combines the various standards set forth by IEEE and ANSI for string
3989 representation of numeric values, as well as the evaluation of arithmetic functions. It describes the
3990 primitive data-types and *bags*. The standard functions are named and their operational semantics
3991 are described.

3992 A.2. Data-types

3993 Although XML instances represent all data-types as strings, an XACML *PDP* must reason about
3994 types of data that, while they have string representations, are not just strings. Types such as
3995 Boolean, integer and double MUST be converted from their XML string representations to values
3996 that can be compared with values in their domain of discourse, such as numbers. The following
3997 primitive data-types are specified for use with XACML and have explicit data representations:

- 3998 • <http://www.w3.org/2001/XMLSchema#string>
- 3999 • <http://www.w3.org/2001/XMLSchema#boolean>
- 4000 • <http://www.w3.org/2001/XMLSchema#integer>
- 4001 • <http://www.w3.org/2001/XMLSchema#double>
- 4002 • <http://www.w3.org/2001/XMLSchema#time>
- 4003 • <http://www.w3.org/2001/XMLSchema#date>
- 4004 • <http://www.w3.org/2001/XMLSchema#dateTime>
- 4005 • <http://www.w3.org/2001/XMLSchema#anyURI>
- 4006 • <http://www.w3.org/2001/XMLSchema#hexBinary>
- 4007 • <http://www.w3.org/2001/XMLSchema#base64Binary>
- 4008 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration>
- 4009 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration>
- 4010 • <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
- 4011 • <urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name>
- 4012 • <urn:oasis:names:tc:xacml:2.0:data-type:ipAddress>
- 4013 • <urn:oasis:names:tc:xacml:2.0:data-type:dnsName>

4014 For the sake of improved interoperability, it is RECOMMENDED that all time references be in UTC
4015 time.

4016 An XACML **PDP** SHALL be capable of converting string representations into various primitive data-
4017 types. For integers and doubles, XACML SHALL use the conversions described in [IEEE754].

4018 XACML defines three data-types; these are:

4019 "urn:oasis:names:tc:xacml:1.0:data-type:x500Name",

4020 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"

4021 "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress"

4022 "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" and

4023 These types represent identifiers for subjects or resources and appear in several standard
4024 applications, such as TLS/SSL and electronic mail.

4025 **X.500 directory name**

4026 The "urn:oasis:names:tc:xacml:1.0:data-type:x500Name" primitive type represents an ITU-T Rec.
4027 X.520 Distinguished Name. The valid syntax for such a name is described in IETF RFC 2253
4028 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names"

4029 **RFC 822 name**

4030 The "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" primitive type represents an electronic
4031 mail address. The valid syntax for such a name is described in IETF RFC 2821, Section 4.1.2,
4032 Command Argument Syntax, under the term "Mailbox".

4033 **IP address**

4034 The "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" primitive type represents an IPv4 or IPv6
4035 network address, with optional mask and optional port or port range. The syntax SHALL be:

4036

4037 ipAddress = address ["/" mask] [":" [portrange]]

4038

4039 For an IPv4 address, the address and mask are formatted in accordance with the syntax for a
4040 "host" in IETF RFC 2396 "Uniform Resource Identifiers (URI): Generic Syntax", section 3.2.

4041 For an IPv6 address, the address and mask are formatted in accordance with the syntax for an

4042 "ipv6reference" in IETF RFC 2732 "Format for Literal IPv6 Addresses in URL's". (Note that an IPv6
4043 address or mask, in this syntax, is enclosed in literal "[" "]" brackets.)

4044

4045 **DNS name**

4046 The "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" primitive type represents a Domain Name
4047 Service (DNS) host name, with optional port or port range. The syntax SHALL be:

4048

4049 dnsName = hostname [":" portrange]

4050

4051 The hostname is formatted in accordance with IETF RFC 2396 "Uniform Resource Identifiers (URI):
4052 Generic Syntax", section 3.2, except that a wildcard "*" may be used in the left-most component of
4053 the hostname to indicate "any subdomain" under the domain specified to its right.

4054

4055 For both the "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" and

4056 "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" data-types, the port or port range syntax
4057 SHALL be

4058

4059 portrange = portnumber | "-"portnumber | portnumber-"[portnumber]

4060

4061 where "portnumber" is a decimal port number. If the port number is of the form "-x", where "x" is a
4062 port number, then the range is all ports numbered "x" and below. If the port number is of the form

4063 "x-", then the range is all ports numbered "x" and above. [This syntax is taken from the Java
4064 SocketPermission.]

4065 **A.3. Functions**

4066 XACML specifies the following functions. If an argument of one of these functions were to evaluate
4067 to "Indeterminate", then the function SHALL be set to "Indeterminate".

4068 **A.3.1 Equality predicates**

4069 The following functions are the *equality* functions for the various primitive types. Each function for a
4070 particular data-type follows a specified standard convention for that data-type.

- 4071 • urn:oasis:names:tc:xacml:1.0:function:string-equal

4072 This function SHALL take two arguments of data-type
4073 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4074 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and
4075 only if the value of both of its arguments are of equal length and each string is determined
4076 to be equal byte-by-byte according to the function "integer-equal". Otherwise, it SHALL
4077 return "False".

- 4078 • urn:oasis:names:tc:xacml:1.0:function:boolean-equal

4079 This function SHALL take two arguments of data-type
4080 "http://www.w3.org/2001/XMLSchema#boolean" and SHALL return an
4081 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and
4082 only if the arguments are equal. Otherwise, it SHALL return "False".

- 4083 • urn:oasis:names:tc:xacml:1.0:function:integer-equal

4084 This function SHALL take two arguments of data-type
4085 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return an
4086 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on
4087 integers according to IEEE 754 [IEEE 754].

- 4088 • urn:oasis:names:tc:xacml:1.0:function:double-equal

4089 This function SHALL take two arguments of data-type
4090 "http://www.w3.org/2001/XMLSchema#double" and SHALL return an
4091 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on
4092 doubles according to IEEE 754 [IEEE 754].

- 4093 • urn:oasis:names:tc:xacml:1.0:function:date-equal

4094 This function SHALL take two arguments of data-type
4095 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4096 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
4097 according to the "op:date-equal" function [XF Section 8.3.11].

- 4098 • urn:oasis:names:tc:xacml:1.0:function:time-equal

4099 This function SHALL take two arguments of data-type
4100 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
4101 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
4102 according to the "op:time-equal" function [XF Section 8.3.14].

- 4103 • urn:oasis:names:tc:xacml:1.0:function:dateTime-equal
- 4104 This function SHALL take two arguments of data-type
 4105 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
 4106 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
 4107 according to the "op:dateTime-equal" function [XF Section 8.3.8].
- 4108 • urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-equal
- 4109 This function SHALL take two arguments of data-type "http://www.w3.org/TR/2002/WD-
 4110 xquery-operators-20020816#dayTimeDuration" and SHALL return an
 4111 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
 4112 according to the "op:dayTimeDuration-equal" function [XF Section 8.3.5]. Note that the
 4113 lexical representation of each argument MUST be converted to a value expressed in
 4114 fractional seconds [XF Section 8.2.2].
- 4115 • urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-equal
- 4116 This function SHALL take two arguments of data-type "http://www.w3.org/TR/2002/WD-
 4117 xquery-operators-20020816#yearMonthDuration" and SHALL return an
 4118 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
 4119 according to the "op:yearMonthDuration-equal" function [XF Section 8.3.2]. Note that the
 4120 lexical representation of each argument MUST be converted to a value expressed in
 4121 integer months [XF Section 8.2.1].
- 4122 • urn:oasis:names:tc:xacml:1.0:function:anyURI-equal
- 4123 This function SHALL take two arguments of data-type
 4124 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return an
 4125 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
 4126 according to the "op:anyURI-equal" function [XF Section 10.2.1].
- 4127 • urn:oasis:names:tc:xacml:1.0:function:x500Name-equal
- 4128 This function SHALL take two arguments of "urn:oasis:names:tc:xacml:1.0:data-
 4129 type:x500Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It
 4130 SHALL return "True" if and only if each Relative Distinguished Name (RDN) in the two
 4131 arguments matches. Otherwise, it SHALL return "False". Two RDNs shall be said to
 4132 match if and only if the result of the following operations is "True"³.
- 4133 1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory
 4134 Access Protocol (v3): UTF-8 String Representation of Distinguished Names".
 - 4135 2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute
 4136 ValuePairs in that RDN in ascending order when compared as octet strings
 4137 (described in ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").
 - 4138 3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key
 4139 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section
 4140 4.1.2.4 "Issuer".
- 4141 • urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal
- 4142 This function SHALL take two arguments of data-type "urn:oasis:names:tc:xacml:1.0:data-
 4143 type:rfc822Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean".

³ 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.

- 4144 It SHALL return "True" if and only if the two arguments are equal. Otherwise, it SHALL
 4145 return "False". An RFC822 name consists of a *local-part* followed by "@" followed by a
 4146 *domain-part*. The *local-part* is case-sensitive, while the *domain-part* (which is usually a
 4147 DNS host name) is not case-sensitive. Perform the following operations:
- 4148 1. Normalize the *domain-part* of each argument to lower case
 - 4149 2. Compare the expressions by applying the function
 4150 "urn:oasis:names:tc:xacml:1.0:function:string-equal" to the normalized arguments.
- 4151 • urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal
 4152 This function SHALL take two arguments of data-type
 4153 "http://www.w3.org/2001/XMLSchema#hexBinary" and SHALL return an
 4154 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the octet
 4155 sequences represented by the value of both arguments have equal length and are equal in
 4156 a conjunctive, point-wise, comparison using the
 4157 "urn:oasis:names:tc:xacml:1.0:function:integer-equal" function. Otherwise, it SHALL return
 4158 "False". The conversion from the string representation to an octet sequence SHALL be as
 4159 specified in [XS Section 8.2.15].
 - 4160 • urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal
 4161 This function SHALL take two arguments of data-type
 4162 "http://www.w3.org/2001/XMLSchema#base64Binary" and SHALL return an
 4163 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the octet
 4164 sequences represented by the value of both arguments have equal length and are equal in
 4165 a conjunctive, point-wise, comparison using the
 4166 "urn:oasis:names:tc:xacml:1.0:function:integer-equal" function. Otherwise, it SHALL return
 4167 "False". The conversion from the string representation to an octet sequence SHALL be as
 4168 specified in [XS Section 8.2.16].

4169 **A.3.2 Arithmetic functions**

4170 All of the following functions SHALL take two arguments of the specified *data-type*, integer or
 4171 double, and SHALL return an element of integer or double data-type, respectively. However, the
 4172 "add" functions MAY take more than two arguments. Each function evaluation SHALL proceed as
 4173 specified by their logical counterparts in IEEE 754 [IEEE 754]. In an expression that contains any
 4174 of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
 4175 "Indeterminate". In the case of the divide functions, if the divisor is zero, then the function SHALL
 4176 evaluate to "Indeterminate".

- 4177 • urn:oasis:names:tc:xacml:1.0:function:integer-add
 4178 This function MAY have two or more arguments.
- 4179 • urn:oasis:names:tc:xacml:1.0:function:double-add
 4180 This function MAY have two or more arguments.
- 4181 • urn:oasis:names:tc:xacml:1.0:function:integer-subtract
- 4182 • urn:oasis:names:tc:xacml:1.0:function:double-subtract
- 4183 • urn:oasis:names:tc:xacml:1.0:function:integer-multiply
- 4184 • urn:oasis:names:tc:xacml:1.0:function:double-multiply
- 4185 • urn:oasis:names:tc:xacml:1.0:function:integer-divide

4186 • urn:oasis:names:tc:xacml:1.0:function:double-divide

4187 • urn:oasis:names:tc:xacml:1.0:function:integer-mod

4188 The following functions SHALL take a single argument of the specified *data-type*. The round and

4189 floor functions SHALL take a single argument of data-type

4190 "http://www.w3.org/2001/XMLSchema#double" and return a value of the data-type

4191 "http://www.w3.org/2001/XMLSchema#double".

4192 • urn:oasis:names:tc:xacml:1.0:function:integer-abs

4193 • urn:oasis:names:tc:xacml:1.0:function:double-abs

4194 • urn:oasis:names:tc:xacml:1.0:function:round

4195 • urn:oasis:names:tc:xacml:1.0:function:floor

4196 **A.3.3 String conversion functions**

4197 The following functions convert between values of the data-type

4198 "http://www.w3.org/2001/XMLSchema#string" primitive types.

4199 • urn:oasis:names:tc:xacml:1.0:function:string-normalize-space

4200 This function SHALL take one argument of data-type

4201 "http://www.w3.org/2001/XMLSchema#string" and SHALL normalize the value by stripping
4202 off all leading and trailing white space characters.

4203 • urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case

4204 This function SHALL take one argument of data-type

4205 "http://www.w3.org/2001/XMLSchema#string" and SHALL normalize the value by
4206 converting each upper case character to its lower case equivalent.

4207 **A.3.4 Numeric data-type conversion functions**

4208 The following functions convert between the data-type

4209 "http://www.w3.org/2001/XMLSchema#integer" and "http://www.w3.org/2001/XMLSchema#double"
4210 primitive types.

4211 • urn:oasis:names:tc:xacml:1.0:function:double-to-integer

4212 This function SHALL take one argument of data-type

4213 "http://www.w3.org/2001/XMLSchema#double" and SHALL truncate its numeric value to a
4214 whole number and return an element of data-type
4215 "http://www.w3.org/2001/XMLSchema#integer".

4216 • urn:oasis:names:tc:xacml:1.0:function:integer-to-double

4217 This function SHALL take one argument of data-type

4218 "http://www.w3.org/2001/XMLSchema#integer" and SHALL promote its value to an element
4219 of data-type "http://www.w3.org/2001/XMLSchema#double" with the same numeric value.

4220 **A.3.5 Logical functions**

4221 This section contains the specification for logical functions that operate on arguments of data-type

4222 "http://www.w3.org/2001/XMLSchema#boolean".

- 4223 • urn:oasis:names:tc:xacml:1.0:function:or
- 4224 This function SHALL return "False" if it has no arguments and SHALL return "True" if at
 4225 least one of its arguments evaluates to "True". The order of evaluation SHALL be from first
 4226 argument to last. The evaluation SHALL stop with a result of "True" if any argument
 4227 evaluates to "True", leaving the rest of the arguments unevaluated.
- 4228 • urn:oasis:names:tc:xacml:1.0:function:and
- 4229 This function SHALL return "True" if it has no arguments and SHALL return "False" if one of
 4230 its arguments evaluates to "False". The order of evaluation SHALL be from first argument
 4231 to last. The evaluation SHALL stop with a result of "False" if any argument evaluates to
 4232 "False", leaving the rest of the arguments unevaluated.
- 4233 • urn:oasis:names:tc:xacml:1.0:function:n-of
- 4234 The first argument to this function SHALL be of data-type
 4235 <http://www.w3.org/2001/XMLSchema#integer>. The remaining arguments SHALL be of
 4236 data-type <http://www.w3.org/2001/XMLSchema#boolean>. The first argument specifies the
 4237 minimum number of the remaining arguments that MUST evaluate to "True" for the
 4238 expression to be considered "True". If the first argument is 0, the result SHALL be "True".
 4239 If the number of arguments after the first one is less than the value of the first argument,
 4240 then the expression SHALL result in "Indeterminate". The order of evaluation SHALL be:
 4241 first evaluate the integer value, then evaluate each subsequent argument. The evaluation
 4242 SHALL stop and return "True" if the specified number of arguments evaluate to "True". The
 4243 evaluation of arguments SHALL stop if it is determined that evaluating the remaining
 4244 arguments will not satisfy the requirement.
- 4245 • urn:oasis:names:tc:xacml:1.0:function:not
- 4246 This function SHALL take one argument of data-type
 4247 "<http://www.w3.org/2001/XMLSchema#boolean>". If the argument evaluates to "True", then
 4248 the result of the expression SHALL be "False". If the argument evaluates to "False", then
 4249 the result of the expression SHALL be "True".
- 4250 Note: When evaluating and, or, or n-of, it MAY NOT be necessary to attempt a full evaluation of
 4251 each argument in order to determine whether the evaluation of the argument would result in
 4252 "Indeterminate". Analysis of the argument regarding the availability of its attributes, or other
 4253 analysis regarding errors, such as "divide-by-zero", may render the argument error free. Such
 4254 arguments occurring in the expression in a position after the evaluation is stated to stop need not
 4255 be processed.

4256 **A.3.6 Numeric comparison functions**

4257 These functions form a minimal set for comparing two numbers, yielding a Boolean result. They
 4258 SHALL comply with the rules governed by IEEE 754 [IEEE 754].

- 4259 • urn:oasis:names:tc:xacml:1.0:function:integer-greater-than
- 4260 • urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal
- 4261 • urn:oasis:names:tc:xacml:1.0:function:integer-less-than
- 4262 • urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal
- 4263 • urn:oasis:names:tc:xacml:1.0:function:double-greater-than
- 4264 • urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal

- 4265 • urn:oasis:names:tc:xacml:1.0:function:double-less-than
- 4266 • urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal

4267 **A.3.7 Date and time arithmetic functions**

4268 These functions perform arithmetic operations with date and time.

- 4269 • urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration

4270 This function SHALL take two arguments, the first SHALL be of data-type
 4271 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be of data-type
 4272 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration". It SHALL
 4273 return a result of "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL
 4274 return the value by adding the second argument to the first argument according to the
 4275 specification of adding durations to date and time [XS Appendix E].

- 4276 • urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration

4277 This function SHALL take two arguments, the first SHALL be a
 4278 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a
 4279 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It
 4280 SHALL return a result of "http://www.w3.org/2001/XMLSchema#dateTime". This function
 4281 SHALL return the value by adding the second argument to the first argument according to
 4282 the specification of adding durations to date and time [XS Appendix E].

- 4283 • urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-dayTimeDuration

4284 This function SHALL take two arguments, the first SHALL be a
 4285 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a
 4286 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration". It SHALL
 4287 return a result of "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument
 4288 is a positive duration, then this function SHALL return the value by adding the
 4289 corresponding negative duration, as per the specification [XS Appendix E]. If the second
 4290 argument is a negative duration, then the result SHALL be as if the function
 4291 "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration" had been applied
 4292 to the corresponding positive duration.

- 4293 • urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-yearMonthDuration

4294 This function SHALL take two arguments, the first SHALL be a
 4295 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a
 4296 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It
 4297 SHALL return a result of "http://www.w3.org/2001/XMLSchema#dateTime". If the second
 4298 argument is a positive duration, then this function SHALL return the value by adding the
 4299 corresponding negative duration, as per the specification [XS Appendix E]. If the second
 4300 argument is a negative duration, then the result SHALL be as if the function
 4301 "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration" had been
 4302 applied to the corresponding positive duration.

- 4303 • urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration

4304 This function SHALL take two arguments, the first SHALL be a
 4305 "http://www.w3.org/2001/XMLSchema#date" and the second SHALL be a
 4306 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It
 4307 SHALL return a result of "http://www.w3.org/2001/XMLSchema#date". This function
 4308 SHALL return the value by adding the second argument to the first argument according to
 4309 the specification of adding duration to date [XS Appendix E].

- 4310 • urn:oasis:names:tc:xacml:1.0:function:date-subtract-yearMonthDuration
- 4311 This function SHALL take two arguments, the first SHALL be a
4312 “http://www.w3.org/2001/XMLSchema#date” and the second SHALL be a
4313 “http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration”. It
4314 SHALL return a result of “http://www.w3.org/2001/XMLSchema#date”. If the second
4315 argument is a positive duration, then this function SHALL return the value by adding the
4316 corresponding negative duration, as per the specification [XS Appendix E]. If the second
4317 argument is a negative duration, then the result SHALL be as if the function
4318 “urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration” had been applied to
4319 the corresponding positive duration.

4320 **A.3.8 Non-numeric comparison functions**

4321 These functions perform comparison operations on two arguments of non-numerical types.

- 4322 • urn:oasis:names:tc:xacml:1.0:function:string-greater-than
- 4323 This function SHALL take two arguments of data-type
4324 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
4325 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return “True” if and only if the
4326 arguments are compared byte by byte and, after an initial prefix of corresponding bytes
4327 from both arguments that are considered equal by
4328 “urn:oasis:names:tc:xacml:1.0:function:integer-equal”, the next byte by byte comparison is
4329 such that the byte from the first argument is greater than the byte from the second
4330 argument by the use of the function “urn:oasis:names:tc:xacml:2.0:function:integer-greater-
4331 then”. Otherwise, it SHALL return “False”.
- 4332 • urn:oasis:names:tc:xacml:1.0:function:string-greater-than-or-equal
- 4333 This function SHALL take two arguments of data-type
4334 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
4335 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated
4336 with the logical function “urn:oasis:names:tc:xacml:1.0:function:or” with two arguments
4337 containing the functions “urn:oasis:names:tc:xacml:1.0:function:string-greater-than” and
4338 “urn:oasis:names:tc:xacml:1.0:function:string-equal” containing the original arguments
- 4339 • urn:oasis:names:tc:xacml:1.0:function:string-less-than
- 4340 This function SHALL take two arguments of data-type
4341 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
4342 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return “True” if and only if the
4343 arguments are compared byte by byte and, after an initial prefix of corresponding bytes
4344 from both arguments that are considered equal by
4345 “urn:oasis:names:tc:xacml:1.0:function:integer-equal”, the next byte by byte comparison is
4346 such that the byte from the first argument is less than the byte from the second argument
4347 by the use of the function “urn:oasis:names:tc:xacml:1.0:function:integer-less-than”.
4348 Otherwise, it SHALL return “False”.
- 4349 • urn:oasis:names:tc:xacml:1.0:function:string-less-than-or-equal
- 4350 This function SHALL take two arguments of data-type
4351 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
4352 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated
4353 with the function “urn:oasis:names:tc:xacml:1.0:function:or” with two arguments containing
4354 the functions “urn:oasis:names:tc:xacml:1.0:function:string-less-than” and
4355 “urn:oasis:names:tc:xacml:1.0:function:string-equal” containing the original arguments.

- 4356 • urn:oasis:names:tc:xacml:1.0:function:time-greater-than
- 4357 This function SHALL take two arguments of data-type
 4358 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
 4359 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
 4360 first argument is greater than the second argument according to the order relation specified
 4361 for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8]. Otherwise, it SHALL
 4362 return "False". Note: it is illegal to compare a time that includes a time-zone value with one
 4363 that does not. In such cases, the time-in-range function should be used.
- 4364 • urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal
- 4365 This function SHALL take two arguments of data-type
 4366 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
 4367 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
 4368 first argument is greater than or equal to the second argument according to the order
 4369 relation specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].
 4370 Otherwise, it SHALL return "False". Note: it is illegal to compare a time that includes a
 4371 time-zone value with one that does not. In such cases, the time-in-range function should
 4372 be used.
- 4373 • urn:oasis:names:tc:xacml:1.0:function:time-less-than
- 4374 This function SHALL take two arguments of data-type
 4375 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
 4376 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
 4377 first argument is less than the second argument according to the order relation specified for
 4378 "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8]. Otherwise, it SHALL
 4379 return "False". Note: it is illegal to compare a time that includes a time-zone value with one
 4380 that does not. In such cases, the time-in-range function should be used.
- 4381 • urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal
- 4382 This function SHALL take two arguments of data-type
 4383 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
 4384 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
 4385 first argument is less than or equal to the second argument according to the order relation
 4386 specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8]. Otherwise, it
 4387 SHALL return "False". Note: it is illegal to compare a time that includes a time-zone value
 4388 with one that does not. In such cases, the time-in-range function should be used.
- 4389 • urn:oasis:names:tc:xacml:1.0:function:time-in-range
- 4390 This function SHALL take three arguments of data-type
 4391 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
 4392 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
 4393 argument falls in the range defined inclusively by the second and third arguments.
 4394 Otherwise, it SHALL return "False". Regardless of its value, the third argument SHALL be
 4395 interpreted as a time that is equal to, or later than by less than twenty-four hours, the
 4396 second argument. If no time zone is provided for the first argument, it SHALL use the
 4397 default time zone at the context handler. If no time zone is provided for the second or third
 4398 arguments, then they SHALL use the time zone from the first argument.
- 4399 • urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than
- 4400 This function SHALL take two arguments of data-type
 4401 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
 4402 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the

4403 first argument is greater than the second argument according to the order relation specified
4404 for “http://www.w3.org/2001/XMLSchema#dateTime” by [XF Section 3.2.7]. Otherwise, it
4405 SHALL return “False”. Note: if a dateTime value does not include a time-zone value, then
4406 an implicit time-zone value SHALL be assigned, as described in [XF].

4407 • urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal

4408 This function SHALL take two arguments of data-type
4409 “http://www.w3.org/2001/XMLSchema#dateTime” and SHALL return an
4410 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return “True” if and only if the
4411 first argument is greater than or equal to the second argument according to the order
4412 relation specified for “http://www.w3.org/2001/XMLSchema#dateTime” by [XF Section
4413 3.2.7]. Otherwise, it SHALL return “False”. Note: if a dateTime value does not include a
4414 time-zone value, then an implicit time-zone value SHALL be assigned, as described in
4415 [XF].

4416 • urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than

4417 This function SHALL take two arguments of data-type
4418 “http://www.w3.org/2001/XMLSchema#dateTime” and SHALL return an
4419 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return “True” if and only if the
4420 first argument is less than the second argument according to the order relation specified for
4421 “http://www.w3.org/2001/XMLSchema#dateTime” by [XF Section 3.2.7]. Otherwise, it
4422 SHALL return “False”. Note: if a dateTime value does not include a time-zone value, then
4423 an implicit time-zone value SHALL be assigned, as described in [XF].

4424 • urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal

4425 This function SHALL take two arguments of data-type
4426 “http://www.w3.org/2001/XMLSchema#dateTime” and SHALL return an
4427 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return “True” if and only if the
4428 first argument is less than or equal to the second argument according to the order relation
4429 specified for “http://www.w3.org/2001/XMLSchema#dateTime” by [XF Section 3.2.7].
4430 Otherwise, it SHALL return “False”. Note: if a dateTime value does not include a time-zone
4431 value, then an implicit time-zone value SHALL be assigned, as described in [XF].

4432 • urn:oasis:names:tc:xacml:1.0:function:date-greater-than

4433 This function SHALL take two arguments of data-type
4434 “http://www.w3.org/2001/XMLSchema#date” and SHALL return an
4435 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return “True” if and only if the
4436 first argument is greater than the second argument according to the order relation specified
4437 for “http://www.w3.org/2001/XMLSchema#date” by [XF Section 3.2.9]. Otherwise, it SHALL
4438 return “False”. Note: if a date value does not include a time-zone value, then an implicit
4439 time-zone value SHALL be assigned, as described in [XF].

4440 • urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal

4441 This function SHALL take two arguments of data-type
4442 “http://www.w3.org/2001/XMLSchema#date” and SHALL return an
4443 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return “True” if and only if the
4444 first argument is greater than or equal to the second argument according to the order
4445 relation specified for “http://www.w3.org/2001/XMLSchema#date” by [XF Section 3.2.9].
4446 Otherwise, it SHALL return “False”. Note: if a date value does not include a time-zone
4447 value, then an implicit time-zone value SHALL be assigned, as described in [XF].

4448 • urn:oasis:names:tc:xacml:1.0:function:date-less-than

4449 This function SHALL take two arguments of data-type
4450 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4451 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4452 first argument is less than the second argument according to the order relation specified for
4453 "http://www.w3.org/2001/XMLSchema#date" by [XF Section 3.2.9]. Otherwise, it SHALL
4454 return "False". Note: if a date value does not include a time-zone value, then an implicit
4455 time-zone value SHALL be assigned, as described in [XF].

- 4456 • urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal

4457 This function SHALL take two arguments of data-type
4458 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4459 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4460 first argument is less than or equal to the second argument according to the order relation
4461 specified for "http://www.w3.org/2001/XMLSchema#date" by [XF Section 3.2.9]. Otherwise,
4462 it SHALL return "False". Note: if a date value does not include a time-zone value, then an
4463 implicit time-zone value SHALL be assigned, as described in [XF].

4464 A.3.9 String functions

4465 The following functions operate on strings and URIs.

- 4466 • urn:oasis:names:tc:xacml:2.0:function:string-concatenate

4467

4468 This function SHALL take two or more arguments of data-type
4469 "http://www.w3.org/2001/XMLSchema#string" and SHALL return a
4470 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the concatenation, in
4471 order, of the arguments.

- 4472 • urn:oasis:names:tc:xacml:2.0:function:url-string-concatenate

4473 This function SHALL take one argument of data-type
4474 "http://www.w3.org/2001/XMLSchema#anyURI" and one or more arguments of type
4475 "http://www.w3.org/2001/XMLSchema#string", and SHALL return a
4476 "http://www.w3.org/2001/XMLSchema#anyURI". The result SHALL be the URI constructed
4477 by appending, in order, the "string" arguments to the "anyURI" argument.

4478 A.3.10 Bag functions

4479 These functions operate on a **bag** of '*type*' values, where *type* is one of the primitive data-types.
4480 Some additional conditions defined for each function below SHALL cause the expression to
4481 evaluate to "Indeterminate".

- 4482 • urn:oasis:names:tc:xacml:1.0:function:type-one-and-only

4483 This function SHALL take a **bag** of '*type*' values as an argument and SHALL return a value
4484 of '*type*'. It SHALL return the only value in the **bag**. If the **bag** does not have one and only
4485 one value, then the expression SHALL evaluate to "Indeterminate".

- 4486 • urn:oasis:names:tc:xacml:1.0:function:type-bag-size

4487 This function SHALL take a **bag** of '*type*' values as an argument and SHALL return an
4488 "http://www.w3.org/2001/XMLSchema#integer" indicating the number of values in the **bag**.

- 4489 • urn:oasis:names:tc:xacml:1.0:function:type-is-in

4490 This function SHALL take an argument of 'type' as the first argument and a **bag** of *type*
4491 values as the second argument and SHALL return an
4492 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL evaluate to "True" if
4493 and only if the first argument matches by the "urn:oasis:names:tc:xacml:x.x:function:type-
4494 equal" any value in the **bag**. Otherwise, it SHALL return "False".

4495 • urn:oasis:names:tc:xacml:1.0:function:type-bag

4496 This function SHALL take any number of arguments of 'type' and return a **bag** of 'type'
4497 values containing the values of the arguments. An application of this function to zero
4498 arguments SHALL produce an empty **bag** of the specified data-type.

4499 **A.3.11 Set functions**

4500 These functions operate on **bags** mimicking sets by eliminating duplicate elements from a **bag**.

4501 • urn:oasis:names:tc:xacml:1.0:function:type-intersection

4502 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL
4503 return a **bag** of 'type' values such that it contains only elements that are common between
4504 the two **bags**, which is determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal".
4505 No duplicates, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal",
4506 SHALL exist in the result.

4507 • urn:oasis:names:tc:xacml:1.0:function:type-at-least-one-member-of

4508 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL
4509 return a "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL evaluate to
4510 "True" if and only if at least one element of the first argument is contained in the second
4511 argument as determined by "urn:oasis:names:tc:xacml:x.x:function:type-is-in".

4512 • urn:oasis:names:tc:xacml:1.0:function:type-union

4513 This function SHALL take two arguments that are both a **bag** of 'type' values. The
4514 expression SHALL return a **bag** of 'type' such that it contains all elements of both **bags**.
4515 No duplicates, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal",
4516 SHALL exist in the result.

4517 • urn:oasis:names:tc:xacml:1.0:function:type-subset

4518 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL
4519 return a "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and
4520 only if the first argument is a subset of the second argument. Each argument SHALL be
4521 considered to have had its duplicates removed, as determined by
4522 "urn:oasis:names:tc:xacml:x.x:function:type-equal", before the subset calculation.

4523 • urn:oasis:names:tc:xacml:1.0:function:type-set-equals

4524 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL
4525 return a "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return the result of
4526 applying "urn:oasis:names:tc:xacml:1.0:function:and" to the application of
4527 "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the first and second arguments and
4528 the application of "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the second and
4529 first arguments.

4530

A.3.12 Higher-order bag functions

4531 This section describes functions in XACML that perform operations on **bags** such that functions
4532 may be applied to the **bags** in general.

4533 In this section, a general-purpose functional language called Haskell [**Haskell**] is used to formally
4534 specify the semantics of these functions. Although the English description is adequate, a formal
4535 specification of the semantics is helpful.

4536 For a quick summary, in the following Haskell notation, a function definition takes the form of
4537 clauses that are applied to patterns of structures, namely lists. The symbol "[]" denotes the empty
4538 list, whereas the expression "(x:xs)" matches against an argument of a non-empty list of which "x"
4539 represents the first element of the list, and "xs" is the rest of the list, which may be an empty list.
4540 We use the Haskell notion of a list, which is an ordered collection of elements, to model the XACML
4541 **bags** of values.

4542 A simple Haskell definition of a familiar function "urn:oasis:names:tc:xacml:1.0:function:and" that
4543 takes a list of values of type Boolean is defined as follows:

```
4544     and:: [Bool]  -> Bool
4545     and []       = True
4546     and (x:xs)  = x && (and xs)
```

4547 The first definition line denoted by a ":" formally describes the data-type of the function, which takes
4548 a list of Booleans, denoted by "[Bool]", and returns a Boolean, denoted by "Bool". The second
4549 definition line is a clause that states that the function "and" applied to the empty list is "True". The
4550 third definition line is a clause that states that for a non-empty list, such that the first element is "x",
4551 which is a value of data-type Bool, the function "and" applied to x SHALL be combined with, using
4552 the logical conjunction function, which is denoted by the infix symbol "&&", the result of recursively
4553 applying the function "and" to the rest of the list. Of course, an application of the "and" function is
4554 "True" if and only if the list to which it is applied is empty or every element of the list is "True". For
4555 example, the evaluation of the following Haskell expressions,

```
4556     (and []), (and [True]), (and [True,True]), (and [True,True,False])
```

4557 evaluate to "True", "True", "True", and "False", respectively.

4558 • urn:oasis:names:tc:xacml:1.0:function:any-of

4559 This function applies a Boolean function between a specific primitive value and a **bag** of
4560 values, and SHALL return "True" if and only if the predicate is "True" for at least one
4561 element of the **bag**.

4562 This function SHALL take three arguments. The first argument SHALL be an
4563 <xacml:Function> element that names a Boolean function that takes two arguments of
4564 primitive types. The second argument SHALL be a value of a primitive data-type. The third
4565 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated
4566 as if the function named in the <xacml:Function> argument were applied to the second
4567 argument and each element of the third argument (the **bag**) and the results are combined
4568 with "urn:oasis:names:tc:xacml:1.0:function:or".

4569 In Haskell, the semantics of this operation are as follows:

```
4570     any_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
4571     any_of f a []           = False
4572     any_of f a (x:xs)      = (f a x) || (any_of f a xs)
```

4573 In the above notation, “f” is the function to be applied, “a” is the primitive value, and “(x:xs)”
4574 represents the first element of the list as “x” and the rest of the list as “xs”.

4575 For example, the following expression SHALL return "True":

```
4576 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">  
4577 <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>  
4578 <AttributeValue  
4579 DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>  
4580 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4581 <AttributeValue  
4582 DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>  
4583 <AttributeValue  
4584 DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>  
4585 <AttributeValue  
4586 DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>  
4587 <AttributeValue  
4588 DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>  
4589 </Apply>  
4590 </Apply>
```

4591 This expression is "True" because the first argument is equal to at least one of the
4592 elements of the **bag**, according to the function.

4593 • urn:oasis:names:tc:xacml:1.0:function:all-of

4594 This function applies a Boolean function between a specific primitive value and a **bag** of
4595 values, and returns "True" if and only if the predicate is "True" for every element of the **bag**.

4596 This function SHALL take three arguments. The first argument SHALL be an
4597 <xacml:Function> element that names a Boolean function that takes two arguments of
4598 primitive types. The second argument SHALL be a value of a primitive data-type. The third
4599 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated
4600 as if the function named in the <xacml:Function> argument were applied to the second
4601 argument and each element of the third argument (the **bag**) and the results were combined
4602 using “urn:oasis:names:tc:xacml:1.0:function:and”.

4603 In Haskell, the semantics of this operation are as follows:

```
4604 all_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool  
4605 all_of f a [] = True  
4606 all_of f a (x:xs) = (f a x) && (all_of f a xs)
```

4607 In the above notation, “f” is the function to be applied, “a” is the primitive value, and “(x:xs)”
4608 represents the first element of the list as “x” and the rest of the list as “xs”.

4609 For example, the following expression SHALL evaluate to "True":

```
4610 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">  
4611 <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater"/>  
4612 <AttributeValue  
4613 DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>  
4614 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">  
4615 <AttributeValue  
4616 DataType="http://www.w3.org/2001/XMLSchema#integer">9</AttributeValue>  
4617 <AttributeValue  
4618 DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>  
4619 <AttributeValue  
4620 DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>  
4621 <AttributeValue  
4622 DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>  
4623 </Apply>  
4624 </Apply>
```

4625 This expression is "True" because the first argument (10) is greater than *all* of the elements
4626 of the **bag** (9,3,4 and 2).

4627 • urn:oasis:names:tc:xacml:1.0:function:any-of-any

4628 This function applies a Boolean function between each element of a **bag** of values and
4629 each element of another **bag** of values, and returns "True" if and only if the predicate is
4630 "True" for at least one comparison.

4631 This function SHALL take three arguments. The first argument SHALL be an
4632 <xacml:Function> element that names a Boolean function that takes two arguments of
4633 primitive types. The second argument SHALL be a **bag** of a primitive data-type. The third
4634 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated
4635 as if the function named in the <xacml:Function> argument were applied between
4636 every element of the second argument and every element of the third argument and the
4637 results were combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics
4638 are that the result of the expression SHALL be "True" if and only if the applied predicate is
4639 "True" for *at least one* comparison of elements from the two **bags**.

4640 In Haskell, taking advantage of the "any_of" function defined above, the semantics of the
4641 "any_of_any" function are as follows:

```
4642 any_of_any :: ( a -> b -> Bool ) -> [a]-> [b] -> Bool
4643 any_of_any f [] ys = False
4644 any_of_any f (x:xs) ys = (any_of f x ys) || (any_of_any f xs ys)
```

4645 In the above notation, "f" is the function to be applied and "(x:xs)" represents the first
4646 element of the list as "x" and the rest of the list as "xs".

4647 For example, the following expression SHALL evaluate to "True":

```
4648 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">
4649   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
4650   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4651     <AttributeValue
4652       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4653     <AttributeValue
4654       DataType="http://www.w3.org/2001/XMLSchema#string">Mary</AttributeValue>
4655   </Apply>
4656   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4657     <AttributeValue
4658       DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4659     <AttributeValue
4660       DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4661     <AttributeValue
4662       DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4663     <AttributeValue
4664       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4665   </Apply>
4666 </Apply>
```

4667 This expression is "True" because at least one of the elements of the first **bag**, namely
4668 "Ringo", is equal to at least one of the elements of the second **bag**.

4669 • urn:oasis:names:tc:xacml:1.0:function:all-of-any

4670 This function applies a Boolean function between the elements of two **bags**. The
4671 expression SHALL be "True" if and only if the supplied predicate is 'True' between each
4672 element of the first **bag** and any element of the second **bag**.

4673 This function SHALL take three arguments. The first argument SHALL be an
4674 <xacml:Function> element that names a Boolean function that takes two arguments of
4675 primitive types. The second argument SHALL be a **bag** of a primitive data-type. The third
4676 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated
4677 as if the "urn:oasis:names:tc:xacml:1.0:function:any-of" function had been applied to each
4678 value of the first **bag** and the whole of the second **bag** using the supplied xacml:Function,
4679 and the results were then combined using "urn:oasis:names:tc:xacml:1.0:function:and".

4680 In Haskell, taking advantage of the "any_of" function defined in Haskell above, the
4681 semantics of the "all_of_any" function are as follows:

```
4682 all_of_any :: ( a -> b -> Bool ) -> [a]-> [b] -> Bool  
4683 all_of_any f [] ys = True  
4684 all_of_any f (x:xs) ys = (any_of f x ys) && (all_of_any f xs ys)
```

4685 In the above notation, "f" is the function to be applied and "(x:xs)" represents the first
4686 element of the list as "x" and the rest of the list as "xs".

4687 For example, the following expression SHALL evaluate to "True":

```
4688 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-any">  
4689 <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater"/>  
4690 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">  
4691 <AttributeValue  
4692 DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>  
4693 <AttributeValue  
4694 DataType="http://www.w3.org/2001/XMLSchema#integer">20</AttributeValue>  
4695 </Apply>  
4696 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">  
4697 <AttributeValue  
4698 DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>  
4699 <AttributeValue  
4700 DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>  
4701 <AttributeValue  
4702 DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>  
4703 <AttributeValue  
4704 DataType="http://www.w3.org/2001/XMLSchema#integer">19</AttributeValue>  
4705 </Apply>  
4706 </Apply>
```

4707 This expression is "True" because each of the elements of the first **bag** is greater than at
4708 least one of the elements of the second **bag**.

4709 • urn:oasis:names:tc:xacml:1.0:function:any-of-all

4710 This function applies a Boolean function between the elements of two **bags**. The
4711 expression SHALL be "True" if and only if the supplied predicate is "True" between each
4712 element of the second **bag** and any element of the first **bag**.

4713 This function SHALL take three arguments. The first argument SHALL be an
4714 <xacml:Function> element that names a Boolean function that takes two arguments of
4715 primitive types. The second argument SHALL be a **bag** of a primitive data-type. The third
4716 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated
4717 as if the "rn:oasis:names:tc:xacml:1.0:function:any-of" function had been applied to each
4718 value of the second **bag** and the whole of the first **bag** using the supplied xacml:Function,
4719 and the results were then combined using "urn:oasis:names:tc:xacml:1.0:function:and".

4720 In Haskell, taking advantage of the "all_of" function defined in Haskell above, the semantics
4721 of the "any_of_all" function are as follows:

```

4722         any_of_all :: ( a -> b -> Bool )   -> [a]-> [b] -> Bool
4723         any_of_all f []      ys             = False
4724         any_of_all f (x:xs) ys             = (all_of f x ys) || ( any_of_all f xs ys)

```

4725 In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first
4726 element of the list as “x” and the rest of the list as “xs”.

4727 For example, the following expression SHALL evaluate to "True":

```

4728 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-all">
4729   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater"/>
4730   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4731     <AttributeValue
4732       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4733     <AttributeValue
4734       DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4735   </Apply>
4736   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4737     <AttributeValue
4738       DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4739     <AttributeValue
4740       DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4741     <AttributeValue
4742       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4743     <AttributeValue
4744       DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4745   </Apply>
4746 </Apply>

```

4747 This expression is “True” because, for all of the values in the second **bag**, there is a value
4748 in the first **bag** that is greater.

4749 • urn:oasis:names:tc:xacml:1.0:function:all-of-all

4750 This function applies a Boolean function between the elements of two **bags**. The
4751 expression SHALL be "True" if and only if the supplied predicate is "True" between each
4752 and every element of the first **bag** collectively against all the elements of the second **bag**.

4753 This function SHALL take three arguments. The first argument SHALL be an
4754 <xacml:Function> element that names a Boolean function that takes two arguments of
4755 primitive types. The second argument SHALL be a **bag** of a primitive data-type. The third
4756 argument SHALL be a **bag** of a primitive data-type. The expression is evaluated as if the
4757 function named in the <xacml:Function> element were applied between every element
4758 of the second argument and every element of the third argument and the results were
4759 combined using “urn:oasis:names:tc:xacml:1.0:function:and”. The semantics are that the
4760 result of the expression is "True" if and only if the applied predicate is "True" for *all*
4761 elements of the first **bag** compared to *all* the elements of the second **bag**.

4762 In Haskell, taking advantage of the “all_of” function defined in Haskell above, the semantics
4763 of the “all_of_all” function is as follows:

```

4764         all_of_all :: ( a -> b -> Bool )   -> [a] -> [b] -> Bool
4765         all_of_all f []      ys             = True
4766         all_of_all f (x:xs) ys             = (all_of f x ys) && (all_of_all f xs ys)

```

4767 In the above notation, “f” is the function to be applied and “(x:xs)” represents the first
4768 element of the list as “x” and the rest of the list as “xs”.

4769 For example, the following expression SHALL evaluate to "True":

```

4770 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-all">
4771   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater"/>
4772   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4773     <AttributeValue
4774     DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>
4775     <AttributeValue
4776     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4777   </Apply>
4778   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4779     <AttributeValue
4780     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4781     <AttributeValue
4782     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4783     <AttributeValue
4784     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4785     <AttributeValue
4786     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4787   </Apply>
4788 </Apply>

```

4789 This expression is "True" because all elements of the first **bag**, "5" and "6", are each
4790 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4791 • urn:oasis:names:tc:xacml:1.0:function:map

4792 This function converts a **bag** of values to another **bag** of values.

4793 This function SHALL take two arguments. The first function SHALL be an
4794 <xacml:Function> element naming a function that takes a single argument of a primitive
4795 data-type and returns a value of a primitive data-type. The second argument SHALL be a
4796 **bag** of a primitive data-type. The expression SHALL be evaluated as if the function named
4797 in the <xacml:Function> element were applied to each element in the **bag** resulting in a
4798 **bag** of the converted value. The result SHALL be a **bag** of the primitive data-type that is
4799 returned by the function named in the <xacml:Function> element.

4800 In Haskell, this function is defined as follows:

```

4801     map:: (a -> b)  -> [a] -> [b]
4802     map f []       = []
4803     map f (x:xs)  = (f x) : (map f xs)

```

4804 In the above notation, "f" is the function to be applied and "(x:xs)" represents the first
4805 element of the list as "x" and the rest of the list as "xs".

4806 For example, the following expression,

```

4807 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:map">
4808   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-normalize-
4809   to-lower-case">
4810   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4811     <AttributeValue
4812     DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>
4813     <AttributeValue
4814     DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>
4815   </Apply>
4816 </Apply>

```

4817 evaluates to a **bag** containing "hello" and "world!".

4818

A.3.13 Regular-expression-based functions

4819 These functions operate on various types using regular expressions and evaluate to
4820 "http://www.w3.org/2001/XMLSchema#boolean".

- 4821 • urn:oasis:names:tc:xacml:1.0:function:string-regexp-match

4822 This function decides a regular expression match. It SHALL take two arguments of
4823 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4824 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4825 expression and the second argument SHALL be a general string. The function
4826 specification SHALL be that of the "xf:matches" function with the arguments reversed [XF
4827 Section 6.3.15].

- 4828 • urn:oasis:names:tc:xacml:2.0:function:anyURI-regexp-match

4829 This function decides a regular expression match. It SHALL take two arguments; the first is
4830 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4831 "http://www.w3.org/2001/XMLSchema#anyURI". It SHALL return an
4832 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4833 expression and the second argument SHALL be a URI. The function SHALL convert the
4834 second argument to type "http://www.w3.org/2001/XMLSchema#string", then apply
4835 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

- 4836 • urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match

4837 This function decides a regular expression match. It SHALL take two arguments; the first is
4838 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4839 "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress". It SHALL return an
4840 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4841 expression and the second argument SHALL be an IPv4 or IPv6 address. The function
4842 SHALL convert the second argument to type "http://www.w3.org/2001/XMLSchema#string",
4843 then apply "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

- 4844 • urn:oasis:names:tc:xacml:2.0:function:dnsName-regexp-match

4845 This function decides a regular expression match. It SHALL take two arguments; the first is
4846 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4847 "urn:oasis:names:tc:xacml:2.0:data-type:dnsName". It SHALL return an
4848 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4849 expression and the second argument SHALL be a DNS name. The function SHALL
4850 convert the second argument to type "http://www.w3.org/2001/XMLSchema#string", then
4851 apply "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

- 4852 • urn:oasis:names:tc:xacml:2.0:function:rfc822Name-regexp-match

4853 This function decides a regular expression match. It SHALL take two arguments; the first is
4854 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4855 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name". It SHALL return an
4856 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4857 expression and the second argument SHALL be an RFC 822 name. The function SHALL
4858 convert the second argument to type "http://www.w3.org/2001/XMLSchema#string", then
4859 apply "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

- 4860 • urn:oasis:names:tc:xacml:2.0:function:x500Name-regexp-match

4861 This function decides a regular expression match. It SHALL take two arguments; the first is
4862 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type

4863 "urn:oasis:names:tc:xacml:1.0:data-type:x500Name". It SHALL return an
4864 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4865 expression and the second argument SHALL be an X.500 directory name. The function
4866 SHALL convert the second argument to type "http://www.w3.org/2001/XMLSchema#string",
4867 then apply "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

4868 **A.3.14 Special match functions**

4869 These functions operate on various types and evaluate to
4870 "http://www.w3.org/2001/XMLSchema#boolean" based on the specified standard matching
4871 algorithm.

- 4872 • urn:oasis:names:tc:xacml:1.0:function:x500Name-match

4873 This function shall take two arguments of "urn:oasis:names:tc:xacml:2.0:data-
4874 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It
4875 shall return "True" if and only if the first argument matches some terminal sequence of
4876 RDNs from the second argument when compared using x500Name-equal.

- 4877 • urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match

4878 This function SHALL take two arguments, the first is of data-type
4879 "http://www.w3.org/2001/XMLSchema#string" and the second is of data-type
4880 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an
4881 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if
4882 the first argument matches the second argument according to the following specification.

4883 An RFC822 name consists of a local-part followed by "@" followed by a domain-part. The
4884 local-part is case-sensitive, while the domain-part (which is usually a DNS name) is not
4885 case-sensitive.⁴

4886 The second argument contains a complete rfc822Name. The first argument is a complete
4887 or partial rfc822Name used to select appropriate values in the second argument as follows.

4888 In order to match a particular address in the second argument, the first argument must
4889 specify the complete mail address to be matched. For example, if the first argument is
4890 "Anderson@sun.com", this matches a value in the second argument of
4891 "Anderson@sun.com" and "Anderson@SUN.COM", but not "Anne.Anderson@sun.com",
4892 "anderson@sun.com" or "Anderson@east.sun.com".

4893 In order to match any address at a particular domain in the second argument, the first
4894 argument must specify only a domain name (usually a DNS name). For example, if the first
4895 argument is "sun.com", this matches a value in the first argument of "Anderson@sun.com"
4896 or "Baxter@SUN.COM", but not "Anderson@east.sun.com".

4897 In order to match any address in a particular domain in the second argument, the first
4898 argument must specify the desired domain-part with a leading ".". For example, if the first
4899 argument is ".east.sun.com", this matches a value in the second argument of
4900 "Anderson@east.sun.com" and "anne.anderson@ISRG.EAST.SUN.COM" but not
4901 "Anderson@sun.com".

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.

4902

A.3.15 XPath-based functions

4903 This section specifies functions that take XPath expressions for arguments. An XPath expression
4904 evaluates to a *node-set*, which is a set of XML nodes that match the expression. A node or node-
4905 set is not in the formal data-type system of XACML. All comparison or other operations on node-
4906 sets are performed in isolation of the particular function specified. That is, the XPath expressions in
4907 these functions are restricted to the XACML request **context**. The `<xacml-context:Request>`
4908 element is the context node for every XPath expression. The following functions are defined:

- 4909 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-count

4910 This function SHALL take an "http://www.w3.org/2001/XMLSchema#string" as an
4911 argument, which SHALL be interpreted as an XPath expression, and evaluates to an
4912 "http://www.w3.org/2001/XMLSchema#integer". The value returned from the function
4913 SHALL be the count of the nodes within the node-set that match the given XPath
4914 expression.

- 4915 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal

4916 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,
4917 which SHALL be interpreted as XPath expressions, and SHALL return an
4918 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if any
4919 of the XML nodes in the node-set matched by the first argument equals, according to the
4920 "op:node-equal" function [XF Section 13.1.6], any of the XML nodes in the node-set
4921 matched by the second argument.

- 4922 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-match

4923 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,
4924 which SHALL be interpreted as XPath expressions and SHALL return an
4925 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if
4926 one of the following two conditions is satisfied: (1) Any of the XML nodes in the node-set
4927 matched by the first argument is equal, according to "op:node-equal" [XF Section 13.1.6],
4928 to any of the XML nodes in the node-set matched by the second argument; (2) any attribute
4929 and element node below any of the XML nodes in the node-set matched by the first
4930 argument is equal, according to "op:node-equal" [XF Section 13.1.6], to any of the XML
4931 nodes in the node-set matched by the second argument.

4932 NOTE: The first condition is equivalent to "xpath-node-equal", and guarantees that "xpath-node-
4933 equal" is a special case of "xpath-node-match".

4934 A.3.16 Extension functions and primitive types

4935 Functions and primitive types are specified by string identifiers allowing for the introduction of
4936 functions in addition to those specified by XACML. This approach allows one to extend the XACML
4937 module with special functions and special primitive data-types.

4938 In order to preserve the integrity of the XACML evaluation strategy, the result of an extension
4939 function SHALL depend only on the values of its arguments. Global and hidden parameters SHALL
4940 NOT affect the evaluation of an expression. Functions SHALL NOT have side effects, as
4941 evaluation order cannot be guaranteed in a standard way.

4942 Appendix B. XACML identifiers (normative)

4943 This section defines standard identifiers for commonly used entities.

4944 B.1. XACML namespaces

4945 There are currently two defined XACML namespaces.

4946 Policies are defined using this identifier.

4947 `urn:oasis:names:tc:xacml:2.0:policy:schema:os`

4948 Request and response **contexts** are defined using this identifier.

4949 `urn:oasis:names:tc:xacml:2.0:context:schema:os`

4950 B.2. Access subject categories

4951 This identifier indicates the system entity that initiated the **access** request. That is, the initial entity
4952 in a request chain. If **subject** category is not specified, this is the default value.

4953 `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

4954 This identifier indicates the system entity that will receive the results of the request (used when it is
4955 distinct from the access-subject).

4956 `urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject`

4957 This identifier indicates a system entity through which the **access** request was passed. There may
4958 be more than one. No means is provided to specify the order in which they passed the message.

4959 `urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject`

4960 This identifier indicates a system entity associated with a local or remote codebase that generated
4961 the request. Corresponding **subject attributes** might include the URL from which it was loaded
4962 and/or the identity of the code-signer. There may be more than one. No means is provided to
4963 specify the order in which they processed the request.

4964 `urn:oasis:names:tc:xacml:1.0:subject-category:codebase`

4965 This identifier indicates a system entity associated with the computer that initiated the **access**
4966 request. An example would be an IPsec identity.

4967 `urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine`

4968 B.3. Data-types

4969 The following identifiers indicate data-types that are defined in Section A.2.

4970 `urn:oasis:names:tc:xacml:1.0:data-type:x500Name.`

4971 `urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name`

4972 `urn:oasis:names:tc:xacml:2.0:data-type:ipAddress`

4973 `urn:oasis:names:tc:xacml:2.0:data-type:dnsName`

4974 The following data-type identifiers are defined by XML Schema **[XS]**.

4975 `http://www.w3.org/2001/XMLSchema#string`

4976 `http://www.w3.org/2001/XMLSchema#boolean`

4977 <http://www.w3.org/2001/XMLSchema#integer>
 4978 <http://www.w3.org/2001/XMLSchema#double>
 4979 <http://www.w3.org/2001/XMLSchema#time>
 4980 <http://www.w3.org/2001/XMLSchema#date>
 4981 <http://www.w3.org/2001/XMLSchema#dateTime>
 4982 <http://www.w3.org/2001/XMLSchema#anyURI>
 4983 <http://www.w3.org/2001/XMLSchema#hexBinary>
 4984 <http://www.w3.org/2001/XMLSchema#base64Binary>
 4985 The following data-type identifiers correspond to the `dayTimeDuration` and `yearMonthDuration`
 4986 data-types defined in [XF Sections 8.2.2 and 8.2.1, respectively].
 4987 <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration>
 4988 <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration>

4989 **B.4. Subject attributes**

4990 These identifiers indicate **attributes** of a **subject**. When used, they SHALL appear within a
 4991 `<Subject>` element of the request **context**. They SHALL be accessed by means of a
 4992 `<SubjectAttributeDesignator>` element, or an `<AttributeSelector>` element that points
 4993 into a `<Subject>` element of the request **context**.

4994 At most one of each of these attributes is associated with each subject. Each attribute associated
 4995 with authentication included within a single `<Subject>` element relates to the same authentication
 4996 event.

4997 This identifier indicates the name of the **subject**. The default format is
 4998 “<http://www.w3.org/2001/XMLSchema#string>”. To indicate other formats, use the `DataType`
 4999 attributes listed in B.3

5000 `urn:oasis:names:tc:xacml:1.0:subject:subject-id`

5001 This identifier indicates the **subject** category. “access-subject” is the default value.

5002 `urn:oasis:names:tc:xacml:1.0:subject-category`

5003 This identifier indicates the security domain of the **subject**. It identifies the administrator and policy
 5004 that manages the name-space in which the **subject** id is administered.

5005 `urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier`

5006 This identifier indicates a public key used to confirm the **subject's** identity.

5007 `urn:oasis:names:tc:xacml:1.0:subject:key-info`

5008 This identifier indicates the time at which the **subject** was authenticated.

5009 `urn:oasis:names:tc:xacml:1.0:subject:authentication-time`

5010 This identifier indicates the method used to authenticate the **subject**.

5011 `urn:oasis:names:tc:xacml:1.0:subject:authn-locality:authentication-method`

5012 This identifier indicates the time at which the **subject** initiated the **access** request, according to the
 5013 **PEP**.

5014 `urn:oasis:names:tc:xacml:1.0:subject:request-time`

5015 This identifier indicates the time at which the **subject's** current session began, according to the
 5016 **PEP**.

5017 `urn:oasis:names:tc:xacml:1.0:subject:session-start-time`

5018 The following identifiers indicate the location where authentication credentials were activated. They
 5019 are intended to support the corresponding entities from the SAML authentication statement
 5020 **[SAML]**.

5021 This identifier indicates that the location is expressed as an IP address.
5022 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address
5023 The corresponding attribute SHALL be of data-type "http://www.w3.org/2001/XMLSchema#string".
5024 This identifier indicates that the location is expressed as a DNS name.
5025 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name
5026 The corresponding attribute SHALL be of data-type "http://www.w3.org/2001/XMLSchema#string".
5027 Where a suitable attribute is already defined in LDAP [LDAP-1, LDAP-2], the XACML identifier
5028 SHALL be formed by adding the **attribute** name to the URI of the LDAP specification. For
5029 example, the **attribute** name for the userPassword defined in the RFC 2256 SHALL be:
5030 http://www.ietf.org/rfc/rfc2256.txt#userPassword

5031 B.6. Resource attributes

5032 These identifiers indicate **attributes** of the **resource**. The corresponding **attributes** MAY appear in
5033 the <Resource> element of the request **context** and be accessed by means of a
5034 <ResourceAttributeDesignator> element, or by an <AttributeSelector> element that
5035 points into the <Resource> element of the request **context**.

5036 This **attribute** identifies the **resource** to which access is requested. If an <xacml-
5037 context:ResourceContent> element is provided, then the resource to which access is
5038 requested SHALL be all or a portion of the resource supplied in the <xacml-
5039 context:ResourceContent> element.

5040 urn:oasis:names:tc:xacml:1.0:resource:resource-id

5041 This **attribute** identifies the namespace of the top element of the contents of the <xacml-
5042 context:ResourceContent> element. In the case where the **resource** content is supplied in the
5043 request **context** and the **resource** namespace is defined in the **resource**, the PDP SHALL confirm
5044 that the namespace defined by this **attribute** is the same as that defined in the **resource**. The type
5045 of the corresponding **attribute** SHALL be "http://www.w3.org/2001/XMLSchema#anyURI".

5046 urn:oasis:names:tc:xacml:2.0:resource:target-namespace

5047 B.7. Action attributes

5048 These identifiers indicate **attributes** of the **action** being requested. When used, they SHALL
5049 appear within the <Action> element of the request **context**. They SHALL be accessed by means
5050 of an <ActionAttributeDesignator> element, or an <AttributeSelector> element that
5051 points into the <Action> element of the request **context**.

5052 This **attribute** identifies the **action** for which **access** is requested.

5053 urn:oasis:names:tc:xacml:1.0:action:action-id

5054 Where the **action** is implicit, the value of the action-id **attribute** SHALL be

5055 urn:oasis:names:tc:xacml:1.0:action:implied-action

5056 This **attribute** identifies the namespace in which the action-id **attribute** is defined.

5057 urn:oasis:names:tc:xacml:1.0:action:action-namespace

5058 B.8. Environment attributes

5059 These identifiers indicate **attributes** of the **environment** within which the **decision request** is to be
5060 evaluated. When used in the **decision request**, they SHALL appear in the <Environment>
5061 element of the request **context**. They SHALL be accessed by means of an
5062 <EnvironmentAttributeDesignator> element, or an <AttributeSelector> element that
5063 points into the <Environment> element of the request **context**.

5064 This identifier indicates the current time at the **context handler**. In practice it is the time at which
5065 the request **context** was created. For this reason, if these identifiers appear in multiple places
5066 within a <Policy> or <PolicySet>, then the same value SHALL be assigned to each occurrence
5067 in the evaluation procedure, regardless of how much time elapses between the processing of the
5068 occurrences.

5069 urn:oasis:names:tc:xacml:1.0:environment:current-time

5070 The corresponding **attribute** SHALL be of data-type
5071 "http://www.w3.org/2001/XMLSchema#time".

5072 urn:oasis:names:tc:xacml:1.0:environment:current-date

5073 The corresponding **attribute** SHALL be of data-type
5074 "http://www.w3.org/2001/XMLSchema#date".

5075 urn:oasis:names:tc:xacml:1.0:environment:current-dateTime

5076 The corresponding **attribute** SHALL be of data-type
5077 "http://www.w3.org/2001/XMLSchema#dateTime".

5078 B.9. Status codes

5079 The following status code values are defined.

5080 This identifier indicates success.

5081 urn:oasis:names:tc:xacml:1.0:status:ok

5082 This identifier indicates that all the attributes necessary to make a policy decision were not available
5083 (see Section 6.16).

5084 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

5085 This identifier indicates that some attribute value contained a syntax error, such as a letter in a
5086 numeric field.

5087 urn:oasis:names:tc:xacml:1.0:status:syntax-error

5088 This identifier indicates that an error occurred during policy evaluation. An example would be
5089 division by zero.

5090 urn:oasis:names:tc:xacml:1.0:status:processing-error

5091 B.10. Combining algorithms

5092 The deny-overrides rule-combining algorithm has the following value for the
5093 ruleCombiningAlgId attribute:

5094 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

5095 The deny-overrides policy-combining algorithm has the following value for the
5096 policyCombiningAlgId attribute:
5097 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides
5098 The permit-overrides rule-combining algorithm has the following value for the
5099 ruleCombiningAlgId attribute:
5100 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides
5101 The permit-overrides policy-combining algorithm has the following value for the
5102 policyCombiningAlgId attribute:
5103 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides
5104 The first-applicable rule-combining algorithm has the following value for the
5105 ruleCombiningAlgId attribute:
5106 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable
5107 The first-applicable policy-combining algorithm has the following value for the
5108 policyCombiningAlgId attribute:
5109 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable
5110 The only-one-applicable-policy policy-combining algorithm has the following value for the
5111 policyCombiningAlgId attribute:
5112 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-
5113 applicable
5114 The ordered-deny-overrides rule-combining algorithm has the following value for the
5115 ruleCombiningAlgId attribute:
5116 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-
5117 overrides
5118 The ordered-deny-overrides policy-combining algorithm has the following value for the
5119 policyCombiningAlgId attribute:
5120 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-
5121 overrides
5122 The ordered-permit-overrides rule-combining algorithm has the following value for the
5123 ruleCombiningAlgId attribute:
5124 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-
5125 overrides
5126 The ordered-permit-overrides policy-combining algorithm has the following value for the
5127 policyCombiningAlgId attribute:
5128 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-
5129 overrides

5130

Appendix C. Combining algorithms (normative)

5131

This section contains a description of the *rule-* and *policy-combining algorithms* specified by XACML.

5132

5133

C.1. Deny-overrides

5134

The following specification defines the "Deny-overrides" *rule-combining algorithm* of a *policy*.

5135

In the entire set of *rules* in the *policy*, if any *rule* evaluates to "Deny", then the result of the *rule* combination SHALL be "Deny". If any *rule* evaluates to "Permit" and all other *rules* evaluate to "NotApplicable", then the result of the *rule* combination SHALL be "Permit". In other words, "Deny" takes precedence, regardless of the result of evaluating any of the other *rules* in the combination. If all *rules* are found to be "NotApplicable" to the *decision request*, then the *rule* combination SHALL evaluate to "NotApplicable".

5136

5137

5138

5139

5140

5141

If an error occurs while evaluating the *target* or *condition* of a *rule* that contains an *effect* value of "Deny" then the evaluation SHALL continue to evaluate subsequent *rules*, looking for a result of "Deny". If no other *rule* evaluates to "Deny", then the combination SHALL evaluate to "Indeterminate", with the appropriate error status.

5142

5143

5144

5145

If at least one *rule* evaluates to "Permit", all other *rules* that do not have evaluation errors evaluate to "Permit" or "NotApplicable" and all *rules* that do have evaluation errors contain *effects* of "Permit", then the result of the combination SHALL be "Permit".

5146

5147

5148

The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

5149

```
Decision denyOverridesRuleCombiningAlgorithm(Rule rule[])
```

5150

```
{
```

5151

```
    Boolean atLeastOneError = false;
```

5152

```
    Boolean potentialDeny = false;
```

5153

```
    Boolean atLeastOnePermit = false;
```

5154

```
    for( i=0 ; i < lengthOf(rules) ; i++ )
```

5155

```
    {
```

5156

```
        Decision decision = evaluate(rule[i]);
```

5157

```
        if (decision == Deny)
```

5158

```
        {
```

5159

```
            return Deny;
```

5160

```
        }
```

5161

```
        if (decision == Permit)
```

5162

```
        {
```

5163

```
            atLeastOnePermit = true;
```

5164

```
            continue;
```

5165

```
        }
```

5166

```
        if (decision == NotApplicable)
```

5167

```
        {
```

5168

```
            continue;
```

5169

```
        }
```

5170

```
        if (decision == Indeterminate)
```

5171

```
        {
```

5172

```
            atLeastOneError = true;
```

5173

5174

```
            if (effect(rule[i]) == Deny)
```

5175

```
            {
```

5176

```
                potentialDeny = true;
```

5177

```
            }
```

```

5178     continue;
5179     }
5180   }
5181   if (potentialDeny)
5182   {
5183     return Indeterminate;
5184   }
5185   if (atLeastOnePermit)
5186   {
5187     return Permit;
5188   }
5189   if (atLeastOneError)
5190   {
5191     return Indeterminate;
5192   }
5193   return NotApplicable;
5194 }

```

5195 The following specification defines the "Deny-overrides" **policy-combining algorithm** of a **policy**
5196 **set**.

5197 In the entire set of **policies** in the **policy set**, if any **policy** evaluates to "Deny", then the
5198 result of the **policy** combination SHALL be "Deny". In other words, "Deny" takes
5199 precedence, regardless of the result of evaluating any of the other **policies** in the **policy**
5200 **set**. If all **policies** are found to be "NotApplicable" to the **decision request**, then the
5201 **policy set** SHALL evaluate to "NotApplicable".

5202 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is
5203 considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set**
5204 SHALL evaluate to "Deny".

5205 The following pseudo-code represents the evaluation strategy of this **policy-combining algorithm**.

```

5206 Decision denyOverridesPolicyCombiningAlgorithm(Policy policy[])
5207 {
5208   Boolean atLeastOnePermit = false;
5209   for( i=0 ; i < lengthOf(policy) ; i++ )
5210   {
5211     Decision decision = evaluate(policy[i]);
5212     if (decision == Deny)
5213     {
5214       return Deny;
5215     }
5216     if (decision == Permit)
5217     {
5218       atLeastOnePermit = true;
5219       continue;
5220     }
5221     if (decision == NotApplicable)
5222     {
5223       continue;
5224     }
5225     if (decision == Indeterminate)
5226     {
5227       return Deny;
5228     }
5229   }
5230   if (atLeastOnePermit)
5231   {
5232     return Permit;
5233   }
5234   return NotApplicable;

```

5235 }

5236 **Obligations** of the individual **policies** shall be combined as described in Section 7.14.

5237 C.2. Ordered-deny-overrides

5238 The following specification defines the "Ordered-deny-overrides" **rule-combining algorithm** of a
5239 **policy**.

5240 The behavior of this algorithm is identical to that of the Deny-overrides **rule-combining**
5241 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL
5242 match the order as listed in the **policy**.

5243 The following specification defines the "Ordered-deny-overrides" **policy-combining algorithm** of a
5244 **policy set**.

5245 The behavior of this algorithm is identical to that of the Deny-overrides **policy-combining**
5246 **algorithm** with one exception. The order in which the collection of **policies** is evaluated
5247 SHALL match the order as listed in the **policy set**.

5248 C.3. Permit-overrides

5249 The following specification defines the "Permit-overrides" **rule-combining algorithm** of a **policy**.

5250 In the entire set of **rules** in the **policy**, if any **rule** evaluates to "Permit", then the result of
5251 the **rule** combination SHALL be "Permit". If any **rule** evaluates to "Deny" and all other
5252 **rules** evaluate to "NotApplicable", then the **policy** SHALL evaluate to "Deny". In other
5253 words, "Permit" takes precedence, regardless of the result of evaluating any of the other
5254 **rules** in the **policy**. If all **rules** are found to be "NotApplicable" to the **decision request**,
5255 then the **policy** SHALL evaluate to "NotApplicable".

5256 If an error occurs while evaluating the **target** or **condition** of a **rule** that contains an **effect**
5257 of "Permit" then the evaluation SHALL continue looking for a result of "Permit". If no other
5258 **rule** evaluates to "Permit", then the **policy** SHALL evaluate to "Indeterminate", with the
5259 appropriate error status.

5260 If at least one **rule** evaluates to "Deny", all other **rules** that do not have evaluation errors
5261 evaluate to "Deny" or "NotApplicable" and all **rules** that do have evaluation errors contain
5262 an **effect** value of "Deny", then the **policy** SHALL evaluate to "Deny".

5263 The following pseudo-code represents the evaluation strategy of this **rule-combining algorithm**.

```
5264 Decision permitOverridesRuleCombiningAlgorithm(Rule rule[])
5265 {
5266     Boolean atLeastOneError = false;
5267     Boolean potentialPermit = false;
5268     Boolean atLeastOneDeny = false;
5269     for( i=0 ; i < lengthOf(rule) ; i++ )
5270     {
5271         Decision decision = evaluate(rule[i]);
5272         if (decision == Deny)
5273         {
5274             atLeastOneDeny = true;
5275             continue;
5276         }
5277         if (decision == Permit)
```

```

5278     {
5279         return Permit;
5280     }
5281     if (decision == NotApplicable)
5282     {
5283         continue;
5284     }
5285     if (decision == Indeterminate)
5286     {
5287         atLeastOneError = true;
5288
5289         if (effect(rule[i]) == Permit)
5290         {
5291             potentialPermit = true;
5292         }
5293         continue;
5294     }
5295 }
5296 if (potentialPermit)
5297 {
5298     return Indeterminate;
5299 }
5300 if (atLeastOneDeny)
5301 {
5302     return Deny;
5303 }
5304 if (atLeastOneError)
5305 {
5306     return Indeterminate;
5307 }
5308 return NotApplicable;
5309 }

```

5310 The following specification defines the "Permit-overrides" **policy-combining algorithm** of a **policy**
5311 **set**.

5312 In the entire set of **policies** in the **policy set**, if any **policy** evaluates to "Permit", then the
5313 result of the **policy** combination SHALL be "Permit". In other words, "Permit" takes
5314 precedence, regardless of the result of evaluating any of the other **policies** in the **policy**
5315 **set**. If all **policies** are found to be "NotApplicable" to the **decision request**, then the
5316 **policy set** SHALL evaluate to "NotApplicable".

5317 If an error occurs while evaluating the **target** of a **policy**, a reference to a **policy** is
5318 considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set**
5319 SHALL evaluate to "Indeterminate", with the appropriate error status, provided no other
5320 **policies** evaluate to "Permit" or "Deny".

5321 The following pseudo-code represents the evaluation strategy of this **policy-combining algorithm**.

```

5322 Decision permitOverridesPolicyCombiningAlgorithm(Policy policy[])
5323 {
5324     Boolean atLeastOneError = false;
5325     Boolean atLeastOneDeny = false;
5326     for( i=0 ; i < lengthOf(policy) ; i++ )
5327     {
5328         Decision decision = evaluate(policy[i]);
5329         if (decision == Deny)
5330         {
5331             atLeastOneDeny = true;
5332             continue;
5333         }
5334         if (decision == Permit)

```

```

5335     {
5336         return Permit;
5337     }
5338     if (decision == NotApplicable)
5339     {
5340         continue;
5341     }
5342     if (decision == Indeterminate)
5343     {
5344         atLeastOneError = true;
5345         continue;
5346     }
5347 }
5348 if (atLeastOneDeny)
5349 {
5350     return Deny;
5351 }
5352 if (atLeastOneError)
5353 {
5354     return Indeterminate;
5355 }
5356 return NotApplicable;
5357 }

```

5358 **Obligations** of the individual *policies* shall be combined as described in Section 7.14.

5359 C.4. Ordered-permit-overrides

5360 The following specification defines the "Ordered-permit-overrides" **rule-combining algorithm** of a
5361 **policy**.

5362 The behavior of this algorithm is identical to that of the Permit-overrides **rule-combining**
5363 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL
5364 match the order as listed in the **policy**.

5365 The following specification defines the "Ordered-permit-overrides" **policy-combining algorithm** of
5366 a **policy set**.

5367 The behavior of this algorithm is identical to that of the Permit-overrides **policy-combining**
5368 **algorithm** with one exception. The order in which the collection of **policies** is evaluated
5369 SHALL match the order as listed in the **policy set**.

5370 C.5. First-applicable

5371 The following specification defines the "First-Applicable " **rule-combining algorithm** of a **policy**.

5372 Each **rule** SHALL be evaluated in the order in which it is listed in the **policy**. For a
5373 particular **rule**, if the **target** matches and the **condition** evaluates to "True", then the
5374 evaluation of the **policy** SHALL halt and the corresponding **effect** of the **rule** SHALL be the
5375 result of the evaluation of the **policy** (i.e. "Permit" or "Deny"). For a particular **rule** selected
5376 in the evaluation, if the **target** evaluates to "False" or the **condition** evaluates to "False",
5377 then the next **rule** in the order SHALL be evaluated. If no further **rule** in the order exists,
5378 then the **policy** SHALL evaluate to "NotApplicable".

5379 If an error occurs while evaluating the **target** or **condition** of a **rule**, then the evaluation
5380 SHALL halt, and the **policy** shall evaluate to "Indeterminate", with the appropriate error
5381 status.

5382 The following pseudo-code represents the evaluation strategy of this **rule-combining algorithm**.

5383

5384

```
5385 Decision firstApplicableEffectRuleCombiningAlgorithm(Rule rule[])
5386 {
5387     for( i = 0 ; i < lengthOf(rule) ; i++ )
5388     {
5389         Decision decision = evaluate(rule[i]);
5390         if (decision == Deny)
5391         {
5392             return Deny;
5393         }
5394         if (decision == Permit)
5395         {
5396             return Permit;
5397         }
5398         if (decision == NotApplicable)
5399         {
5400             continue;
5401         }
5402         if (decision == Indeterminate)
5403         {
5404             return Indeterminate;
5405         }
5406     }
5407     return NotApplicable;
5408 }
```

5409 The following specification defines the "First-applicable" **policy-combining algorithm** of a **policy**
5410 **set**.

5411 Each **policy** is evaluated in the order that it appears in the **policy set**. For a particular
5412 **policy**, if the **target** evaluates to "True" and the **policy** evaluates to a determinate value of
5413 "Permit" or "Deny", then the evaluation SHALL halt and the **policy set** SHALL evaluate to
5414 the **effect** value of that **policy**. For a particular **policy**, if the **target** evaluate to "False", or
5415 the **policy** evaluates to "NotApplicable", then the next **policy** in the order SHALL be
5416 evaluated. If no further **policy** exists in the order, then the **policy set** SHALL evaluate to
5417 "NotApplicable".

5418 If an error were to occur when evaluating the **target**, or when evaluating a specific **policy**,
5419 the reference to the **policy** is considered invalid, or the **policy** itself evaluates to
5420 "Indeterminate", then the evaluation of the **policy-combining algorithm** shall halt, and the
5421 **policy set** shall evaluate to "Indeterminate" with an appropriate error status.

5422 The following pseudo-code represents the evaluation strategy of this **policy-combination**
5423 **algorithm**.

```
5424 Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy policy[])
5425 {
5426     for( i = 0 ; i < lengthOf(policy) ; i++ )
5427     {
5428         Decision decision = evaluate(policy[i]);
5429         if(decision == Deny)
5430         {
5431             return Deny;
5432         }
5433     }
5434     return NotApplicable;
5435 }
```

```

5432     }
5433     if (decision == Permit)
5434     {
5435         return Permit;
5436     }
5437     if (decision == NotApplicable)
5438     {
5439         continue;
5440     }
5441     if (decision == Indeterminate)
5442     {
5443         return Indeterminate;
5444     }
5445 }
5446 return NotApplicable;
5447 }

```

5448 **Obligations** of the individual **policies** shall be combined as described in Section 7.14.

5449 C.6. Only-one-applicable

5450 The following specification defines the "Only-one-applicable" **policy-combining algorithm** of a
5451 **policy set**.

5452 In the entire set of **policies** in the **policy set**, if no **policy** is considered applicable by virtue of
5453 its **target**, then the result of the **policy** combination algorithm SHALL be "NotApplicable".
5454 If more than one **policy** is considered applicable by virtue of its **target**, then the result of
5455 the **policy** combination algorithm SHALL be "Indeterminate".

5456 If only one **policy** is considered applicable by evaluation of its **target**, then the result of the
5457 **policy-combining algorithm** SHALL be the result of evaluating the **policy**.

5458 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is
5459 considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set**
5460 SHALL evaluate to "Indeterminate", with the appropriate error status.

5461 The following pseudo-code represents the evaluation strategy of this policy combining algorithm.

```

5462 Decision onlyOneApplicablePolicyPolicyCombiningAlgorithm(Policy policy[])
5463 {
5464     Boolean          atLeastOne      = false;
5465     Policy           selectedPolicy  = null;
5466     ApplicableResult appResult;
5467
5468     for ( i = 0; i < lengthOf(policy) ; i++ )
5469     {
5470         appResult = isApplicable(policy[i]);
5471
5472         if ( appResult == Indeterminate )
5473         {
5474             return Indeterminate;
5475         }
5476         if( appResult == Applicable )
5477         {
5478             if ( atLeastOne )
5479             {
5480                 return Indeterminate;
5481             }
5482             else

```

```
5483     {
5484         atLeastOne     = true;
5485         selectedPolicy = policy[i];
5486     }
5487 }
5488 if ( appResult == NotApplicable )
5489 {
5490     continue;
5491 }
5492 }
5493 if ( atLeastOne )
5494 {
5495     return evaluate(selectedPolicy);
5496 }
5497 else
5498 {
5499     return NotApplicable;
5500 }
5501 }
5502
```

5503

Appendix D. Acknowledgments

5504

The following individuals contributed to the development of the specification:

5505

Anne Anderson

5506

Anthony Nadalin

5507

Bill Parducci

5508

Daniel Engovatov

5509

Don Flinn

5510

Ed Coyne

5511

Ernesto Damiani

5512

Frank Siebenlist

5513

Gerald Brose

5514

Hal Lockhart

5515

Haruyuki Kawabe

5516

James MacLean

5517

John Merrells

5518

Ken Yagen

5519

Konstantin Beznosov

5520

Michiharu Kudo

5521

Michael McIntosh

5522

Pierangela Samarati

5523

Pirasenna Velandai Thiyagarajan

5524

Polar Humenn

5525

Rebekah Metz

5526

Ron Jacobson

5527

Satoshi Hada

5528

Sekhar Vajjhala

5529

Seth Proctor

5530

Simon Godik

5531

Steve Anderson

5532

Steve Crocker

5533

Suresh Damodaran

5534

Tim Moses

5535

Von Welch

5536

Appendix E. Notices

5537 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
5538 that might be claimed to pertain to the implementation or use of the technology described in this
5539 document or the extent to which any license under such rights might or might not be available;
5540 neither does it represent that it has made any effort to identify any such rights. Information on
5541 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
5542 website. Copies of claims of rights made available for publication and any assurances of licenses to
5543 be made available, or the result of an attempt made to obtain a general license or permission for
5544 the use of such proprietary rights by implementors or users of this specification, can be obtained
5545 from the OASIS Executive Director.

5546 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
5547 applications, or other proprietary rights which may cover technology that may be required to
5548 implement this specification. Please address the information to the OASIS Executive Director.

5549 **Copyright © OASIS Open 2004-2005. All Rights Reserved.**

5550 This document and translations of it may be copied and furnished to others, and derivative works
5551 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
5552 published and distributed, in whole or in part, without restriction of any kind, provided that the above
5553 copyright notice and this paragraph are included on all such copies and derivative works. However,
5554 this document itself does not be modified in any way, such as by removing the copyright notice or
5555 references to OASIS, except as needed for the purpose of developing OASIS specifications, in
5556 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights
5557 document must be followed, or as required to translate it into languages other than English.

5558 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
5559 successors or assigns. This document and the information contained herein is provided on an "AS
5560 IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT
5561 NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL
5562 NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR
5563 FITNESS FOR A PARTICULAR PURPOSE.