



1

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

4 **Committee Specification 1.0 (Revision 1), 12**
5 **December 2002**

6 Document identifier: cs-xacml-specification-1.0-1.doc

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

8 Send comments to: xacml-comment@lists.oasis-open.org

9 Editors:

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

12 Contributors:

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

35 Abstract:

36 This specification defines an XML schema for an extensible access-control policy
37 language.

38 Status:

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

41 If you are on the xacml@lists.oasis-open.org list for committee members, send comments
42 there. If you are not on that list, subscribe to the xacml-comment@lists.oasis-open.org list
43 and send comments there. To subscribe, send an email message to [xacml-comment-](mailto:xacml-comment-request@lists.oasis-open.org)
44 [request@lists.oasis-open.org](mailto:xacml-comment-request@lists.oasis-open.org) with the word "subscribe" as the body of the message.

45

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

47	Table of contents	
48	1. Introduction (non-normative)	9
49	1.1. Glossary	9
50	1.1.1 Preferred terms	9
51	1.1.2 Related terms	10
52	1.2. Notation	10
53	1.3. Schema organization and namespaces	11
54	2. Background (non-normative)	11
55	2.1. Requirements	12
56	2.2. Rule and policy combining	13
57	2.3. Combining algorithms	13
58	2.4. Multiple subjects	14
59	2.5. Policies based on subject and resource attributes	14
60	2.6. Multi-valued attributes	14
61	2.7. Policies based on resource contents	15
62	2.8. Operators	15
63	2.9. Policy distribution	16
64	2.10. Policy indexing	16
65	2.11. Abstraction layer	16
66	2.12. Actions performed in conjunction with enforcement	17
67	3. Models (non-normative)	17
68	3.1. Data-flow model	17
69	3.2. XACML context	19
70	3.3. Policy language model	19
71	3.3.1 Rule	20
72	3.3.2 Policy	22
73	3.3.3 Policy set	23
74	4. Examples (non-normative)	23
75	4.1. Example one	24
76	4.1.1 Example policy	24
77	4.1.2 Example request context	26
78	4.1.3 Example response context	27
79	4.2. Example two	27
80	4.2.1 Example medical record instance	28
81	4.2.2 Example request context	29
82	4.2.3 Example plain-language rules	31

83	4.2.4	Example XACML rule instances	31
84	5.	Policy syntax (normative, with the exception of the schema fragments)	44
85	5.1.	Element <PolicySet>	44
86	5.2.	Element <Description>	46
87	5.3.	Element <PolicySetDefaults>	46
88	5.4.	Element <XPathVersion>	47
89	5.5.	Element <Target>	47
90	5.6.	Element <Subjects>	47
91	5.7.	Element <Subject>	48
92	5.8.	Element <AnySubject>	48
93	5.9.	Element <SubjectMatch>	48
94	5.10.	Element <Resources>	49
95	5.11.	Element <Resource>	49
96	5.12.	Element <AnyResource>	50
97	5.13.	Element <ResourceMatch>	50
98	5.14.	Element <Actions>	50
99	5.15.	Element <Action>	51
100	5.16.	Element <AnyAction>	51
101	5.17.	Element <ActionMatch>	51
102	5.18.	Element <PolicySetIdReference>	52
103	5.19.	Element <PolicyIdReference>	52
104	5.20.	Element <Policy>	52
105	5.21.	Element <PolicyDefaults>	53
106	5.22.	Element <Rule>	54
107	5.23.	Simple type EffectType	54
108	5.24.	Element <Condition>	55
109	5.25.	Element <Apply>	55
110	5.26.	Element <Function>	56
111	5.27.	Complex type AttributeDesignatorType	56
112	5.28.	Element SubjectAttributeDesignator	57
113	5.29.	Element <ResourceAttributeDesignator>	58
114	5.30.	Element <ActionAttributeDesignator>	59
115	5.31.	Element <EnvironmentAttributeDesignator>	59
116	5.32.	Element <AttributeSelector>	60
117	5.33.	Element <AttributeValue>	61
118	5.34.	Element <Obligations>	61
119	5.35.	Element <Obligation>	61

120	5.36. Element <AttributeAssignment>	62
121	6. Context syntax (normative with the exception of the schema fragments)	63
122	6.1. Element <Request>	63
123	6.2. Element <Subject>	64
124	6.3. Element <Resource>	64
125	6.4. Element <ResourceContent>	65
126	6.5. Element <Action>	65
127	6.6. Element <Environment>	65
128	6.7. Element <Attribute>	66
129	6.8. Element <AttributeValue>	66
130	6.9. Element <Response>	67
131	6.10. Element <Result>	67
132	6.11. Element <Decision>	68
133	6.12. Element <Status>	68
134	6.13. Element <StatusCode>	69
135	6.14. Element <StatusMessage>	69
136	6.15. Element <StatusDetail>	69
137	7. Functional requirements (normative)	70
138	7.1. Policy enforcement point	70
139	7.2. Base policy	71
140	7.3. Target evaluation	71
141	7.4. Condition evaluation	71
142	7.5. Rule evaluation	71
143	7.6. Policy evaluation	72
144	7.7. Policy Set evaluation	73
145	7.8. Hierarchical resources	73
146	7.9. Attributes	74
147	7.9.1. Attribute Matching	74
148	7.9.2. Attribute Retrieval	75
149	7.9.3. Environment Attributes	75
150	7.10. Authorization decision	75
151	7.11. Obligations	76
152	7.12. Unsupported functionality	76
153	7.13. Syntax and type errors	76
154	8. XACML extensibility points (non-normative)	76
155	8.1. Extensible XML attribute types	76
156	8.2. Structured attributes	77

157	9. Security and privacy considerations (non-normative)	77
158	9.1. Threat model	77
159	9.1.1. Unauthorized disclosure	78
160	9.1.2. Message replay	78
161	9.1.3. Message insertion	78
162	9.1.4. Message deletion	78
163	9.1.5. Message modification	79
164	9.1.6. NotApplicable results	79
165	9.1.7. Negative rules	79
166	9.2. Safeguards	80
167	9.2.1. Authentication	80
168	9.2.2. Policy administration	80
169	9.2.3. Confidentiality	81
170	9.2.4. Policy integrity	81
171	9.2.5. Policy identifiers	82
172	9.2.6. Trust model	82
173	9.2.7. Privacy	82
174	10. Conformance (normative)	83
175	10.1. Introduction	83
176	10.2. Conformance tables	83
177	10.2.1. Schema elements	83
178	10.2.2. Identifier Prefixes	84
179	10.2.3. Algorithms	84
180	10.2.4. Status Codes	85
181	10.2.5. Attributes	85
182	10.2.6. Identifiers	85
183	10.2.7. Data-types	85
184	10.2.8. Functions	86
185	11. References	90
186	Appendix A. Standard data-types, functions and their semantics (normative)	92
187	A.1. Introduction	92
188	A.2. Primitive types	92
189	A.3. Structured types	93
190	A.4. Representations	93
191	A.5. Bags	94
192	A.6. Expressions	94
193	A.7. Element <AttributeValue>	95

194	A.8.	Elements <AttributeDesignator> and <AttributeSelector>	95
195	A.9.	Element <Apply>	95
196	A.10.	Element <Condition>	95
197	A.11.	Element <Function>	96
198	A.12.	Matching elements	96
199	A.13.	Arithmetic evaluation	97
200	A.14.	XACML standard functions	98
201	A14.1	Equality predicates	98
202	A14.2	Arithmetic functions	100
203	A14.3	String conversion functions	101
204	A14.4	Numeric data-type conversion functions	101
205	A14.5	Logical functions	101
206	A14.6	Arithmetic comparison functions	102
207	A14.7	Date and time arithmetic functions	103
208	A14.8	Non-numeric comparison functions	104
209	A14.9	Bag functions	106
210	A14.10	Set functions	107
211	A14.11	Higher-order bag functions	107
212	A14.12	Special match functions	114
213	A14.13	XPath-based functions	115
214	A14.14	Extension functions and primitive types	115
215		Appendix B. XACML identifiers (normative)	116
216	B.1.	XACML namespaces	116
217	B.2.	Access subject categories	116
218	B.3.	XACML functions	116
219	B.4.	Data-types	116
220	B.5.	Subject attributes	117
221	B.6.	Resource attributes	118
222	B.7.	Action attributes	118
223	B.8.	Environment attributes	119
224	B.9.	Status codes	119
225	B.10.	Combining algorithms	119
226		Appendix C. Combining algorithms (normative)	121
227	C.1.	Deny-overrides	121
228	C.2.	Permit-overrides	123
229	C.3.	First-applicable	125
230	C.4.	Only-one-applicable	126

231	Appendix D. Acknowledgments	128
232	Appendix E. Tim Moses, Entrust, tim.moses@entrust.comRevision history	129
233	Appendix F. Notices	130
234		

235

236 1. Introduction (non-normative)

237 1.1. Glossary

238 1.1.1 Preferred terms

239 **Access** - Performing an *action*

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

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

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

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

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

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

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

252 **Conjunctive sequence** - a sequence of boolean elements combined using the logical 'AND'
253 operation

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

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

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

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

260 **Disjunctive sequence** - a sequence of boolean elements combined using the logical 'OR'
261 operation

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

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

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

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

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

270 **Policy-combining algorithm** - The procedure for combining the **decision** and **obligations** from
271 multiple **policies**

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

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

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

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

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

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

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

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

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

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

286 1.1.2 Related terms

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

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

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

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

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

294 1.2. Notation

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

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

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

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

306 Listings of XACML schemas appear like this.

307
308 Example code listings appear like this.

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

- 312 • The prefix `xacml`: stands for the XACML policy namespace.
- 313 • The prefix `xacml-context`: stands for the XACML context namespace.
- 314 • The prefix `ds`: stands for the W3C XML Signature namespace [DS].
- 315 • The prefix `xs`: stands for the W3C XML Schema namespace [XS].
- 316 • The prefix `xf`: stands for the XPath query and function specification namespace [XF].

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

320 1.3. Schema organization and namespaces

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

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

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

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

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

327 `http://www.w3.org/2000/09/xmlsig#`

328 2. Background (non-normative)

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

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

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

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

355 2.1. Requirements

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

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

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

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

380 2.2. Rule and policy combining

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

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

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

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

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

400 2.3. Combining algorithms

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

- 408 • Deny-overrides,
- 409 • Permit-overrides,
- 410 • First applicable and
- 411 • Only-one-applicable.

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

416 combining algorithm, the combined result is the same as the result of evaluating the first <Rule>,
417 <Policy> or <PolicySet> element in the list of **rules** whose **target** is applicable to the **decision**
418 **request**. The "Only-one-applicable" **policy-combining algorithm** only applies to **policies**. The
419 result of this combining algorithm ensures that one and only one **policy** or **policy set** is applicable
420 by virtue of their **targets**. If no **policy** or **policy set** applies, then the result is "NotApplicable", but if
421 more than one **policy** or **policy set** is applicable, then the result is "Indeterminate". When exactly
422 one **policy** or **policy set** is applicable, the result of the combining algorithm is the result of
423 evaluating the single **applicable policy** or **policy set**.

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

425 **2.4. Multiple subjects**

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

432 **2.5. Policies based on subject and resource attributes**

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

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

449 **2.6. Multi-valued attributes**

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

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

458 2.7. Policies based on resource contents

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

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

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

469 2.8. Operators

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

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

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

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

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

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

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

499 2.9. Policy distribution

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

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

511 2.10. Policy indexing

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

517 Two approaches are supported:

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

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

528 2.11. Abstraction layer

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

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

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

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

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

551 **2.12. Actions performed in conjunction with enforcement**

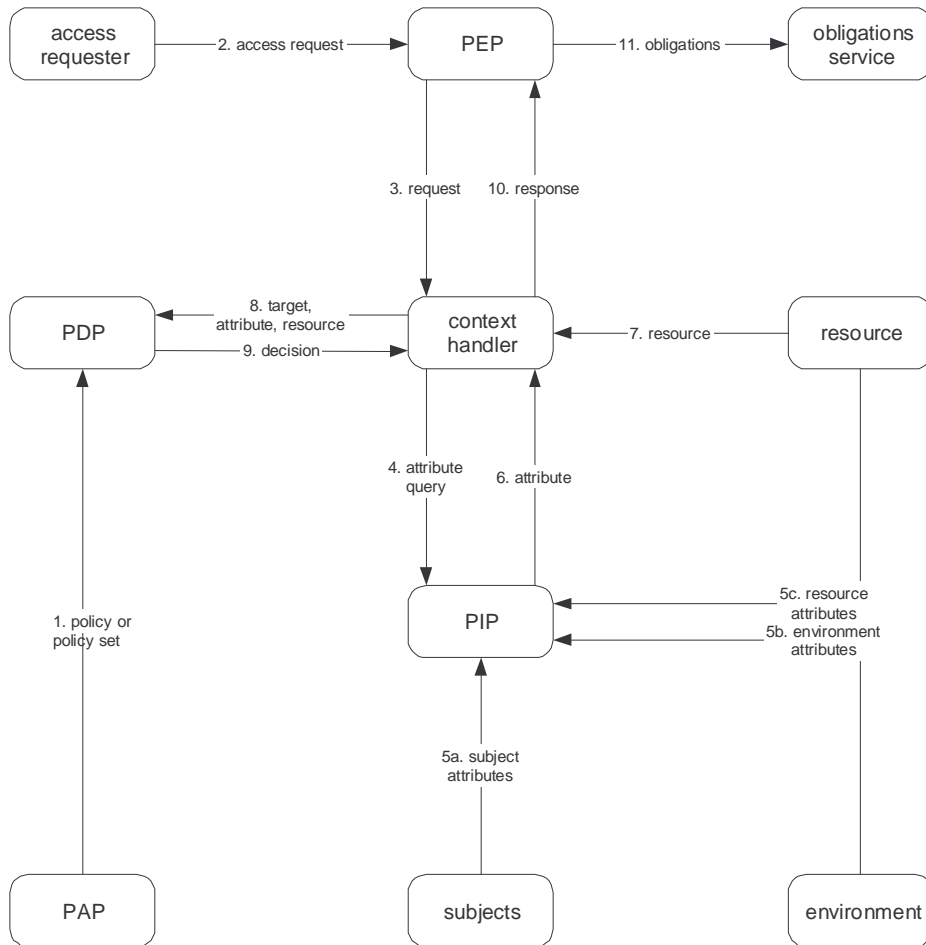
552 In many applications, policies specify actions that MUST be performed, either instead of, or in
553 addition to, actions that MAY be performed. This idea was described by Sloman [Sloman94].
554 XACML provides facilities to specify actions that MUST be performed in conjunction with policy
555 evaluation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

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

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

563 **3.1. Data-flow model**

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



565

566

Figure 1 - Data-flow diagram

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

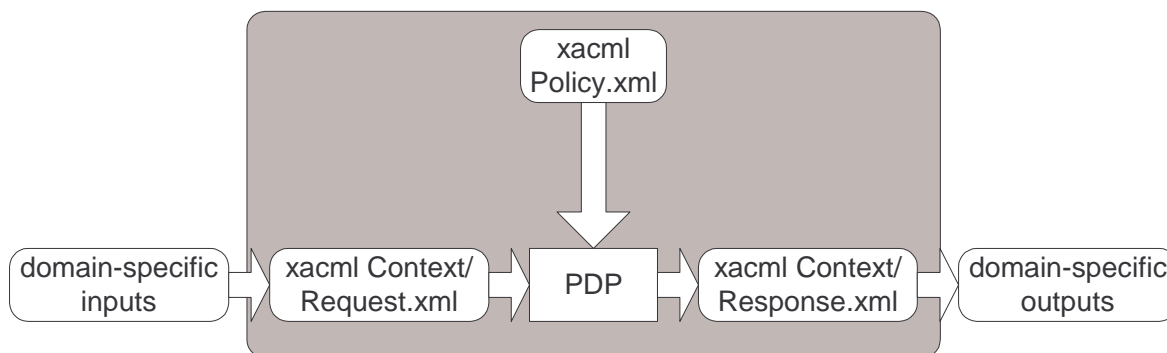
572 The model operates by the following steps.

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

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

593 3.2. XACML context

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



605
 606 **Figure 2 - XACML context**

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

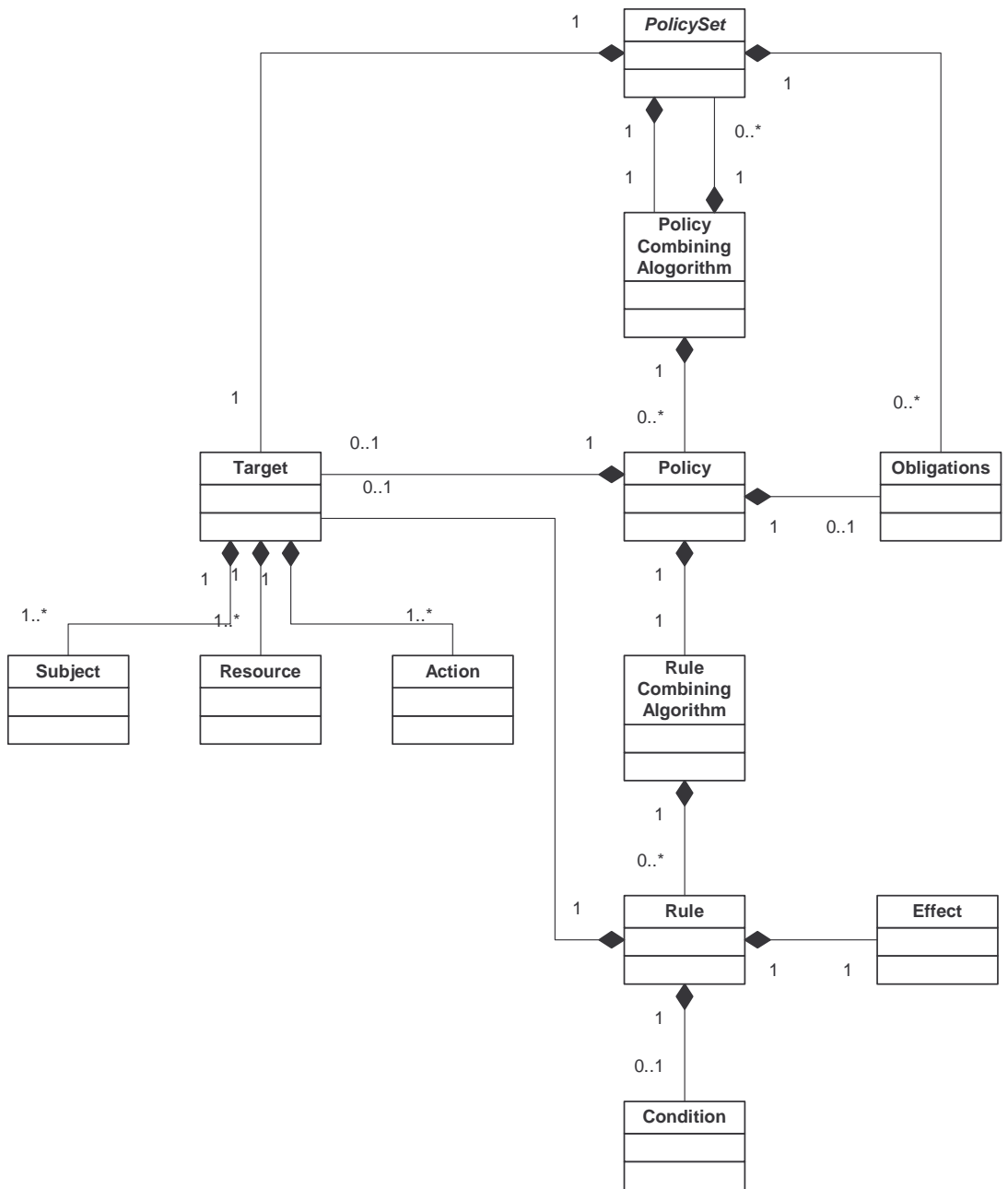
608 See Section 7.9 for a more detailed discussion of the request **context**.

609 3.3. Policy language model

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

- 611 • **Rule**;
- 612 • **Policy**; and

- 613 • **Policy set.**
- 614 These are described in the following sub-sections.



615

616

Figure 3 - Policy language model

617

3.3.1 Rule

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

- 622 • a **target**,
- 623 • an **effect**, and
- 624 • a **condition**.

625 These are discussed in the following sub-sections.

626 3.3.1.1. Rule target

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

- 628 • **resources**;
- 629 • **subjects**; and
- 630 • **actions**

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

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

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

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

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

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

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

- 661 1. the contents of the identified node only,
- 662 2. the contents of the identified node and the contents of its immediate child nodes or
- 663 3. the contents of the identified node and all its descendant nodes.

664 All three options are supported in XACML.

665 **3.3.1.2. Effect**

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

668 **3.3.1.3. Condition**

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

671 **3.3.2 Policy**

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

- 674 • a **target**,
- 675 • a **rule-combining algorithm**-identifier;
- 676 • a set of **rules**; and
- 677 • **obligations**.

678 **Rules** are described above. The remaining components are described in the following sub-
679 sections.

680 **3.3.2.1. Policy target**

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

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

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

702 3.3.2.2. Rule-combining algorithm

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

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

708 3.3.2.3. Obligations

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

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

713 3.3.3 Policy set

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

- 715 • a **target**,
- 716 • a **policy-combining algorithm**-identifier
- 717 • a set of **policies**; and
- 718 • **obligations**.

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

721 3.3.3.1. Policy-combining algorithm

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

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

727 3.3.3.2. Obligations

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

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

732 4. Examples (non-normative)

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

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

737 4.1. Example one

738 4.1.1 Example policy

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

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

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

745 The header for this policy is

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

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

748 Line [p02] introduces the XACML Policy itself.

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

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

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

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

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

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

```
[p11] <Target>
[p12] <Subjects>
[p13] <AnySubject/>
[p14] </Subjects>
[p15] <Resources>
[p16] <AnyResource/>
[p17] </Resources>
[p18] <Actions>
[p19] <AnyAction/>
[p20] </Actions>
[p21] </Target>
```


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

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

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

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

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

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

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

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

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

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

780 The **rule target** is similar to the **target** of the **policy** itself, but with one important difference. Lines
781 [p32-p41] do not say <AnySubject/>, but instead spell out a specific value that the **subject** in the
782 **decision request** must match. The <SubjectMatch> element specifies a matching function in
783 the MatchId attribute, a pointer to a specific **subject attribute** in the request **context** by means of

784 the <SubjectAttributeDesignator> element, and a literal value of “medico.com”. The
785 matching function will be used to compare the value of the **subject attribute** with the literal value.
786 Only if the match returns “True” will this **rule** apply to a particular **decision request**. If the match
787 returns “False”, then this **rule** will return a value of “NotApplicable”.

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

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

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

793 4.1.2 Example request context

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

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

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

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

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

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

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

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

807 The <Resource> element contains one or more **attributes** of the **resource** to which
808 the **subject** (or **subjects**) has requested **access**. There can be only one <Resource>
809 per **decision request**. Lines [c13-c16] contain the one **attribute** of the **resource**
810 to which Bart Simpson has requested **access**: the **resource** unix file-system path-
811 name, which is “/medico/record/patient/BartSimpson”.

```
[c18] <Action>
[c19] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
```

```
[c20]      DataType="http://www.w3.org/2001/XMLSchema#string">
[c21]      <AttributeValue>read</AttributeValue>
[c22]      </Attribute>
[c23]      </Action>
```

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

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

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

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

822 The *PDP* now compares the *subject*, *resource* and *action* in the request *context* with the *target*
823 of the one *rule* in this *policy*. The requested *resource* matches the <AnyResource/> element
824 and the requested *action* matches the <AnyAction/> element, but the requesting subject-id
825 *attribute* does not match “*@medico.com”.

826 4.1.3 Example response context

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

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

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

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

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

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

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

836 4.2. Example two

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

840

4.2.1 Example medical record instance

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

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

```

901     <name>Blue Cross</name>
902     <street>1234 Main St</street>
903     <city>Springfield</city>
904     <state>MA</state>
905     <zip>12345</zip>
906     <phone>555.123.5678</phone>
907     <fax>555.123.5679</fax>
908     <email/>
909 </insurer>
910 <medical>
911   <treatment>
912     <drug>
913       <name>methylphenidate hydrochloride</name>
914       <dailyDosage>30mgs</dailyDosage>
915       <startDate>1999-01-12</startDate>
916     </drug>
917     <comment>patient exhibits side-effects of skin coloration and carpal
918 degeneration</comment>
919   </treatment>
920   <result>
921     <test>blood pressure</test>
922     <value>120/80</value>
923     <date>2001-06-09</date>
924     <performedBy>Nurse Betty</performedBy>
925   </result>
926 </medical>
927 </record>

```

928 4.2.2 Example request context

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

```

932 [01] <?xml version="1.0" encoding="UTF-8"?>
933 [02] <Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
934 [03] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
935 [04] <Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-
936 category:access-subject">
937   <Attribute AttributeId=
938     [06] "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
939     [07] DataType=
940     [08] "urn:oasis:names:tc:xacml:1.0:data-type:x500name"
941     [09] Issuer="www.medico.com"
942     [10] IssueInstant="2001-12-17T09:30:47-05:00">
943     [11] <AttributeValue>CN=Julius Hibbert</AttributeValue>
944   </Attribute>
945   <Attribute AttributeId=
946     [14] "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
947     [15] DataType="http://www.w3.org/2001/XMLSchema#string"
948     [16] Issuer="www.medico.com"
949     [17] IssueInstant="2001-12-17T09:30:47-05:00">
950     [18] <AttributeValue>physician</AttributeValue>
951   </Attribute>
952   <Attribute AttributeId=
953     [21] "urn:oasis:names:tc:xacml:1.0:example:attribute:physician-id"
954     [22] DataType="http://www.w3.org/2001/XMLSchema#string"
955     [23] Issuer="www.medico.com"
956     [24] IssueInstant="2001-12-17T09:30:47-05:00">
957     [25] <AttributeValue>jh1234</AttributeValue>
958   </Attribute>
959 [27] </Subject>
960 [28] </Resource>

```

```

961 [29] <ResourceContent>
962 [30]   <md:record
963 [31]     xmlns:md="//http:www.medico.com/schemas/record.xsd">
964 [32]     <md:patient>
965 [33]       <md:patientDoB>1992-03-21</md:patientDoB>
966 [34]     </md:patient>
967 [35]     <!-- other fields -->
968 [36]   </md:record>
969 [37] </ResourceContent>
970 [38] <Attribute AttributeId=
971 [39]   "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
972 [40]   DataType="http://www.w3.org/2001/XMLSchema#string">
973 [41]   <AttributeValue>
974 [42]     //medico.com/records/bart-simpson.xml#
975 [43]     xmlns(md="//http:www.medico.com/schemas/record.xsd)
976 [44]     xpointer(/md:record/md:patient/md:patientDoB)
977 [45]   </AttributeValue>
978 [46] </Attribute>
979 [47] <Attribute AttributeId=
980 [48]   "urn:oasis:names:tc:xacml:1.0:resource:xpath"
981 [49]   DataType="http://www.w3.org/2001/XMLSchema#string">
982 [50]   <AttributeValue>
983 [51]     xmlns(md=http:www.medico.com/schemas/record.xsd)
984 [52]     xpointer(/md:record/md:patient/md:patientDoB)
985 [53]   </AttributeValue>
986 [54] </Attribute>
987 [55] <Attribute AttributeId=
988 [56]   "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
989 [57]   DataType="http://www.w3.org/2001/XMLSchema#string">
990 [58]   <AttributeValue>
991 [59]     http://www.medico.com/schemas/record.xsd
992 [60]   </AttributeValue>
993 [61] </Attribute>
994 [62] </Resource>
995 [63] <Action>
996 [64]   <Attribute AttributeId=
997 [65]     "urn:oasis:names:tc:xacml:1.0:action:action-id"
998 [66]     DataType="http://www.w3.org/2001/XMLSchema#string">
999 [67]     <AttributeValue>read</AttributeValue>
1000 [68]   </Attribute>
1001 [69] </Action>
1002 [70] </Request>

```

1003 [02]-[03] Standard namespace declarations.

1004 [04]-[27] **Subject** attributes are placed in the Subject section of the Request. Each **attribute**
1005 consists of the **attribute** meta-data and the **attribute** value.

1006 [04] Each Subject element has SubjectCategory xml attribute. The value of this attribute
1007 describes the role that the **subject** plays in making the **decision request**. The value of "access-
1008 subject" denotes the identity for which the request was issued.

1009 [05]-[12] **Subject** subject-id **attribute**.

1010 [13]-[19] **Subject** role **attribute**.

1011 [20]-[26] **Subject** physician-id **attribute**.

1012 [28]-[62] **Resource** attributes are placed in the Resource section of the Request. Each **attribute**
1013 consists of **attribute** meta-data and an **attribute** value.

1014 [29]-[36] **Resource** content. The XML document that is being requested is placed here.

- 1015 [38]-[46] **Resource** identifier.
- 1016 [47]-[61] The **Resource** is identified with an Xpointer expression that names the URI of the file that
 1017 is accessed, the target namespace of the document, and the XPath location path to the specific
 1018 element.
- 1019 [47]-[54] The XPath location path in the “resource-id” attribute is extracted and placed in the
 1020 xpath attribute.
- 1021 [55]-[61] **Resource** target-namespace **attribute**.
- 1022 [63]-[69] **Action attributes** are placed in the Action section of the Request.
- 1023 [64]-[68] **Action** identifier.

4.2.3 Example plain-language rules

1024 The following plain-language rules are to be enforced:

1026 Rule 1: A person, identified by his or her patient number, may read any record for which he
 1027 or she is the designated patient.

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

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

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

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

4.2.4 Example XACML rule instances

4.2.4.1. Rule 1

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

```

1039 [01] <?xml version="1.0" encoding="UTF-8"?>
1040 [02] <Rule
1041 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1042 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1043 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1044 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1045 [07]   RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"
1046 [08]   Effect="Permit">
1047 [09]   <Description>
1048 [10]     A person may read any medical record in the
1049 [11]     http://www.medico.com/schemas/record.xsd namespace
1050 [12]     for which he or she is a designated patient
1051 [13]   </Description>
1052 [14]   <Target>
1053 [15]     <Subjects>
1054 [16]       <AnySubject/>
1055 [17]     </Subjects>
1056 [18]     <Resources>
1057 [20]       <Resource>

```

```

1058 [21]         <!-- match document target namespace -->
1059 [22]         <ResourceMatch
1060 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1061 [23]         <AttributeValue
1062 DataType="http://www.w3.org/2001/XMLSchema#string">
1063 [24]             http://www.medico.com/schemas/record.xsd
1064 [25]         </AttributeValue>
1065 [26]         <ResourceAttributeDesignator AttributeId=
1066 [27]             "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1067 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1068 [28]         </ResourceMatch>
1069 [29]         <!-- match requested xml element -->
1070 [30]         <ResourceMatch
1071 MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1072 [31]         <AttributeValue
1073 DataType="http://www.w3.org/2001/XMLSchema#string">/md:record</AttributeValue>
1074 [32]         <ResourceAttributeDesignator AttributeId=
1075 [33]             "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1076 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1077 [34]         </ResourceMatch>
1078 [35]     </Resource>
1079 [36] </Resources>
1080 [37] <Actions>
1081 [38]     <Action>
1082 [39]         <ActionMatch
1083 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1084 [40]         <AttributeValue
1085 DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1086 [41]         <ActionAttributeDesignator AttributeId=
1087 [42]             "urn:oasis:names:tc:xacml:1.0:action:action-id"
1088 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1089 [43]         </ActionMatch>
1090 [44]     </Action>
1091 [45] </Actions>
1092 [46] </Target>
1093 [47] <!-- compare policy number in the document with
1094 [48]     policy-number attribute -->
1095 [49] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1096 equal">
1097 [50]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1098 and-only">
1099 [51]         <!-- policy-number attribute -->
1100 [52]         <SubjectAttributeDesignator AttributeId=
1101 [53]             "urn:oasis:names:tc:xacml:1.0:examples:attribute:policy-number"
1102 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1103 [54]         </Apply>
1104 [55]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1105 and-only">
1106 [56]         <!-- policy number in the document -->
1107 [57]         <AttributeSelector RequestContextPath=
1108 [58]             "//md:record/md:patient/md:patient-number/text()"
1109 DataType="http://www.w3.org/2001/XMLSchema#string">
1110 [59]         </AttributeSelector>
1111 [60]     </Apply>
1112 [61] </Condition>
1113 [62] </Rule>

```

1114 [02]-[06]. XML namespace declarations.

1115 [07] **Rule** identifier.

1116 [08]. When a **rule** evaluates to 'True' it emits the value of the **Effect** attribute. This value is
1117 combined with the **Effect** values of other rules according to the **rule-combining algorithm**.

1118 [09]-[13] Free form description of the *rule*.

1119 [14]-[46]. A *rule target* defines a set of *decision requests* that are applicable to the *rule*. A
1120 *decision request*, such that the value of the
1121 “urn:oasis:names:tc:xacml:1.0:resource:target-namespace” *resource attribute* is
1122 equal to “http://www.medico.com/schema/records.xsd” and the value of the
1123 “urn:oasis:names:tc:xacml:1.0:resource:xpath” *resource attribute* matches the XPath
1124 expression “/md:record” and the value of the
1125 “urn:oasis:names:tc:xacml:1.0:action:action-id” *action attribute* is equal to “read”,
1126 matches the *target* of this *rule*.

1127 [15]-[17]. The *Subjects* element may contain either a *disjunctive sequence* of *Subject*
1128 elements or *AnySubject* element.

1129 [16] The *AnySubject* element is a special element that matches any *subject* in the request
1130 *context*.

1131 [18]-[36]. The *Resources* element may contain either a *disjunctive sequence* of *Resource*
1132 elements or *AnyResource* element.

1133 [20]-[35] The *Resource* element encloses the *conjunctive sequence* of *ResourceMatch*
1134 elements.

1135 [22]-[28] The *ResourceMatch* element compares its first and second child elements according to
1136 the matching function. A match is positive if the value of the first argument matches any of the
1137 values selected by the second argument. This match compares the target namespace of the
1138 requested document with the value of “http://www.medico.com/schema.records.xsd”.

1139 [22] The *MatchId* attribute names the matching function.

1140 [23]-[25] Literal attribute value to match.

1141 [26]-[27] The *ResourceAttributeDesignator* element selects the *resource attribute* values
1142 from the request *context*. The *attribute* name is specified by the *AttributeId*. The selection
1143 result is a *bag* of values.

1144 [30]-[34] The *ResourceMatch*. This match compares the results of two XPath expressions. The
1145 first XPath expression is /md:record and the second XPath expression is the location path to the
1146 requested xml element. The “xpath-node-match” function evaluates to “True” if the requested XML
1147 element is below the /md:record element.

1148 [30] *MatchId* attribute names the matching function.

1149 [31] The literal XPath expression to match. The md prefix is resolved using a standard namespace
1150 declaration.

1151 [32]-[33] The *ResourceAttributeDesignator* selects the *bag* of values for the
1152 “urn:oasis:names:tc:xacml:1.0:xpath” *resource attribute*. Here, there is just one
1153 element in the *bag*, which is the location path for the requested XML element.

1154 [37]-[45] The *Actions* element may contain either a *disjunctive sequence* of *Action* elements
1155 or an *AnyAction* element.

1156 [38]-[44] The *Action* element contains a *conjunctive sequence* of *ActionMatch* elements.

1157 [39]-[43] The *ActionMatch* element compares its first and second child elements according to the
1158 matching function. Match is positive if the value of the first argument matches any of the values
1159 selected by the second argument. In this case, the value of the *action-id* action attribute in the
1160 request *context* is compared with the value “read”.

1161 [39] The `MatchId` attribute names the matching function.

1162 [40] The **Attribute** value to match. This is an **action** name.

1163 [41]-[42] The `ActionAttributeDesignator` selects **action attribute** values from the request
1164 **context**. The **attribute** name is specified by the `AttributeId`. The selection result is a **bag** of
1165 values. “urn:oasis:names:tc:xacml:1.0:action:action-id” is the predefined name for
1166 the action identifier.

1167 [49]-[61] The `<Condition>` element. A **condition** must evaluate to “True” for the **rule** to be
1168 applicable. This condition evaluates the truth of the statement: the `patient-number` **subject**
1169 **attribute** is equal to the `patient-number` in the XML document.

1170 [49] The `FunctionId` attribute of the `<Condition>` element names the function to be used for
1171 comparison. In this case, comparison is done with
1172 `urn:oasis:names:tc:xacml:1.0:function:string-equal`; this function takes two
1173 arguments of the “`http://www.w3.org/2001/XMLSchema#string`” data-type.

1174 [50] The first argument to the `urn:oasis:names:tc:xacml:1.0:function:string-equal`
1175 in the `Condition`. Functions can take other functions as arguments. The `Apply` element
1176 encodes the function call with the `FunctionId` attribute naming the function. Since
1177 `urn:oasis:names:tc:xacml:1.0:function:string-equal` takes arguments of the
1178 “`http://www.w3.org/2001/XMLSchema#string`” data-type and
1179 `SubjectAttributeDesignator` selects a **bag** of
1180 “`http://www.w3.org/2001/XMLSchema#string`” values,
1181 “`urn:oasis:names:tc:xacml:1.0:function:string-one-and-only`” is used. This
1182 function guarantees that its argument evaluates to a **bag** containing one and only one
1183 “`http://www.w3.org/2001/XMLSchema#string`” element.

1184 [52]-[53] The `SubjectAttributeDesignator` selects a **bag** of values for the `policy-number`
1185 **subject attribute** in the request **context**.

1186 [55] The second argument to the “`urn:oasis:names:tc:xacml:1.0:function:string-`
1187 `equal`” in the `Condition`. Functions can take other functions as arguments. The `Apply` element
1188 encodes function call with the `FunctionId` attribute naming the function. Since
1189 “`urn:oasis:names:tc:xacml:1.0:function:string-equal`” takes arguments of the
1190 “`http://www.w3.org/2001/XMLSchema#string`” data-type and the `AttributeSelector`
1191 selects a **bag** of “`http://www.w3.org/2001/XMLSchema#string`” values,
1192 “`urn:oasis:names:tc:xacml:1.0:function:string-one-and-only`” is used. This
1193 function guarantees that its argument evaluates to a **bag** containing one and only one
1194 “`http://www.w3.org/2001/XMLSchema#string`” element.

1195 [57] The `AttributeSelector` element selects a **bag** of values from the request **context**. The
1196 `AttributeSelector` is a free-form XPath pointing device into the request **context**. The
1197 `RequestContextPath` attribute specifies an XPath expression over the content of the requested
1198 XML document, selecting the policy number. Note that the namespace prefixes in the XPath
1199 expression are resolved with the standard XML namespace declarations.

1200 4.2.4.2. Rule 2

1201 Rule 2 illustrates the use of a mathematical function, i.e. the `<Apply>` element with `functionId`
1202 “urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration” to calculate date. It also
1203 illustrates the use of **predicate** expressions, with the `functionId`
1204 “urn:oasis:names:tc:xacml:1.0:function:and”.

1205 [01] `<?xml version="1.0" encoding="UTF-8" ?>`

```

1206 [02] <Rule
1207 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1208 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1209 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1210 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1211 [07]   RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
1212 [08]   Effect="Permit">
1213 [09]   <Description>
1214 [10]     A person may read any medical record in the
1215 [11]     http://www.medico.com/records.xsd namespace
1216 [12]     for which he or she is the designated parent or guardian,
1217 [13]     and for which the patient is under 16 years of age
1218 [14]   </Description>
1219 [15]   <Target>
1220 [16]     <Subjects>
1221 [17]       <AnySubject/>
1222 [18]     </Subjects>
1223 [19]     <Resources>
1224 [20]       <Resource>
1225 [21]         <!-- match document target namespace -->
1226 [22]         <ResourceMatch
1227 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1228 [23]           <AttributeValue
1229 DataType="http://www.w3.org/2001/XMLSchema#string">
1230 [24]             http://www.medico.com/schemas/record.xsd
1231 [25]           </AttributeValue>
1232 [26]           <ResourceAttributeDesignator AttributeId=
1233 [27]             "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1234 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1235 [28]         </ResourceMatch>
1236 [29]         <!-- match requested xml element -->
1237 [30]         <ResourceMatch
1238 MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1239 [31]           <AttributeValue
1240 DataType="http://www.w3.org/2001/XMLSchema#string">/md:record</AttributeValue>
1241 [32]           <ResourceAttributeDesignator AttributeId=
1242 [33]             "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1243 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1244 [34]         </ResourceMatch>
1245 [35]       </Resource>
1246 [36]     </Resources>
1247 [37]     <Actions>
1248 [38]       <Action>
1249 [39]         <!-- match 'read' action -->
1250 [40]         <ActionMatch
1251 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1252 [41]           <AttributeValue
1253 DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1254 [42]           <ActionAttributeDesignator AttributeId=
1255 [43]             "urn:oasis:names:tc:xacml:1.0:action:action-id"
1256 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1257 [44]         </ActionMatch>
1258 [45]       </Action>
1259 [46]     </Actions>
1260 [47]   </Target>
1261 [48]   <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
1262 [49]     <!-- compare parent-guardian-id subject attribute with
1263 [50]     the value in the document -->
1264 [51]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1265 [52]     equal">
1266 [52]       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1267 [53]     and-only">
1268 [53]       <!-- parent-guardian-id subject attribute -->

```

```

1269 [54] <SubjectAttributeDesignator AttributeId=
1270 [55] "urn:oasis:names:tc:xacml:1.0:examples:attribute:
1271 [56] parent-guardian-id"
1272 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1273 [57] </Apply>
1274 [58] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1275 and-only">
1276 [59] <!-- parent-guardian-id element in the document -->
1277 [60] <AttributeSelector RequestContextPath=
1278 [61] "//md:record/md:parentGuardian/md:parentGuardianId/text()"
1279 [62] DataType="http://www.w3.org/2001/XMLSchema#string">
1280 [63] </AttributeSelector>
1281 [64] </Apply>
1282 [65] </Apply>
1283 [66] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-or-
1284 equal">
1285 [67] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-
1286 and-only">
1287 [68] <EnvironmentAttributeDesignator AttributeId=
1288 [69] "urn:oasis:names:tc:xacml:1.0:environment:current-date"
1289 DataType="http://www.w3.org/2001/XMLSchema#date"/>
1290 [70] </Apply>
1291 [71] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-
1292 yearMonthDuration">
1293 [73] <Apply
1294 FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-only">
1295 [74] <!-- patient dob recorded in the document -->
1296 [75] <AttributeSelector RequestContextPath=
1297 [76] "//md:record/md:patient/md:patientDoB/text()"
1298 DataType="http://www.w3.org/2001/XMLSchema#date">
1299 [77] </AttributeSelector>
1300 [78] </Apply>
1301 [79] <AttributeValue DataType="http://www.w3.org/TR/xquery-
1302 operators#yearMonthDuration">
1303 [80] P16Y
1304 [81] </AttributeValue>
1305 [82] </Apply>
1306 [83] </Apply>
1307 [84] </Condition>
1308 [85] </Rule>

```

1309 [02]-[47] **Rule** declaration and **rule target**. See Rule 1 in section 4.2.4.1 for the detailed
1310 explanation of these elements.

1311 [48]-[82] The `Condition` element. **Condition** must evaluate to “True” for the **rule** to be applicable.
1312 This **condition** evaluates the truth of the statement: the requestor is the designated parent or
1313 guardian and the patient is under 16 years of age.

1314 [48] The `Condition` is using the “urn:oasis:names:tc:xacml:1.0:function:and”
1315 function. This is a boolean function that takes one or more boolean arguments (2 in this case) and
1316 performs the logical “AND” operation to compute the truth value of the expression.

1317 [51]-[65] The truth of the first part of the condition is evaluated: The requestor is the designated
1318 parent or guardian. The `Apply` element contains a function invocation. The function name is
1319 contained in the `FunctionId` attribute. The comparison is done with
1320 “urn:oasis:names:tc:xacml:1.0:function:string-equal” that takes 2 arguments of
1321 “http://www.w3.org/2001/XMLSchema#string” data-type.

1322 [52] Since “urn:oasis:names:tc:xacml:1.0:function:string-equal” takes arguments
1323 of the “http://www.w3.org/2001/XMLSchema#string” data-type,
1324 “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used to ensure
1325 that the **subject attribute** “urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id” in

1326 the request **context** contains one and only one value.
1327 “urn:oasis:names:tc:xacml:1.0:function:string-equal” takes an argument
1328 expression that evaluates to a **bag** of “http://www.w3.org/2001/XMLSchema#string”
1329 values.

1330 [54] Value of the **subject attribute**
1331 “urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id” is
1332 selected from the request **context** with the <SubjectAttributeDesignator> element. This
1333 expression evaluates to a bag of “http://www.w3.org/2001/XMLSchema#string” values.

1334 [58] “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used to
1335 ensure that the **bag** of values selected by its argument contains one and only one value of data-
1336 type “http://www.w3.org/2001/XMLSchema#string”.

1337 [60] The value of the md:parentGuardianId element is selected from the **resource** content with
1338 the AttributeSelector element. AttributeSelector is a free-form XPath expression,
1339 pointing into the request **context**. The RequestContextPath XML attribute contains an XPath
1340 expression over the request **context**. Note that all namespace prefixes in the XPath expression
1341 are resolved with standard namespace declarations. The AttributeSelector evaluates to the
1342 **bag** of values of data-type “http://www.w3.org/2001/XMLSchema#string”.

1343 [66]-[83] The expression: “the patient is under 16 years of age” is evaluated. The patient is under
1344 16 years of age if the current date is less than the date computed by adding 16 to the patient’s date
1345 of birth.

1346 [66] “urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal” is used to
1347 compute the difference of two dates.

1348 [67] “urn:oasis:names:tc:xacml:1.0:function:date-one-and-only” is used to ensure
1349 that the **bag** of values selected by its argument contains one and only one value of data-type
1350 “http://www.w3.org/2001/XMLSchema#date”.

1351 [68]-[69] Current date is evaluated by selecting the
1352 “urn:oasis:names:tc:xacml:1.0:environment:current-date” **environment attribute**.

1353 [71] “urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration” is
1354 used to compute the date by adding 16 to the patient’s date of birth. The first argument is a
1355 “http://www.w3.org/2001/XMLSchema#date”, and the second argument is an
1356 “http://www.w3.org/TR/xquery-operators#yearMonthDuration”.

1357 [73] “urn:oasis:names:tc:xacml:1.0:function:date-one-and-only” is used to ensure
1358 that the **bag** of values selected by its argument contains one and only one value of data-type
1359 “http://www.w3.org/2001/XMLSchema#date”.

1360 [75]-[76] The <AttributeSelector> element selects the patient’s date of birth by taking the
1361 XPath expression over the document content.

1362 [79]-[81] Year Month Duration of 16 years.

4.2.4.3. Rule 3

1364 Rule 3 illustrates the use of an **obligation**. The XACML <Rule> element syntax does not include
1365 an element suitable for carrying an **obligation**, therefore Rule 3 has to be formatted as a
1366 <Policy> element.

```
1367 [01] <?xml version="1.0" encoding="UTF-8"?>
1368 [02] <Policy
1369 [03]     xmlns="urn:oasis:names:tc:xacml:1.0:policy"
```

```

1370 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1371 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1372 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1373 [07]   PolicyId="urn:oasis:names:tc:xacml:examples:policyid:3"
1374 [08]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
1375 [09]     rule-combining-algorithm:deny-overrides">
1376 [10] <Description>
1377 [11]   Policy for any medical record in the
1378 [12]   http://www.medico.com/schemas/record.xsd namespace
1379 [13] </Description>
1380 [14] <Target>
1381 [15]   <Subjects>
1382 [16]     <AnySubject/>
1383 [17]   </Subjects>
1384 [18]   <Resources>
1385 [19]     <Resource>
1386 [20]       <!-- match document target namespace -->
1387 [21]       <ResourceMatch
1388 [22] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1389 [23]         <AttributeValue
1390 [24] DataType="http://www.w3.org/2001/XMLSchema#string">
1391 [25]           http://www.medico.com/schemas/record.xsd
1392 [26]         </AttributeValue>
1393 [27]         <ResourceAttributeDesignator AttributeId=
1394 [28]           "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1395 [29] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1396 [30]         </ResourceMatch>
1397 [31]       </Resource>
1398 [32]     </Resources>
1399 [33]   </Actions>
1400 [34]   <AnyAction/>
1401 [35] </Actions>
1402 [36] </Target>
1403 [37] <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:3"
1404 [38]   Effect="Permit">
1405 [39]   <Description>
1406 [40]     A physician may write any medical element in a record
1407 [41]     for which he or she is the designated primary care
1408 [42]     physician, provided an email is sent to the patient
1409 [43]   </Description>
1410 [44]   <Target>
1411 [45]     <Subjects>
1412 [46]       <Subject>
1413 [47]         <!-- match subject group attribute -->
1414 [48]         <SubjectMatch
1415 [49] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1416 [50]           <AttributeValue
1417 [51] DataType="http://www.w3.org/2001/XMLSchema#string">physician</AttributeValue>
1418 [52]           <SubjectAttributeDesignator AttributeId=
1419 [53]             "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
1420 [54] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1421 [55]           </SubjectMatch>
1422 [56]         </Subject>
1423 [57]       </Subjects>
1424 [58]     </Resources>
1425 [59]   </Resource>
1426 [60]     <!-- match requested xml element -->
1427 [61]     <ResourceMatch
1428 [62] MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1429 [63]       <AttributeValue
1430 [64] DataType="http://www.w3.org/2001/XMLSchema#string">
1431 [65]         /md:record/md:medical
1432 [66]       </AttributeValue>

```



```

1433 [59] <ResourceAttributeDesignator AttributeId=
1434 [60] "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1435 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1436 [61] </ResourceMatch>
1437 [62] </Resource>
1438 [63] </Resources>
1439 [64] <Actions>
1440 [65] <Action>
1441 [66] <!-- match action -->
1442 [67] <ActionMatch
1443 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1444 [68] <AttributeValue
1445 DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
1446 [069] <ActionAttributeDesignator AttributeId=
1447 [070] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1448 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1449 [071] </ActionMatch>
1450 [072] </Action>
1451 [073] </Actions>
1452 [074] </Target>
1453 [075] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1454 equal">
1455 [076] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1456 and-only">
1457 [077] <!-- physician-id subject attribute -->
1458 [078] <SubjectAttributeDesignator AttributeId=
1459 [079] "urn:oasis:names:tc:xacml:1.0:example:
1460 [080] attribute:physician-id"
1461 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1462 [081] </Apply>
1463 [082] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1464 and-only">
1465 [083] <AttributeSelector RequestContextPath=
1466 [084] "//md:record/md:primaryCarePhysician/md:registrationID/text()"
1467 [085] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1468 [086] </Apply>
1469 [087] </Condition>
1470 [089] </Rule>
1471 [090] <Obligations>
1472 [091] <!-- send e-mail message to the document owner -->
1473 [092] <Obligation ObligationId=
1474 [093] "urn:oasis:names:tc:xacml:example:obligation:email"
1475 [094] FulfillOn="Permit">
1476 [095] <AttributeAssignment AttributeId=
1477 [096] "urn:oasis:names:tc:xacml:1.0:example:attribute:mailto"
1478 [097] DataType="http://www.w3.org/2001/XMLSchema#string">
1479 [098] <AttributeSelector RequestContextPath=
1480 [099] "//md:/record/md:patient/md:patientContact/md:email"
1481 [100] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1482 [101] </AttributeAssignment>
1483 [102] <AttributeAssignment AttributeId=
1484 [103] "urn:oasis:names:tc:xacml:1.0:example:attribute:text"
1485 [104] DataType="http://www.w3.org/2001/XMLSchema#string">
1486 [105] <AttributeValue>
1487 [106] Your medical record has been accessed by:
1488 [107] </AttributeValue>
1489 [108] </AttributeAssignment>
1490 [109] <AttributeAssignment AttributeId=
1491 [110] "urn:oasis:names:tc:xacml:example:attribute:text"
1492 [111] DataType="http://www.w3.org/2001/XMLSchema#string">
1493 [112] <SubjectAttributeDesignator AttributeId=
1494 [113] "urn:osasis:names:tc:xacml:1.0:subject:subject-id"
1495 DataType="http://www.w3.org/2001/XMLSchema#string"/>

```

```
1496 [114] </AttributeAssignment>
1497 [115] </Obligation>
1498 [116] </Obligations>
1499 [117] </Policy>
```

1500 [01]-[09] The `Policy` element includes standard namespace declarations as well as policy specific
1501 parameters, such as `PolicyId` and `RuleCombiningAlgId`.

1502 [07] **Policy** identifier. This parameter is used for the inclusion of the `Policy` in the `PolicySet`
1503 element.

1504 [08]-[09] **Rule combining algorithm** identifier. This parameter is used to compute the combined
1505 outcome of **rule effects** for **rules** that are applicable to the **decision request**.

1506 [10-13] Free-form description of the **policy**.

1507 [14]-[33] **Policy target**. The **policy target** defines a set of applicable decision requests. The
1508 structure of the `Target` element in the `Policy` is identical to the structure of the `Target` element
1509 in the `Rule`. In this case, the **policy target** is a set of all XML documents conforming to the
1510 "http://www.medico.com/schemas/record.xsd" target namespace. For the detailed description of
1511 the `Target` element see Rule 1, section 4.2.4.1.

1512 [34]-[89] The only `Rule` element included in this `Policy`. Two parameters are specified in the **rule**
1513 header: `RuleId` and `Effect`. For the detailed description of the `Rule` structure see Rule 1,
1514 section 4.2.4.1.

1515 [41]-[74] A **rule target** narrows down a **policy target**. **Decision requests** with the value of
1516 "urn:oasis:names:tc:xacml:1.0:example:attribute:role" **subject attribute** equal to
1517 "physician" [42]-[51], and that access elements of the medical record that "xpath-node-match"
1518 the "/md:record/md:medical" XPath expression [52]-[63], and that have the value of the
1519 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** equal to "read".

1520 [65]-[73] match the **target** of this **rule**. For a detailed description of the rule target see example 1,
1521 section 4.2.4.1.

1522 [75]-[87] The `Condition` element. For the **rule** to be applicable to the authorization request,
1523 **condition** must evaluate to True. This **rule condition** compares the value of the
1524 "urn:oasis:names:tc:xacml:1.0:examples:attribute:physician-id" **subject**
1525 **attribute** with the value of the `physician id` element in the medical record that is being
1526 accessed. For a detailed explanation of rule condition see Rule 1, section 4.2.4.1.

1527 [90]-[116] The `Obligations` element. **Obligations** are a set of operations that must be
1528 performed by the **PEP** in conjunction with an **authorization decision**. An **obligation** may be
1529 associated with a positive or negative **authorization decision**.

1530 [92]-[115] The `Obligation` element consists of the `ObligationId`, the authorization decision
1531 value for which it must fulfill, and a set of attribute assignments.

1532 [92]-[93] `ObligationId` identifies an **obligation**. **Obligation** names are not interpreted by the
1533 **PDP**.

1534 [94] `FulfillOn` attribute defines an **authorization decision** value for which this **obligation** must
1535 be fulfilled.

1536 [95]-[101] **Obligation** may have one or more parameters. The **obligation** parameter
1537 "urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto" is assigned the value
1538 from the content of the xml document.

1539 [95-96] `AttributeId` declares
1540 "urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto" **obligation** parameter.

1541 [97] The **obligation** parameter data-type is defined.

1542 [98]-[100] The **obligation** parameter value is selected from the content of the XML document that is
 1543 being accessed with the XPath expression over request **context**.

1544 [102]-[108] The **obligation** parameter
 1545 “urn:oasis:names:tc:xacml:1.0:examples:attribute:text” of data-type
 1546 “http://www.w3.org/2001/XMLSchema#string” is assigned the literal value “Your
 1547 medical record has been accessed by:”

1548 [109]-[114] The **obligation** parameter
 1549 “urn:oasis:names:tc:xacml:1.0:examples:attribute:text” of the
 1550 “http://www.w3.org/2001/XMLSchema#string” data-type is assigned the value of the
 1551 “urn:oasis:names:tc:xacml:1.0:subject:subject-id” **subject attribute**.

4.2.4.4. Rule 4

1552 Rule 4 illustrates the use of the "Deny" Effect value, and a Rule with no Condition element.

```

1554 [01] <?xml version="1.0" encoding="UTF-8"?>
1555 [02] <Rule
1556 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1557 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1558 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1559 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1560 [07]   RuleId="urn:oasis:names:tc:xacml:example:ruleid:4"
1561 [08]   Effect="Deny">
1562 [09]   <Description>
1563 [10]     An Administrator shall not be permitted to read or write
1564 [11]     medical elements of a patient record in the
1565 [12]     http://www.medico.com/records.xsd namespace.
1566 [13]   </Description>
1567 [14]   <Target>
1568 [15]     <Subjects>
1569 [16]       <Subject>
1570 [17]         <!-- match role subject attribute -->
1571 [18]         <SubjectMatch
1572 [19]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1573 [20]             <AttributeValue
1574 [21]               DataType="http://www.w3.org/2001/XMLSchema#string">administrator</AttributeVal
1575 [22]               ue>
1576 [23]               <SubjectAttributeDesignator AttributeId=
1577 [24]                 "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
1578 [25]               DataType="http://www.w3.org/2001/XMLSchema#string"/>
1579 [26]             </SubjectMatch>
1580 [27]           </Subject>
1581 [28]         </Subjects>
1582 [29]       <Resources>
1583 [30]         <Resource>
1584 [31]           <!-- match document target namespace -->
1585 [32]           <ResourceMatch
1586 [33]             MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1587 [34]               <AttributeValue
1588 [35]                 DataType="http://www.w3.org/2001/XMLSchema#string">
1589 [36]                   http://www.medico.com/schemas/record.xsd
1590 [37]                 </AttributeValue>
1591 [38]               <ResourceAttributeDesignator AttributeId=
1592 [39]                 "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1593 [40]               DataType="http://www.w3.org/2001/XMLSchema#string"/>
1594 [41]             </ResourceMatch>
1595 [42]           </Resource>
1596 [43]         </Resources>
1597 [44]       </Target>
1598 [45]     </Rule>
  
```

```

1596 [ 36] <ResourceMatch
1597 MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1598 [ 37] <AttributeValue
1599 DataType="http://www.w3.org/2001/XMLSchema#string">
1600 [ 38] /md:record/md:medical
1601 [ 39] </AttributeValue>
1602 [ 40] <ResourceAttributeDesignator AttributeId=
1603 [ 41] "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1604 DataType="http://www.w3.org/2001/XMLSchema#string" />
1605 [ 42] </ResourceMatch>
1606 [ 43] </Resource>
1607 [ 44] </Resources>
1608 [ 45] <Actions>
1609 [ 46] <Action>
1610 [ 47] <!-- match 'read' action -->
1611 [ 48] <ActionMatch
1612 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1613 [ 49] <AttributeValue
1614 DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1615 [ 50] <ActionAttributeDesignator AttributeId=
1616 [ 51] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1617 DataType="http://www.w3.org/2001/XMLSchema#string" />
1618 [ 52] </ActionMatch>
1619 [ 53] </Action>
1620 [ 54] <Action>
1621 [ 55] <!-- match 'write' action -->
1622 [ 56] <ActionMatch
1623 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1624 [ 57] <AttributeValue
1625 DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
1626 [ 58] <ActionAttributeDesignator AttributeId=
1627 [ 59] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1628 DataType="http://www.w3.org/2001/XMLSchema#string" />
1629 [ 60] </ActionMatch>
1630 [ 61] </Action>
1631 [ 62] </Actions>
1632 [ 63] </Target>
1633 [ 64] </Rule>

```

1634 [01]-[08] The **Rule** element declaration. The most important parameter here is **Effect**. See Rule
1635 1, section 4.2.4.1 for a detailed explanation of the **Rule** structure.

1636 [08] **Rule Effect**. Every **rule** that evaluates to “True” emits **rule effect** as its value that will be
1637 combined later on with other **rule effects** according to the **rule combining algorithm**. This **rule**
1638 **Effect** is “Deny” meaning that according to this rule, access must be denied.

1639 [09]-[13] Free form description of the **rule**.

1640 [14]-[63] **Rule target**. The **Rule target** defines a set of **decision requests** that are applicable to
1641 the **rule**. This **rule** is matched by:

- 1642 • a **decision request** with **subject attribute**
- 1643 “urn:oasis:names:tc:xacml:1.0:examples:attribute:role” equal to
- 1644 “administrator”;
- 1645 • the value of **resource attribute**
- 1646 “urn:oasis:names:tc:xacml:1.0:resource:target-namespace” is equal to
- 1647 “http://www.medico.com/schemas/record.xsd”
- 1648 • the value of the requested XML element matches the XPath expression
- 1649 “/md:record/md:medical”;

- 1650 • the value of **action attribute** "urn:oasis:names:tc:xacml:1.0:action:action-id" is equal to
1651 "read"

1652 See Rule 1, section 4.2.4.1 for the detailed explanation of the `Target` element.

1653 This **rule** does not have a `Condition` element.

1654 4.2.4.5. Example PolicySet

1655 This section uses the examples of the previous sections to illustrate the process of combining
1656 **policies**. The policy governing read access to medical elements of a record is formed from each of
1657 the four **rules described in Section 4.2.3**. In plain language, the combined rule is:

- 1658 • Either the requestor is the patient; or
- 1659 • the requestor is the parent or guardian and the patient is under 16; or
- 1660 • the requestor is the primary care physician and a notification is sent to the patient; and
- 1661 • the requestor is not an administrator.

1662 The following XACML `<PolicySet>` illustrates the combined **policies**. **Policy 3** is included by
1663 reference and **policy 2** is explicitly included.

```
1664 [01] <?xml version="1.0" encoding="UTF-8"?>
1665 [02] <PolicySet
1666 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1667 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1668 [05]   PolicySetId=
1669 [06]     "urn:oasis:names:tc:xacml:1.0:examples:policysetid:1"
1670 [07]   PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
1671 [071]     policy-combining-algorithm:deny-overrides" />
1672 [08] <Description>
1673 [09]   Example policy set.
1674 [10] </Description>
1675 [11] <Target>
1676 [12]   <Subjects>
1677 [13]     <Subject>
1678 [14]       <!-- any subject -->
1679 [15]       <AnySubject/>
1680 [16]     </Subject>
1681 [17]   </Subjects>
1682 [18]   <Resources>
1683 [19]     <Resource>
1684 [20]       <!-- any resource in the target namespace -->
1685 [21]       <ResourceMatch
1686 [22]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1687 [23]           <AttributeValue
1688 [24]             DataType="http://www.w3.org/2001/XMLSchema#string">
1689 [25]               http://www.medico.com/records.xsd
1690 [26]             </AttributeValue>
1691 [27]           <ResourceAttributeDesignator AttributeId=
1692 [28]             "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1693 [29]             DataType="http://www.w3.org/2001/XMLSchema#string" />
1694 [30]           </ResourceMatch>
1695 [31]         </Resource>
1696 [32]       </Resources>
1697 [33]     <Actions>
1698 [34]       <Action>
1699 [35]         <!-- any action -->
1700 [36]         <AnyAction/>
1701 [37]       </Action>
```

```

1702 [35] </Actions>
1703 [36] </Target>
1704 [37] <!-- include policy from the example 3 by reference -->
1705 [38] <PolicyIdReference>
1706 [39]   urn:oasis:names:tc:xacml:1.0:examples:policyid:3
1707 [40] </PolicyIdReference>
1708 [41] <!-- policy 2 combines rules from the examples 1, 2,
1709 [42]   and 4 is included by value. -->
1710 [43] <Policy>
1711 [44]   PolicyId="urn:oasis:names:tc:xacml:examples:policyid:2"
1712 [45]   RuleCombiningAlgId=
1713 [46] "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
1714 [47]   <Description>
1715 [48]     Policy for any medical record in the
1716 [49]     http://www.medico.com/schemas/record.xsd namespace
1717 [50]   </Description>
1718 [51]   <Target> ... </Target>
1719 [52]   <Rule
1720 [53]     RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"
1721 [54]     Effect="Permit"> ... </Rule>
1722 [55]   <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
1723 [56]     Effect="Permit"> ... </Rule>
1724 [57]   <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:4"
1725 [58]     Effect="Deny"> ... </Rule>
1726 [59]   <Obligations> ... </Obligations>
1727 [60] </Policy>
1728 [61] </PolicySet>
1729

```

1730 [02]-[07] PolicySet declaration. Standard XML namespace declarations are included as well as
1731 PolicySetId, and **policy combining algorithm** identifier.

1732 [05]-[06] PolicySetId is used for identifying this **policy set** and for possible inclusion of this
1733 **policy set** into another **policy set**.

1734 [07] **Policy combining algorithm** identifier. Policies in the **policy set** are combined according to
1735 the specified **policy combining algorithm** identifier when the **authorization decision** is
1736 computed.

1737 [08]-[10] Free form description of the **policy set**.

1738 [11]-[36] PolicySet Target element defines a set of **decision requests** that are applicable to
1739 this PolicySet.

1740 [38]-[40] PolicyIdReference includes **policy** by id.

1741 [43]-[60] **Policy 2** is explicitly included in this **policy set**.

1742 5. Policy syntax (normative, with the exception of 1743 the schema fragments)

1744 5.1. Element <PolicySet>

1745 The <PolicySet> element is a top-level element in the XACML policy schema. <PolicySet> is
1746 an aggregation of other **policy sets** and **policies**. **Policy sets** MAY be included in an enclosing
1747 <PolicySet> element either directly using the <PolicySet> element or indirectly using the

1748 <PolicySetIdReference> element. **Policies** MAY be included in an enclosing <PolicySet>
1749 element either directly using the <Policy> element or indirectly using the <PolicyIdReference>
1750 element.

1751 If a <PolicySet> element contains references to other **policy sets** or **policies** in the form of
1752 URLs, then these references MAY be resolvable.

1753 **Policies** included in the <PolicySet> element MUST be combined by the algorithm specified by
1754 the PolicyCombiningAlgId attribute.

1755 The <Target> element defines the applicability of the <PolicySet> to a set of **decision**
1756 **requests**. If the <Target> element within <PolicySet> matches the **request context**, then the
1757 <PolicySet> element MAY be used by the **PDP** in making its **authorization decision**.

1758 The <Obligations> element contains a set of **obligations** that MUST be fulfilled by the **PEP** in
1759 conjunction with the **authorization decision**. If the **PEP** does not understand any of the
1760 **obligations**, then it MUST act as if the **PDP** had returned a "Deny" **authorization decision** value.

```
1761 <xs:element name="PolicySet" type="xacml:PolicySetType"/>  
1762 <xs:complexType name="PolicySetType">  
1763 <xs:sequence>  
1764 <xs:element ref="xacml:Description" minOccurs="0"/>  
1765 <xs:element ref="xacml:PolicySetDefaults" minOccurs="0"/>  
1766 <xs:element ref="xacml:Target"/>  
1767 <xs:choice minOccurs="0" maxOccurs="unbounded">  
1768 <xs:element ref="xacml:PolicySet"/>  
1769 <xs:element ref="xacml:Policy"/>  
1770 <xs:element ref="xacml:PolicySetIdReference"/>  
1771 <xs:element ref="xacml:PolicyIdReference"/>  
1772 </xs:choice>  
1773 <xs:element ref="xacml:Obligations" minOccurs="0"/>  
1774 </xs:sequence>  
1775 <xs:attribute name="PolicySetId" type="xs:anyURI" use="required"/>  
1776 <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI"  
1777 use="required"/>  
1778 </xs:complexType>
```

1779 The <PolicySet> element is of **PolicySetType** complex type.

1780 The <PolicySet> element contains the following attributes and elements:

1781 PolicySetId [Required]

1782 **Policy set** identifier. It is the responsibility of the **PAP** to ensure that no two **policies**
1783 visible to the **PDP** have the same identifier. This MAY be achieved by following a
1784 predefined URN or URI scheme. If the **policy set** identifier is in the form of a URL, then it
1785 MAY be resolvable.

1786 PolicyCombiningAlgId [Required]

1787 The identifier of the **policy-combining algorithm** by which the <PolicySet>
1788 components MUST be combined. Standard **policy-combining algorithms** are listed in
1789 Appendix C. Standard **policy-combining algorithm** identifiers are listed in Section B.10.

1790 <Description> [Optional]

1791 A free-form description of the <PolicySet>.

1792 <PolicySetDefaults> [Optional]

1793 A set of default values applicable to the <PolicySet>. The scope of the
1794 <PolicySetDefaults> element SHALL be the enclosing **policy set**.

1795 <Target> [Required]

1796 The <Target> element defines the applicability of a <PolicySet> to a set of **decision**
 1797 **requests**.

1798 The <Target> element MAY be declared by the creator of the <PolicySet> or it MAY be
 1799 computed from the <Target> elements of the referenced <Policy> elements, either as
 1800 an intersection or as a union.

1801 <PolicySet> [Any Number]

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

1803 <Policy> [Any Number]

1804 A **policy** component that is included in this **policy set**.

1805 <PolicySetIdReference> [Any Number]

1806 A reference to a <PolicySet> component that MUST be included in this **policy set**. If
 1807 <PolicySetIdReference> is a URL, then it MAY be resolvable.

1808 <PolicyIdReference> [Any Number]

1809 A reference to a <Policy> component that MUST be included in this **policy set**. If the
 1810 <PolicyIdReference> is a URL, then it MAY be resolvable.

1811 <Obligations> [Optional]

1812 Contains the set of <Obligation> elements. See Section 7.11 for a description of how
 1813 the set of **obligations** to be returned by the **PDP** shall be determined.

1814 5.2. Element <Description>

1815 The <Description> element is used for a free-form description of the <PolicySet> element,
 1816 <Policy> element and <Rule> element. The <Description> element is of **xs:string** simple
 1817 type.

```
1818 <xs:element name="Description" type="xs:string"/>
```

1819 5.3. Element <PolicySetDefaults>

1820 The <PolicySetDefaults> element SHALL specify default values that apply to the
 1821 <PolicySet> element.

```
1822 <xs:element name="PolicySetDefaults" type="xacml:DefaultsType"/>
1823 <xs:complexType name="DefaultsType">
1824   <xs:sequence>
1825     <xs:choice>
1826       <xs:element ref="xacml:XPathVersion" minOccurs="0"/>
1827     </xs:choice>
1828   </xs:sequence>
1829 </xs:complexType>
```

1830 <PolicySetDefaults> element is of **DefaultsType** complex type.

1831 The <PolicySetDefaults> element contains the following elements:

1832 <XPathVersion> [Optional]

1833 Default XPath version.

1834 5.4. Element <XPathVersion>

1835 The <XPathVersion> element SHALL specify the version of the XPath specification to be used by
1836 <AttributeSelector> elements.

```
1837 <xs:element name="XPathVersion" type="xs:anyURI"/>
```

1838 The URI for the XPath 1.0 specification is "<http://www.w3.org/TR/1999/Rec-xpath-19991116>". The <XPathVersion> element is REQUIRED if the XACML enclosing **policy set**
1839 or **policy** contains <AttributeSelector> elements.
1840

1841 5.5. Element <Target>

1842 The <Target> element identifies the set of **decision requests** that the parent element is intended
1843 to evaluate. The <Target> element SHALL appear as a child of <PolicySet>, <Policy> and
1844 <Rule> elements. It contains definitions for **subjects**, **resources** and **actions**.

1845 The <Target> element SHALL contain a **conjunctive sequence** of <Subjects>, <Resources>
1846 and <Actions> elements. For the parent of the <Target> element to be applicable to the
1847 **decision request**, there MUST be at least one positive match between each section of the
1848 <Target> element and the corresponding section of the <xacml-context:Request> element.

```
1849 <xs:element name="Target" type="xacml:TargetType"/>  
1850 <xs:complexType name="TargetType">  
1851 <xs:sequence>  
1852 <xs:element ref="xacml:Subjects"/>  
1853 <xs:element ref="xacml:Resources"/>  
1854 <xs:element ref="xacml:Actions"/>  
1855 </xs:sequence>  
1856 </xs:complexType>
```

1857 The <Target> element is of **TargetType** complex type.

1858 The <Target> element contains the following elements:

1859 <Subjects> [Required]

1860 Matching specification for the **subject attributes** in the **context**.

1861 <Resources> [Required]

1862 Matching specification for the **resource attributes** in the **context**.

1863 <Actions> [Required]

1864 Matching specification for the **action attributes** in the **context**.

1865 5.6. Element <Subjects>

1866 The <Subjects> element SHALL contain a **disjunctive sequence** of <Subject> elements.

```
1867 <xs:element name="Subjects" type="xacml:SubjectsType"/>  
1868 <xs:complexType name="SubjectsType">  
1869 <xs:choice>  
1870 <xs:element ref="xacml:Subject" maxOccurs="unbounded"/>  
1871 <xs:element ref="xacml:AnySubject"/>  
1872 </xs:choice>  
1873 </xs:complexType>
```

1874 The <Subjects> element is of **SubjectsType** complex type.

1875 The <Subjects> element contains the following elements:

1876 <Subject> [One To Many, Required Choice]

1877 See section 5.7.

1878 <AnySubject> [Required Choice]

1879 See section 5.8.

1880 5.7. Element <Subject>

1881 The <Subject> element SHALL contain a *conjunctive sequence* of <SubjectMatch>
1882 elements.

```
1883 <xs:element name="Subject" type="xacml:SubjectType" />
1884 <xs:complexType name="SubjectType">
1885 <xs:sequence>
1886 <xs:element ref="xacml:SubjectMatch" maxOccurs="unbounded" />
1887 </xs:sequence>
1888 </xs:complexType>
```

1889 The <Subject> element is of **SubjectType** complex type.

1890 The <Subject> element contains the following elements:

1891 <SubjectMatch> [One to Many]

1892 A *conjunctive sequence* of individual matches of the *subject attributes* in the *context*
1893 and the embedded *attribute* values.

1894 5.8. Element <AnySubject>

1895 The <AnySubject> element SHALL match any *subject attribute* in the *context*.

```
1896 <xs:element name="AnySubject" />
```

1897 5.9. Element <SubjectMatch>

1898 The <SubjectMatch> element SHALL identify a set of *subject*-related entities by matching
1899 *attribute* values in a <xacml-context:Subject> element of the *context* with the embedded
1900 *attribute* value.

```
1901 <xs:element name="SubjectMatch" type="xacml:SubjectMatchType" />
1902 <xs:complexType name="SubjectMatchType">
1903 <xs:sequence>
1904 <xs:element ref="xacml:AttributeValue" />
1905 <xs:choice>
1906 <xs:element ref="xacml:SubjectAttributeDesignator" />
1907 <xs:element ref="xacml:AttributeSelector" />
1908 </xs:choice>
1909 </xs:sequence>
1910 <xs:attribute name="MatchId" type="xs:anyURI" use="required" />
1911 </xs:complexType>
```

1912 The <SubjectMatch> element is of **SubjectMatchType** complex type.

1913 The <SubjectMatch> element contains the following attributes and elements:

1914 MatchId [Required]

1915 Specifies a matching function. The value of this attribute MUST be of type xs:anyURI with
1916 legal values documented in Section A.12.

1917 <AttributeValue> [Required]

1918 Embedded **attribute** value.

1919 <SubjectAttributeDesignator> [Required choice]

1920 Identifies one or more **attribute** values in a <Subject> element of the **context**.

1921 <AttributeSelector> [Required choice]

1922 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
1923 expression SHOULD resolve to an **attribute** in a <Subject> element of the **context**.

1924 **5.10. Element <Resources>**

1925 The <Resources> element SHALL contain a **disjunctive sequence** of <Resource> elements.

```
1926 <xs:element name="Resources" type="xacml:ResourcesType" />  
1927 <xs:complexType name="ResourcesType">  
1928 <xs:choice>  
1929 <xs:element ref="xacml:Resource" maxOccurs="unbounded" />  
1930 <xs:element ref="xacml:AnyResource" />  
1931 </xs:choice>  
1932 </xs:complexType>
```

1933 The <Resources> element is of **ResourcesType** complex type.

1934 The <Resources> element contains the following elements:

1935 <Resource> [One To Many, Required Choice]

1936 See section 5.11.

1937 <AnyResource> [Required Choice]

1938 See section 5.12.

1939 **5.11. Element <Resource>**

1940 The <Resource> element SHALL contain a **conjunctive sequence** of <ResourceMatch>
1941 elements.

```
1942 <xs:element name="Resource" type="xacml:ResourceType" />  
1943 <xs:complexType name="ResourceType">  
1944 <xs:sequence>  
1945 <xs:element ref="xacml:ResourceMatch" maxOccurs="unbounded" />  
1946 </xs:sequence>  
1947 </xs:complexType>
```

1948 The <Resource> element is of **ResourceType** complex type.

1949 The <Resource> element contains the following elements:

1950 <ResourceMatch> [One to Many]

1951 A **conjunctive sequence** of individual matches of the **resource attributes** in the **context**
1952 and the embedded **attribute** values.

1953 **5.12. Element <AnyResource>**

1954 The <AnyResource> element SHALL match any *resource attribute* in the *context*.

```
1955 <xs:element name="AnyResource" />
```

1956 **5.13. Element <ResourceMatch>**

1957 The <ResourceMatch> element SHALL identify a set of *resource*-related entities by matching
1958 *attribute* values in the <xacml-context:Resource> element of the *context* with the embedded
1959 *attribute* value.

```
1960 <xs:element name="ResourceMatch" type="xacml:ResourceMatchType" />  
1961 <xs:complexType name="ResourceMatchType">  
1962   <xs:sequence>  
1963     <xs:element ref="xacml:AttributeValue" />  
1964     <xs:choice>  
1965       <xs:element ref="xacml:ResourceAttributeDesignator" />  
1966       <xs:element ref="xacml:AttributeSelector" />  
1967     </xs:choice>  
1968   </xs:sequence>  
1969   <xs:attribute name="MatchId" type="xs:anyMatch" use="required" />  
1970 </xs:complexType>
```

1971 The <ResourceMatch> element is of **ResourceMatchType** complex type.

1972 The <ResourceMatch> element contains the following attributes and elements:

1973 MatchId [Required]

1974 Specifies a matching function. Values of this attribute MUST be of type xs:anyURI, with
1975 legal values documented in Section A.12.

1976 <AttributeValue> [Required]

1977 Embedded *attribute* value.

1978 <ResourceAttributeDesignator> [Required Choice]

1979 Identifies one or more *attribute* values in the <Resource> element of the *context*.

1980 <AttributeSelector> [Required Choice]

1981 MAY be used to identify one or more *attribute* values in the request *context*. The XPath
1982 expression SHOULD resolve to an *attribute* in the <Resource> element of the *context*.

1983 **5.14. Element <Actions>**

1984 The <Actions> element SHALL contain a *disjunctive sequence* of <Action> elements.

```
1985 <xs:element name="Actions" type="xacml:ActionsType" />  
1986 <xs:complexType name="ActionsType">  
1987   <xs:choice>  
1988     <xs:element ref="xacml:Action" maxOccurs="unbounded" />  
1989     <xs:element ref="xacml:AnyAction" />  
1990   </xs:choice>  
1991 </xs:complexType>
```

1992 The <Actions> element is of **ActionsType** complex type.

1993 The <Actions> element contains the following elements:

1994 <Action> [One To Many, Required Choice]

1995 See section 5.15.

1996 <AnyAction> [Required Choice]

1997 See section 5.16.

1998 5.15. Element <Action>

1999 The <Action> element SHALL contain a **conjunctive sequence** of <ActionMatch> elements.

```
2000 <xs:element name="Action" type="xacml:ActionType"/>
2001 <xs:complexType name="ActionType">
2002   <xs:sequence>
2003     <xs:element ref="xacml:ActionMatch" maxOccurs="unbounded"/>
2004   </xs:sequence>
2005 </xs:complexType>
```

2006 The <Action> element is of **ActionType** complex type.

2007 The <Action> element contains the following elements:

2008 <ActionMatch> [One to Many]

2009 A **conjunctive sequence** of individual matches of the **action** attributes in the **context** and
2010 the embedded **attribute** values.

2011 5.16. Element <AnyAction>

2012 The <AnyAction> element SHALL match any **action attribute** in the **context**.

```
2013 <xs:element name="AnyAction"/>
```

2014

2015 5.17. Element <ActionMatch>

2016 The <ActionMatch> element SHALL identify a set of **action**-related entities by matching **attribute**
2017 values in the <xacml-context:Action> element of the **context** with the embedded **attribute**
2018 value.

```
2019 <xs:element name="ActionMatch" type="xacml:ActionMatchType"/>
2020 <xs:complexType name="ActionMatchType">
2021   <xs:sequence>
2022     <xs:element ref="xacml:AttributeValue"/>
2023     <xs:choice>
2024       <xs:element ref="xacml:ActionAttributeDesignator"/>
2025       <xs:element ref="xacml:AttributeSelector"/>
2026     </xs:choice>
2027   </xs:sequence>
2028   <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
2029 </xs:complexType>
```

2030 The <ActionMatch> element is of **ActionMatchType** complex type.

2031 The <ActionMatch> element contains the following attributes and elements:

2032 MatchId [Required]

2033 Specifies a matching function. The value of this attribute MUST be of type xs:anyURI, with
2034 legal values documented in Section A.12.

2035 <AttributeValue> [Required]
 2036 Embedded **attribute** value.
 2037 <ActionAttributeDesignator> [Required Choice]
 2038 Identifies one or more **attribute** values in the <Action> element of the **context**.
 2039 <AttributeSelector> [Required Choice]
 2040 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
 2041 expression SHOULD resolve to an **attribute** in the <Action> element of the **context**.

2042 **5.18. Element <PolicySetIdReference>**

2043 The <PolicySetIdReference> element SHALL be used to reference a <PolicySet> element
 2044 by id. If <PolicySetIdReference> is a URL, then it MAY be resolvable to the <PolicySet>.
 2045 The mechanism for resolving a **policy set** reference to the corresponding **policy set** is outside the
 2046 scope of this specification.

```
2047 <xs:element name="PolicySetIdReference" type="xs:anyURI"/>
```

2048 Element <PolicySetIdReference> is of **xs:anyURI** simple type.

2049 **5.19. Element <PolicyIdReference>**

2050 The <xacml:PolicyIdReference> element SHALL be used to reference a <Policy> element
 2051 by id. If <PolicyIdReference> is a URL, then it MAY be resolvable to the <Policy>. The
 2052 mechanism for resolving a **policy** reference to the corresponding **policy** is outside the scope of this
 2053 specification.

```
2054 <xs:element name="PolicyIdReference" type="xs:anyURI"/>
```

2055 Element <PolicyIdReference> is of **xs:anyURI** simple type.

2056 **5.20. Element <Policy>**

2057 The <Policy> element is the smallest entity that SHALL be presented to the **PDP** for evaluation.

2058 The main components of this element are the <Target>, <Rule> and <Obligations> elements
 2059 and the RuleCombiningAlgId attribute.

2060 The <Target> element SHALL define the applicability of the <Policy> to a set of **decision**
 2061 **requests**.

2062 **Rules** included in the <Policy> element MUST be combined by the algorithm specified by the
 2063 RuleCombiningAlgId attribute.

2064 The <Obligations> element SHALL contain a set of **obligations** that MUST be fulfilled by the
 2065 **PDP** in conjunction with the **authorization decision**.

```
2066 <xs:element name="Policy" type="xacml:PolicyType"/>
2067 <xs:complexType name="PolicyType">
2068 <xs:sequence>
2069 <xs:element ref="xacml:Description" minOccurs="0"/>
2070 <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>
2071 <xs:element ref="xacml:Target"/>
2072 <xs:element ref="xacml:Rule" minOccurs="0" maxOccurs="unbounded"/>
2073 <xs:element ref="xacml:Obligations" minOccurs="0"/>
2074 </xs:sequence>
```

```

2075     <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
2076     <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>
2077 </xs:complexType>

```

2078 The <Policy> element is of **PolicyType** complex type.

2079 The <Policy> element contains the following attributes and elements:

2080 PolicyId [Required]

2081 **Policy** identifier. It is the responsibility of the **PAP** to ensure that no two **policies** visible to
 2082 the **PDP** have the same identifier. This MAY be achieved by following a predefined URN or
 2083 URI scheme. If the **policy** identifier is in the form of a URL, then it MAY be resolvable.

2084 RuleCombiningAlgId [Required]

2085 The identifier of the rule-combining algorithm by which the <Policy> components MUST
 2086 be combined. Standard rule-combining algorithms are listed in Appendix C. Standard rule-
 2087 combining algorithm identifiers are listed in Section B.10.

2088 <Description> [Optional]

2089 A free-form description of the **policy**. See Section 5.2 Element <Description>.

2090 <PolicyDefaults> [Optional]

2091 Defines a set of default values applicable to the **policy**. The scope of the
 2092 <PolicyDefaults> element SHALL be the enclosing policy.

2093 <Target> [Required]

2094 The <Target> element SHALL define the applicability of a <Policy> to a set of **decision**
 2095 **requests**.

2096 The <Target> element MAY be declared by the creator of the <Policy> element, or it
 2097 MAY be computed from the <Target> elements of the referenced <Rule> elements either
 2098 as an intersection or as a union.

2099 <Rule> [Any Number]

2100 A sequence of authorizations that MUST be combined according to the
 2101 RuleCombiningAlgId attribute. **Rules** whose <Target> elements match the **decision**
 2102 **request** MUST be considered. **Rules** whose <Target> elements do not match the
 2103 **decision request** SHALL be ignored.

2104 <Obligations> [Optional]

2105 A **conjunctive sequence** of **obligations** that MUST be fulfilled by the **PEP** in conjunction
 2106 with the **authorization decision**. See Section 7.11 for a description of how the set of
 2107 **obligations** to be returned by the **PDP** SHALL be determined.

2108 5.21. Element <PolicyDefaults>

2109 The <PolicyDefaults> element SHALL specify default values that apply to the <Policy>
 2110 element.

```

2111     <xs:element name="PolicyDefaults" type="xacml:DefaultsType"/>
2112     <xs:complexType name="DefaultsType">
2113       <xs:sequence>
2114         <xs:choice>
2115           <xs:element ref="xacml:XPathVersion" minOccurs="0"/>

```

```
2116     </xs:choice>
2117     </xs:sequence>
2118 </xs:complexType>
```

2119 <PolicyDefaults> element is of **DefaultsType** complex type.

2120 The <PolicyDefaults> element contains the following elements:

2121 <XPathVersion> [Optional]

2122 Default XPath version.

2123 **5.22. Element <Rule>**

2124 The <Rule> element SHALL define the individual **rules** in the **policy**. The main components of
2125 this element are the <Target> and <Condition> elements and the **Effect** attribute.

```
2126     <xs:element name="Rule" type="xacml:RuleType" />
2127     <xs:complexType name="RuleType">
2128       <xs:sequence>
2129         <xs:element ref="xacml:Description" minOccurs="0" />
2130         <xs:element ref="xacml:Target" minOccurs="0" />
2131         <xs:element ref="xacml:Condition" minOccurs="0" />
2132       </xs:sequence>
2133       <xs:attribute name="RuleId" type="xs:anyURI" use="required" />
2134       <xs:attribute name="Effect" type="xacml:EffectType" use="required" />
2135     </xs:complexType>
```

2136 The <Rule> element is of **RuleType** complex type.

2137 The <Rule> element contains the following attributes and elements:

2138 RuleId [Required]

2139 A URN identifying this **rule**.

2140 Effect [Required]

2141 **Rule effect.** Values of this attribute are either “Permit” or “Deny”.

2142 <Description> [optional]

2143 A free-form description of the **rule**.

2144 <Target> [optional]

2145 Identifies the set of **decision requests** that the <Rule> element is intended to evaluate. If
2146 this element is omitted, then the **target** for the <Rule> SHALL be defined by the
2147 <Target> element of the enclosing <Policy> element. See Section 5.5 for details.

2148 <Condition> [optional]

2149 A **predicate** that MUST be satisfied for the **rule** to be assigned its **Effect** value. A
2150 **condition** is a boolean function over a combination of **subject**, **resource**, **action** and
2151 **environment attributes** or other functions.

2152 **5.23. Simple type EffectType**

2153 The **EffectType** simple type defines the values allowed for the **Effect** attribute of the <Rule>
2154 element and for the **FulfillOn** attribute of the <Obligation> element.

```
2155     <xs:simpleType name="EffectType">
```



```

2156     <xs:restriction base="xs:string">
2157         <xs:enumeration value="Permit"/>
2158         <xs:enumeration value="Deny"/>
2159     </xs:restriction>
2160 </xs:simpleType>

```

2161 5.24. Element <Condition>

2162 The <Condition> element is a boolean function over **subject**, **resource**, **action** and
 2163 **environment attributes** or functions of **attributes**. If the <Condition> element evaluates to
 2164 "True", then the enclosing <Rule> element is assigned its Effect value.

```

2165     <xs:element name="Condition" type="xacml:ApplyType"/>

```

2166 The <Condition> element is of **ApplyType** complex type.

2167 5.25. Element <Apply>

2168 The <Apply> element denotes application of a function to its arguments, thus encoding a function
 2169 call. The <Apply> element can be applied to any combination of <Apply> ,
 2170 <AttributeValue> , <SubjectAttributeDesignator> ,
 2171 <ResourceAttributeDesignator> , <ActionAttributeDesignator> ,
 2172 <EnvironmentAttributeDesignator> and <AttributeSelector> arguments.

```

2173     <xs:element name="Apply" type="xacml:ApplyType"/>
2174     <xs:complexType name="ApplyType">
2175         <xs:choice minOccurs="0" maxOccurs="unbounded">
2176             <xs:element ref="xacml:Function"/>
2177             <xs:element ref="xacml:Apply"/>
2178             <xs:element ref="xacml:AttributeValue"/>
2179             <xs:element ref="xacml:SubjectAttributeDesignator"/>
2180             <xs:element ref="xacml:ResourceAttributeDesignator"/>
2181             <xs:element ref="xacml:ActionAttributeDesignator"/>
2182             <xs:element ref="xacml:EnvironmentAttributeDesignator"/>
2183             <xs:element ref="xacml:AttributeSelector"/>
2184         </xs:choice>
2185         <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>
2186     </xs:complexType>

```

2187 The <Apply> element is of **ApplyType** complex type.

2188 The <Apply> element contains the following attributes and elements:

2189 FunctionId [Required]

2190 The URN of a function. XACML-defined functions are described in Appendix A.

2191 <Function> [Optional]

2192 The name of a function that is applied to the elements of a **bag**. See section A14.11.

2193 <Apply> [Optional]

2194 A nested function-call argument.

2195 <AttributeValue> [Optional]

2196 A literal value argument.

2197 <SubjectAttributeDesignator> [Optional]

2198 A **subject attribute** argument.

2199 <ResourceAttributeDesignator> [Optional]

2200 A **resource attribute** argument.

2201 <ActionAttributeDesignator> [Optional]

2202 An **action attribute** argument.

2203 <EnvironmentAttributeDesignator> [Optional]

2204 An **environment attribute** argument.

2205 <AttributeSelector> [Optional]

2206 An **attribute** selector argument.

2207 5.26. Element <Function>

2208 The `Function` element SHALL be used to name a function that is applied by the higher-order **bag**
2209 functions to every element of a **bag**. The higher-order **bag** functions are described in Section
2210 A14.11.

```
2211 <xs:element name="Function" type="xacml:FunctionType"/>  
2212 <xs:complexType name="FunctionType">  
2213   <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>  
2214 </xs:complexType>
```

2215 The `Function` element is of **FunctionType** complex type.

2216 The `Function` element contains the following attributes:

2217 `FunctionId` [Required]

2218 The identifier for the function that is applied to the elements of a **bag** by the higher-order
2219 **bag** functions.

2220 5.27. Complex type `AttributeDesignatorType`

2221 The **AttributeDesignatorType** complex type is the type for elements and extensions that identify
2222 **attributes**. An element of this type contains properties by which it MAY be matched to **attributes**
2223 in the request **context**.

2224 In addition, elements of this type MAY control behaviour in the event that no matching **attribute** is
2225 present in the **context**.

2226 Elements of this type SHALL NOT alter the match semantics of named **attributes**, but MAY narrow
2227 the search space.

```
2228 <xs:complexType name="AttributeDesignatorType">  
2229   <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2230   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>  
2231   <xs:attribute name="Issuer" type="xs:anyURI" use="optional"/>  
2232   <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"  
2233   default="false"/>  
2234 </xs:complexType>
```

2235 A named **attribute** SHALL match an **attribute** if the values of their respective `AttributeId`,
2236 `DataType` and `Issuer` attributes match. The **attribute** designator's `AttributeId` MUST match,
2237 by URI equality, the `AttributeId` of the **attribute**. The **attribute** designator's `DataType` MUST
2238 match, by URI equality, the `DataType` of the same **attribute**.

2239 If the `Issuer` attribute is present in the **attribute** designator, then it MUST match, by URI equality,
2240 the `Issuer` of the same **attribute**. If the `Issuer` is not present in the **attribute** designator, then
2241 the matching of the **attribute** to the named **attribute** SHALL be governed by `AttributeId` and
2242 `DataType` attributes alone.

2243 The `<AttributeDesignatorType>` contains the following attributes:

2244 `AttributeId` [Required]

2245 This attribute SHALL specify the `AttributeId` with which to match the **attribute**.

2246 `DataType` [Required]

2247 This attribute SHALL specify the data-type with which to match the **attribute**.

2248 `Issuer` [Optional]

2249 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **attribute**.

2250 `MustBePresent` [Optional]

2251 This attribute governs whether the element returns “Indeterminate” in the case where the
2252 the named **attribute** is absent. If the *named attribute* is absent and `MustBePresent` is
2253 “True”, then this element SHALL result in “Indeterminate”. The default value SHALL be
2254 “False”.

2255 5.28. Element `SubjectAttributeDesignator`

2256 The `<SubjectAttributeDesignator>` element is of the **SubjectAttributeDesignatorType**.
2257 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType**
2258 complex type. It is the base type for elements and extensions that refer to *named categorized*
2259 **subject attributes**. A *named categorized subject attribute* is defined as follows:

2260 A **subject** is represented by a `<Subject>` element in the `<xacml-context:Request>` element.
2261 Each `<Subject>` element SHALL contain the XML attribute `SubjectCategory`. This attribute is
2262 called the *subject category attribute*.

2263 A *categorized subject* is a **subject** that is identified by a particular *subject category attribute*.

2264 A **subject attribute** is an **attribute** of a particular **subject**, i.e. contained within a `<Subject>`
2265 element.

2266 A *named subject attribute* is a *named attribute* for a **subject**.

2267 A *named categorized subject attribute* is a *named subject attribute* for a particular *categorized*
2268 **subject**.

2269 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType** with a
2270 `SubjectCategory` attribute. The **SubjectAttributeDesignatorType** extends the match
2271 semantics of the **AttributeDesignatorType** such that it narrows the **attribute** search space to the
2272 specific *categorized subject* such that the value of this element’s `SubjectCategory` attribute
2273 matches, by string-equality, the value of the `<Request>` element’s *subject category attribute*.

2274 If there are multiple **subjects** with the same `SubjectCategory` xml attribute, then they SHALL be
2275 treated as if they were one *categorized subject*.

2276 Elements and extensions of the **SubjectAttributeDesignatorType** complex type determine the
2277 presence of select **attribute values** associated with *named categorized subject attributes*.

2278 Elements and extensions of the **SubjectAttributeDesignatorType** SHALL NOT alter the match
2279 semantics of *named categorized subject attributes*, but MAY narrow the search space.

```
2280 <xs:complexType name="SubjectAttributeDesignatorType">  
2281   <xs:complexContent>  
2282     <xs:extension base="xacml:AttributeDesignatorType">  
2283       <xs:attribute name="SubjectCategory"  
2284         type="xs:anyURI"  
2285         use="optional"  
2286         default=  
2287           "urn:oasis:tc:xacml:1.0:subject-category:access-subject" />  
2288     </xs:extension>  
2289   </xs:complexContent>  
2290 </xs:complexType>
```

2291 The <SubjectAttributeDesignatorType> complex type contains the following attribute in
2292 addition to the attributes of the **AttributeDesignatorType** complex type:

2293 SubjectCategory [Optional]

2294 This attribute SHALL specify the *categorized subject* from which to match *named subject*
2295 **attributes**. If SubjectCategory is not present, then its default value of
2296 "urn:oasis:tc:xacml:1.0:subject-category:access-subject" SHALL be used.

2297 **5.29. Element <ResourceAttributeDesignator>**

2298 The <ResourceAttributeDesignator> element retrieves a **bag** of values for a *named*
2299 **resource attribute**. A **resource attribute** is an **attribute** contained within the <Resource>
2300 element of the <xacml-context:Request> element. A *named resource attribute* is a *named*
2301 **attribute** that matches a **resource attribute**. A *named resource attribute* SHALL be considered
2302 *present* if there is at least one **resource attribute** that matches the criteria set out below. A
2303 **resource attribute** value is an **attribute** value that is contained within a **resource attribute**.

2304 The <ResourceAttributeDesignator> element SHALL return a **bag** containing all the
2305 **resource attribute** values that are matched by the *named resource attribute*. The
2306 `MustBePresent` attribute governs whether this element returns an empty **bag** or "Indeterminate"
2307 in the case that the *named resource attribute* is absent. If the *named resource attribute* is not
2308 present and the `MustBePresent` attribute is "False" (its default value), then this element SHALL
2309 evaluate to an empty **bag**. If the *named resource attribute* is not present and the
2310 `MustBePresent` attribute is "True", then this element SHALL evaluate to "Indeterminate".
2311 Regardless of the `MustBePresent` attribute, if it cannot be determined whether the *named*
2312 **resource attribute** is present or not in the **request context**, or the value of the *named resource*
2313 **attribute** is unavailable, then the expression SHALL evaluate to "Indeterminate".

2314 A *named resource attribute* SHALL match a **resource attribute** as per the match semantics
2315 specified in the **AttributeDesignatorType** complex type [Section 5.27]

2316 The <ResourceAttributeDesignator> MAY appear in the <ResourceMatch> element and
2317 MAY be passed to the <Apply> element as an argument.

```
2318 <xs:element name="ResourceAttributeDesignator"  
2319   type="xacml:AttributeDesignatorType" />
```

2320 The <ResourceAttributeDesignator> element is of the **AttributeDesignatorType**
2321 complex type.

2322

5.30. Element <ActionAttributeDesignator>

2323 The <ActionAttributeDesignator> element retrieves a **bag** of values for a *named action*
2324 **attribute**. An **action attribute** is an **attribute** contained within the <Action> element of the
2325 <xacml-context:Request> element. A *named action attribute* has specific criteria (described
2326 below) with which to match an **action attribute**. A *named action attribute* SHALL be considered
2327 *present*, if there is at least one **action attribute** that matches the criteria. An **action attribute value**
2328 is an **attribute value** that is contained within an **action attribute**.

2329 The <ActionAttributeDesignator> element SHALL return a **bag** of all the **action attribute**
2330 values that are matched by the *named action attribute*. The `MustBePresent` attribute governs
2331 whether this element returns an empty **bag** or “Indeterminate” in the case that the *named action*
2332 **attribute** is absent. If the *named action attribute* is not present and the `MustBePresent` attribute
2333 is “False” (its default value), then this element SHALL evaluate to an empty **bag**. If the *named*
2334 **action attribute** is not present and the `MustBePresent` attribute is “True”, then this element
2335 SHALL evaluate to “Indeterminate”. Regardless of the `MustBePresent` attribute, if it cannot be
2336 determined whether the *named action attribute* is present or not present in the request **context**, or
2337 the value of the *named action attribute* is unavailable, then the expression SHALL evaluate to
2338 “Indeterminate”.

2339 A *named action attribute* SHALL match an **action attribute** as per the match semantics specified
2340 in the **AttributeDesignatorType** complex type [Section 5.27].

2341 The <ActionAttributeDesignator> MAY appear in the <ActionMatch> element and MAY
2342 be passed to the <Apply> element as an argument.

```
2343 <xs:element name="ActionAttributeDesignator"  
2344           type="xacml:AttributeDesignatorType" />
```

2345 The <ActionAttributeDesignator> element is of the **AttributeDesignatorType** complex
2346 type.

2347

5.31. Element <EnvironmentAttributeDesignator>

2348 The <EnvironmentAttributeDesignator> element retrieves a **bag** of values for a *named*
2349 **environment attribute**. An **environment attribute** is an **attribute** contained within the
2350 <Environment> element of the <xacml-context:Request> element. A *named environment*
2351 **attribute** has specific criteria (described below) with which to match an **environment attribute**. A
2352 *named environment attribute* SHALL be considered *present*, if there is at least one **environment**
2353 **attribute** that matches the criteria. An **environment attribute value** is an **attribute value** that is
2354 contained within an **environment attribute**.

2355 The <EnvironmentAttributeDesignator> element SHALL evaluate to a **bag** of all the
2356 **environment attribute** values that are matched by the *named environment attribute*. The
2357 `MustBePresent` attribute governs whether this element returns an empty **bag** or “Indeterminate”
2358 in the case that the *named environment attribute* is absent. If the *named environment attribute*
2359 is not present and the `MustBePresent` attribute is “False” (its default value), then this element
2360 SHALL evaluate to an empty **bag**. If the *named environment attribute* is not present and the
2361 `MustBePresent` attribute is “True”, then this element SHALL evaluate to “Indeterminate”.
2362 Regardless of the `MustBePresent` attribute, if it cannot be determined whether the *named*
2363 **environment attribute** is present or not present in the request **context**, or the value of the *named*
2364 **environment attribute** is unavailable, then the expression SHALL evaluate to “Indeterminate”.

2365 A *named environment attribute* SHALL match an **environment attribute** as per the match
2366 semantics specified in the **AttributeDesignatorType** complex type [Section 5.27].

2367 The <EnvironmentAttributeDesignator> MAY be passed to the <Apply> element as an
2368 argument.

```
2369 <xs:element name="EnvironmentAttributeDesignator"  
2370 type="xacml:AttributeDesignatorType" />
```

2371 The <EnvironmentAttributeDesignator> element is of the **AttributeDesignatorType**
2372 complex type.

2373 **5.32. Element <AttributeSelector>**

2374 The **AttributeSelector** element's **RequestContextPath** XML attribute SHALL contain a
2375 legal XPath expression whose context node is the <xacml-context:Request> element. The
2376 **AttributeSelector** element SHALL evaluate to a **bag** of values whose data-type is specified by
2377 the element's **DataType** attribute. If the **DataType** specified in the **AttributeSelector** is a
2378 primitive data type defined in [XQO] or [XS], then the value returned by the XPath expression
2379 SHALL be converted to the **DataType** specified in the **AttributeSelector** using the constructor
2380 function below [from XQO] that corresponds to the **DataType**. If an error results from using the
2381 constructor function, then the value of the **AttributeSelector** SHALL be "Indeterminate".

```
2382  
2383 xs:string()  
2384 xs:boolean()  
2385 xs:integer()  
2386 xs:double()  
2387 xs:dateTime()  
2388 xs:date()  
2389 xs:time()  
2390 xs:hexBinary()  
2391 xs:base64Binary()  
2392 xf:anyURI()  
2393 fn:yearMonthDuration()  
2394 fn:dayTimeDuration()
```

2395
2396 If the **DataType** specified in the **AttributeSelector** is not one of the preceding primitive
2397 **DataTypes**, then the **AttributeSelector** SHALL return a bag of instances of the specified
2398 **DataType**. If there are errors encountered in converting the values returned by the XPath
2399 expression to the specified **DataType**, then the result of the **AttributeSelector** SHALL be
2400 "Indeterminate".

2401
2402 If the policy writer intends to select the string value of an element's contents rather than the node
2403 representing the element itself, then the XPath expression MUST terminate in "/text()". The
2404 resulting sequence of string-data SHALL be converted to a **bag** of values of the type that is implied
2405 by the type system.

2406 Support for the <AttributeSelector> element is OPTIONAL.

```
2407 <xs:element name="AttributeSelector" type="xacml:AttributeSelectorType" />  
2408 <xs:complexType name="AttributeSelectorType">  
2409 <xs:attribute name="RequestContextPath" type="xs:string" use="required" />  
2410 <xs:attribute name="DataType" type="xs:anyURI" use="required" />  
2411 <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"  
2412 default="false"  
2413 </xs:complexType>
```

2414 The <AttributeSelector> element is of **AttributeSelectorType** complex type.

2415 The <AttributeSelector> element has the following attributes:

2416 **RequestContextPath** [Required]

2417 An XPath expression whose context node is the <xacml-context:Request> element.
2418 There SHALL be no restriction on the XPath syntax.

2419 DataType [Required]

2420 The bag of values returned by the AttributeSelector SHALL be of this data type.

2421 MustBePresent [Optional]

2422 Whether or not the designated *attribute* must be present in the *context*.

2423 5.33. Element <AttributeValue>

2424 The <AttributeValue> element SHALL contain a literal *attribute* value.

```
2425 <xs:element name="AttributeValue" type="xacml:AttributeValueType" />
2426 <xs:complexType name="AttributeValueType" mixed="true">
2427   <xs:sequence>
2428     <xs:any namespace="##any" processContents="lax" minOccurs="0"
2429     maxOccurs="unbounded" />
2430   </xs:sequence>
2431   <xs:attribute name="DataType" type="xs:anyURI" use="required" />
2432   <xs:anyAttribute namespace="##any" processContents="lax" />
2433 </xs:complexType>
```

2434 The <AttributeValue> element is of **AttributeValueType** complex type.

2435 The <AttributeValue> element has the following attributes:

2436 DataType [Required]

2437 The data-type of the *attribute* value.

2438 5.34. Element <Obligations>

2439 The <Obligations> element SHALL contain a set of <Obligation> elements.

2440 Support for the <Obligations> element is OPTIONAL.

```
2441 <xs:element name="Obligations" type="xacml:ObligationsType" />
2442 <xs:complexType name="ObligationsType">
2443   <xs:sequence>
2444     <xs:element ref="xacml:Obligation" maxOccurs="unbounded" />
2445   </xs:sequence>
2446 </xs:complexType>
```

2447 The <Obligations> element is of **ObligationsType** complexType.

2448 The <Obligations> element contains the following element:

2449 <Obligation> [One to Many]

2450 A sequence of *obligations*

2451 5.35. Element <Obligation>

2452 The <Obligation> element SHALL contain an identifier for the *obligation* and a set of *attributes*
2453 that form arguments of the action defined by the *obligation*. The FulfillOn attribute SHALL
2454 indicate the *effect* for which this *obligation* applies.

```
2455 <xs:element name="Obligation" type="xacml:ObligationType" />
```



```

2456 <xs:complexType name="ObligationType">
2457 <xs:sequence>
2458 <xs:element ref="xacml:AttributeAssignment" maxOccurs="unbounded"/>
2459 </xs:sequence>
2460 <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
2461 <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required"/>
2462 </xs:complexType>

```

2463 The <Obligation> element is of **ObligationType** complexType. See Section 7.11 for a
 2464 description of how the set of **obligations** to be returned by the PDP is determined.

2465 The <Obligation> element contains the following elements and attributes:

2466 ObligationId [required]

2467 **Obligation** identifier. The value of the **obligation** identifier SHALL be interpreted by the
 2468 **PEP**.

2469 FulfillOn [required]

2470 The **effect** for which this **obligation** applies.

2471 <AttributeAssignment> [required]

2472 **Obligation** arguments assignment. The values of the **obligation** arguments SHALL be
 2473 interpreted by the **PEP**.

2474 5.36. Element <AttributeAssignment>

2475 The <AttributeAssignment> element SHALL contain an AttributeId and the corresponding
 2476 **attribute** value. The AttributeId is part of **attribute** meta-data, and is used when the **attribute**
 2477 cannot be referenced by its location in the <xacml-context:Request>. This situation may arise
 2478 in an <Obligation> element if the **obligation** includes parameters.

```

2479 <xs:element name="AttributeAssignment"
2480 type="xacml:AttributeAssignmentType"/>
2481 <xs:complexType name="AttributeAssignmentType" mixed="true">
2482 <xs:complexContent>
2483 <xs:extension base="xacml:AttributeValueType">
2484 <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
2485 </xs:extension>
2486 </xs:complexContent>
2487 </xs:complexType>

```

2488 The <AttributeAssignment> element is of **AttributeAssignmentType** complex type.

2489 The <AttributeAssignment> element contains the following attributes:

2490 AttributeId [Required]

2491 The **attribute** Identifier

2492 DataType [Required]

2493 The data-type for the assigned value.

2494

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

2495

2496

6.1. Element <Request>

2497

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

2498

2499

2500

2501

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

2502

2503

2504

2505

```
<xs:element name="Request" type="xacml-context:RequestType"/>
<xs:complexType name="RequestType">
  <xs:sequence>
    <xs:element ref="xacml-context:Subject" maxOccurs="unbounded"/>
    <xs:element ref="xacml-context:Resource"/>
    <xs:element ref="xacml-context:Action"/>
    <xs:element ref="xacml-context:Environment" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

2506

2507

2508

2509

2510

2511

2512

2513

2514

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

2515

The <Request> element contains the following elements:

2516

<Subject> [One to Many]

2517

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

2518

2519

2520

2521

2522

2523

2524

2525

2526

<Resource> [Required]

2527

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

2528

2529

2530

<Action> [Required]

2531

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

2532

2533

<Environment> [Optional]

2534

Contains a set of <Attribute> elements of the *environment*. These <Attribute> elements MAY form a part of *policy* evaluation.

2535

2536

6.2. Element <Subject>

2537 The <Subject> element specifies a **subject** by listing a sequence of <Attribute> elements
2538 associated with the **subject**.

```
2539 <xs:element name="Subject" type="xacml-context:SubjectType"/>
2540 <xs:complexType name="SubjectType">
2541 <xs:sequence>
2542 <xs:element ref="xacml-context:Attribute" minOccurs="0"
2543 maxOccurs="unbounded"/>
2544 </xs:sequence>
2545 <xs:attribute name="SubjectCategory" type="xs:anyURI" use="optional"
2546 default="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"/>
2547 </xs:complexType>
```

2548 The <Subject> element is of **SubjectType** complex type.

2549 The <Subject> element contains the following elements:

2550 SubjectCategory [Optional]

2551 This attribute indicates the role that the parent <Subject> played in the formation of the
2552 access request. If this attribute is not present in a given <Subject> element, then the
2553 default value of "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" SHALL be
2554 used, indicating that the parent <Subject> element represents the entity ultimately
2555 responsible for initiating the **access** request.

2556 If more than one <Subject> element contains a "urn:oasis:names:tc:xacml:1.0:subject-
2557 category" attribute with the same value, then the PDP SHALL treat the contents of those
2558 elements as if they were contained in the same <Subject> element.

2559 <Attribute> [Any Number]

2560 A sequence of attributes that apply to the subject.

2561 Typically, a <Subject> element will contain an <Attribute> with an AttributeId of
2562 "urn:oasis:names:tc:xacml:1.0:subject:subject-id", containing the identity of the **subject**.

2563 A <Subject> element MAY contain additional <Attribute> elements.

2564 6.3. Element <Resource>

2565 The <Resource> element specifies information about the **resource** to which **access** is requested,
2566 by listing a sequence of <Attribute> elements associated with the **resource**. It MAY include the
2567 **resource** content.

```
2568 <xs:element name="Resource" type="xacml-context:ResourceType"/>
2569 <xs:complexType name="ResourceType">
2570 <xs:sequence>
2571 <xs:element ref="xacml-context:ResourceContent" minOccurs="0"/>
2572 <xs:element ref="xacml-context:Attribute" minOccurs="0"
2573 maxOccurs="unbounded"/>
2574 </xs:sequence>
2575 </xs:complexType>
```

2576 The <Resource> element is of **ResourceType** complex type.

2577 The <Resource> element contains the following elements:

2578 <ResourceContent> [Optional]

2579 The **resource** content.

2580 <Attribute> [Any Number]

2581 A sequence of **resource attributes**. The <Resource> element MUST contain one and
 2582 only one <Attribute> with an AttributeId of
 2583 "urn:oasis:names:tc:xacml:1.0:resource:resource-id". This **attribute**
 2584 specifies the identity of the **resource** to which **access** is requested.

2585 A <Resource> element MAY contain additional <Attribute> elements.

2586 6.4. Element <ResourceContent>

2587 The <ResourceContent> element is a notional placeholder for the **resource** content. If an
 2588 XACML **policy** references the contents of the **resource**, then the <ResourceContent> element
 2589 SHALL be used as the reference point.

```
2590 <xs:complexType name="ResourceContentType" mixed="true">
2591 <xs:sequence>
2592 <xs:any namespace="##any" processContents="lax" minOccurs="0"
2593 maxOccurs="unbounded" />
2594 </xs:sequence>
2595 <xs:anyAttribute namespace="##any" processContents="lax" />
2596 </xs:complexType>
```

2597 The <ResourceContent> element is of **ResourceContentType** complex type.

2598 The <ResourceContent> element allows arbitrary elements and attributes.

2599 6.5. Element <Action>

2600 The <Action> element specifies the requested **action** on the **resource**, by listing a set of
 2601 <Attribute> elements associated with the **action**.

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

2609 The <Action> element is of **ActionType** complex type.

2610 The <Action> element contains the following elements:

2611 <Attribute> [Any Number]

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

2613 6.6. Element <Environment>

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

```
2616 <xs:element name="Environment" type="xacml-context:EnvironmentType" />
2617 <xs:complexType name="EnvironmentType">
2618 <xs:sequence>
2619 <xs:element ref="xacml-context:Attribute" minOccurs="0"
2620 maxOccurs="unbounded" />
2621 </xs:sequence>
2622 </xs:complexType>
```

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

2624 The <Environment> element contains the following elements:

2625 <Attribute> [Any Number]

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

2629 6.7. Element <Attribute>

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

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

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

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

2646 AttributeId [Required]

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

2649 DataType [Required]

2650 The data-type of the contents of the <AttributeValue> element. This SHALL be either
2651 a primitive type defined by the XACML 1.0 specification or a type defined in a namespace
2652 declared in the <xacml-context> element.

2653 Issuer [Optional]

2654 **Attribute** issuer. This attribute value MAY be an x500Name that binds to a public key, or it
2655 may be some other identifier exchanged out-of-band by issuing and relying parties.

2656 IssueInstant [Optional]

2657 The date and time at which the **attribute** was issued.

2658 <AttributeValue> [Optional]

2659 At most one **attribute** value.

2660 6.8. Element <AttributeValue>

2661 The <AttributeValue> element contains the value of an **attribute**.

```
2662 <xs:element name="AttributeValue" type="xacml-context:AttributeValueType"/>
```

```

2663     <xs:complexType name="AttributeValueType" mixed="true">
2664         <xs:sequence>
2665             <xs:any namespace="##any" processContents="lax" minOccurs="0"
2666             maxOccurs="unbounded" />
2667         </xs:sequence>
2668         <xs:anyAttribute namespace="##any" processContents="lax" />
2669     </xs:complexType>

```

2670 The <AttributeValue> element is of **AttributeValueType** type.

2671 The data-type of the <AttributeValue> MAY be specified by using the `DataType` attribute of
2672 the parent <Attribute> element.

2673 **6.9. Element <Response>**

2674 The <Response> element is a top-level element in the XACML *context* schema. The
2675 <Response> element is an abstraction layer used by the *policy* language. Any proprietary system
2676 using the XACML specification MUST transform an XACML *context* <Response> into the form of
2677 its *authorization decision*.

2678 The <Response> element encapsulates the *authorization decision* produced by the *PDP*. It
2679 includes a sequence of one or more results, with one <Result> element per requested *resource*.
2680 Multiple results MAY be returned when the value of the "urn:oasis:xacml:1.0:resource:scope"
2681 resource *attribute* in the request *context* is "Descendants" or "Children". Support for multiple
2682 results is OPTIONAL.

```

2683     <xs:element name="Response" type="xacml-context:ResponseType" />
2684     <xs:complexType name="ResponseType">
2685         <xs:sequence>
2686             <xs:element ref="xacml-context:Result" maxOccurs="unbounded" />
2687         </xs:sequence>
2688     </xs:complexType>

```

2689 The <Response> element is of **ResponseType** complex type.

2690 The <Response> element contains the following elements:

2691 <Result> [One to Many]

2692 An authorization decision result.

2693 **6.10. Element <Result>**

2694 The <Result> element represents an *authorization decision* result for the *resource* specified by
2695 the `ResourceId` *attribute*. It MAY include a set of *obligations* that MUST be fulfilled by the *PEP*.
2696 If the *PEP* does not understand an *obligation*, then it MUST act as if the *PDP* had denied *access*
2697 to the requested *resource*.

```

2698     <xs:element name="Result" type="xacml-context:ResultType" />
2699     <xs:complexType name="ResultType">
2700         <xs:sequence>
2701             <xs:element ref="xacml-context:Decision" />
2702             <xs:element ref="xacml-context:Status" />
2703             <xs:element ref="xacml:Obligations" minOccurs="0" />
2704         </xs:sequence>
2705         <xs:attribute name="ResourceId" type="xs:string" use="optional" />
2706     </xs:complexType>

```

2707 The <Result> element is of **ResultType** complex type.

2708 The <Result> element contains the following attributes and elements:

2709 ResourceId [Optional]

2710 The identifier of the requested **resource**. If this attribute is omitted, then the **resource**
 2711 identity is specified by the "urn:oasis:names:tc:xacml:1.0:resource:resource-
 2712 id" **resource attribute** in the corresponding <Request> element.

2713 <Decision> [Required]

2714 The **authorization decision**: "Permit", "Deny", "Indeterminate" or "NotApplicable".

2715 <Status> [Optional]

2716 Indicates whether errors occurred during evaluation of the **decision request**, and
 2717 optionally, information about those errors.

2718 <xacml:Obligations> [Optional]

2719 A list of **obligations** that MUST be fulfilled by the **PEP**. If the **PEP** does not understand an
 2720 **obligation**, then it MUST act as if the **PDP** had denied **access** to the requested **resource**.
 2721 See Section 7.11 for a description of how the set of **obligations** to be returned by the PDP
 2722 is determined.

2723 6.11. Element <Decision>

2724 The <Decision> element contains the result of **policy** evaluation.

```

2725 <xs:element name="Decision" type="xacml-context:DecisionType" />
2726 <xs:simpleType name="DecisionType">
2727   <xs:restriction base="xs:string">
2728     <xs:enumeration value="Permit" />
2729     <xs:enumeration value="Deny" />
2730     <xs:enumeration value="Indeterminate" />
2731     <xs:enumeration value="NotApplicable" />
2732   </xs:restriction>
2733 </xs:simpleType>

```

2734 The <Decision> element is of **DecisionType** simple type.

2735 The values of the <Decision> element have the following meanings:

2736 "Permit": the requested **access** is permitted.

2737 "Deny": the requested **access** is denied.

2738 "Indeterminate": the **PDP** is unable to evaluate the requested **access**. Reasons for such
 2739 inability include: missing **attributes**, network errors while retrieving **policies**, division by
 2740 zero during **policy** evaluation, syntax errors in the **decision request** or in the **policy**, etc..

2741 "NotApplicable": the **PDP** does not have any **policy** that applies to this **decision request**.

2742 6.12. Element <Status>

2743 The <Status> element represents the status of the **authorization decision** result.

```

2744 <xs:element name="Status" type="xacml-context:StatusType" />
2745 <xs:complexType name="StatusType">
2746   <xs:sequence>
2747     <xs:element ref="xacml-context:StatusCode" />
2748     <xs:element ref="xacml-context:StatusMessage" minOccurs="0" />
2749     <xs:element ref="xacml-context:StatusDetail" minOccurs="0" />
2750   </xs:sequence>

```


2751 `</xs:complexType>`

2752 The `<Status>` element is of **StatusType** complex type.

2753 The `<Status>` element contains the following elements:

2754 `<StatusCode>` [Required]

2755 Status code.

2756 `<StatusMessage>` [Optional]

2757 A status message describing the status code.

2758 `<StatusDetail>` [Optional]

2759 Additional status information.

2760 **6.13. Element `<StatusCode>`**

2761 The `<StatusCode>` element contains a major status code value and an optional sequence of
2762 minor status codes.

```
2763 <xs:element name="StatusCode" type="xacml-context:StatusCodeType" />
2764 <xs:complexType name="StatusCodeType">
2765 <xs:sequence>
2766 <xs:element ref="xacml-context:StatusCode" minOccurs="0" />
2767 </xs:sequence>
2768 <xs:attribute name="Value" type="xs:anyURI" use="required" />
2769 </xs:complexType>
```

2770 The `<StatusCode>` element is of **StatusCodeType** complex type.

2771 The `<StatusCode>` element contains the following attributes and elements:

2772 Value [Required]

2773 See Section B.7 for a list of values.

2774 `<StatusCode>` [Any Number]

2775 Minor status code. This status code qualifies its parent status code.

2776 **6.14. Element `<StatusMessage>`**

2777 The `<StatusMessage>` element is a free-form description of the status code.

```
2778 <xs:element name="StatusMessage" type="xs:string" />
```

2779 The `<StatusMessage>` element is of **xs:string** type.

2780 **6.15. Element `<StatusDetail>`**

2781 The `<StatusDetail>` element qualifies the `<Status>` element with additional information.

```
2782 <xs:element name="StatusDetail" type="xacml-context:StatusDetailType" />
2783 <xs:complexType name="StatusDetailType">
2784 <xs:sequence>
2785 <xs:any namespace="##any" processContents="lax" minOccurs="0"
2786 maxOccurs="unbounded" />
2787 </xs:sequence>
2788 </xs:complexType>
```

2789 The <StatusDetail> element is of **StatusDetailType** complex type.

2790 The <StatusDetail> element allows arbitrary XML content.

2791 Inclusion of a <StatusDetail> element is optional. However, if a **PDP** returns one of the
2792 following XACML-defined <StatusCode> values and includes a <StatusDetail> element, then
2793 the following rules apply.

2794 urn:oasis:names:tc:xacml:1.0:status:ok

2795 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “ok” status value.

2796 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

2797 A **PDP** MAY choose not to return any <StatusDetail> information or MAY choose to return a
2798 <StatusDetail> element containing one or more <xacml-context:Attribute> elements. If
2799 the **PDP** includes <AttributeValue> elements in the <Attribute> element, then this indicates
2800 the acceptable values for that **attribute**. If no <AttributeValue> elements are included, then
2801 this indicates the names of **attributes** that the **PDP** failed to resolve during its evaluation. The list
2802 of **attributes** may be partial or complete. There is no guarantee by the **PDP** that supplying the
2803 missing values or **attributes** will be sufficient to satisfy the **policy**.

2804 urn:oasis:names:tc:xacml:1.0:status:syntax-error

2805 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “syntax-error” status
2806 value. A syntax error may represent either a problem with the **policy** being used or with the
2807 request **context**. The **PDP** MAY return a <StatusMessage> describing the problem.

2808 urn:oasis:names:tc:xacml:1.0:status:processing-error

2809 A **PDP** MUST NOT return <StatusDetail> element in conjunction with the “processing-error”
2810 status value. This status code indicates an internal problem in the **PDP**. For security reasons, the
2811 **PDP** MAY choose to return no further information to the **PEP**. In the case of a divide-by-zero error
2812 or other computational error, the **PDP** MAY return a <StatusMessage> describing the nature of
2813 the error.

2814 7. Functional requirements (normative)

2815 This section specifies certain functional requirements that are not directly associated with the
2816 production or consumption of a particular XACML element.

2817 7.1. Policy enforcement point

2818 This section describes the requirements for the **PEP**.

2819 An application functions in the role of the **PEP** if it guards access to a set of **resources** and asks
2820 the **PDP** for an **authorization decision**. The **PEP** MUST abide by the **authorization decision** in
2821 the following way:

2822 A **PEP** SHALL allow access to the **resource** only if a valid XACML response of "Permit" is returned
2823 by the **PDP**. The **PEP** SHALL deny access to the **resource** in all other cases. An XACML
2824 response of "Permit" SHALL be considered valid only if the **PEP** understands all of the **obligations**
2825 contained in the response.

2826

7.2. Base policy

2827 A **PDP** SHALL represent one **policy** or **policy set**, called its **base policy**. This base **policy** MAY be
 2828 a <Policy> element containing a <Target> element that matches every possible **decision**
 2829 **request**, or (for instance) it MAY be a <Policy> element containing a <Target> element that
 2830 matches only a specific **subject**. In such cases, the base policy SHALL form the root-node of a
 2831 tree of policies connected by <PolicyIdReference> and <PolicySetIdReference>
 2832 elements to all the **rules** that may be applicable to any **decision request** that the **PDP** is capable
 2833 of evaluating.

2834 In the case of a **PDP** that retrieves **policies** according to the **decision request** that it is processing,
 2835 the base policy SHALL contain a <Policy> element containing a <Target> element that matches
 2836 every possible **decision request** and a PolicyCombiningAlgId attribute with the value "Only-
 2837 one-applicable". In other words, the **PDP** SHALL return an error if it retrieves policies that do not
 2838 form a single tree.

2839

7.3. Target evaluation

2840 The **target** value SHALL be "Match" if the **subject**, **resource** and **action** specified in the **target** all
 2841 match values in the request **context**. The **target** value SHALL be "No-match" if one or more of the
 2842 **subject**, **resource** and **action** specified in the **target** do not match values in the request **context**.
 2843 The value of a <SubjectMatch>, <ResourceMatch> or <ActionMatch> element, in which a
 2844 referenced **attribute** value cannot be obtained, depends on the value of the MustBePresent
 2845 attribute of the <AttributeDesignator>. If the MustBePresent attribute is "True", then the result of
 2846 the <SubjectMatch>, <ResourceMatch> or <ActionMatch> element SHALL be
 2847 "Indeterminate" in this case. If the MustBePresent attribute is "False" or missing, then the result
 2848 of the <SubjectMatch>, <ResourceMatch> or <ActionMatch> element SHALL be "False".

2849

7.4. Condition evaluation

2850 The **condition** value SHALL be "True" if the <Condition> element is absent, or if it evaluates to
 2851 "True" for the **attribute** values supplied in the request **context**. Its value is "False" if the
 2852 <Condition> element evaluates to "False" for the **attribute** values supplied in the request
 2853 **context**. If any **attribute** value referenced in the **condition** cannot be obtained, then the **condition**
 2854 SHALL evaluate to "Indeterminate".

2855

7.5. Rule evaluation

2856 A **rule** has a value that can be calculated by evaluating its contents. **Rule** evaluation involves
 2857 separate evaluation of the **rule's target** and **condition**. The **rule** truth table is shown in Table 1.

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

2858

Table 1 - Rule truth table

2859 If the **target** value is "No-match" or "Indeterminate" then the **rule** value SHALL be "NotApplicable"
2860 or "Indeterminate", respectively, regardless of the value of the **condition**. For these cases,
2861 therefore, the **condition** need not be evaluated in order to determine the **rule** value.

2862 If the **target** value is "Match" and the **condition** value is "True", then the **effect** specified in the **rule**
2863 SHALL determine the **rule** value.

2864 **7.6. Policy evaluation**

2865 The value of a **policy** SHALL be determined only by its contents, considered in relation to the
2866 contents of the **request context**. A **policy's** value SHALL be determined by evaluation of the
2867 **policy's target** and **rules**, according to the specified **rule-combining algorithm**.

2868 The **policy's target** SHALL be evaluated to determine the applicability of the **policy**. If the **target**
2869 evaluates to "Match", then the value of the **policy** SHALL be determined by evaluation of the
2870 **policy's rules**, according to the specified **rule-combining algorithm**. If the **target** evaluates to
2871 "No-Match", then the value of the **policy** SHALL be "NotApplicable". If the **target** evaluates to
2872 "Indeterminate", then the value of the **policy** SHALL be "Indeterminate".

2873 The **policy** truth table is shown in Table 2.

Target	Rule values	Policy Value
"Match"	At least one rule value is its Effect	Specified by the 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"

2874

Table 2 - Policy truth table

2875 A Rules value of "At-least-one-applicable" SHALL be used if the <Rule> element is absent, or if
2876 one or more of the **rules** contained in the **policy** is applicable to the **decision request** (i.e., returns
2877 a value of "Effect"; see Section 7.5). A value of "None-applicable" SHALL be used if no **rule**
2878 contained in the **policy** is applicable to the request and if no **rule** contained in the **policy** returns a
2879 value of "Indeterminate". If no **rule** contained in the **policy** is applicable to the request but one or
2880 more **rule** returns a value of "Indeterminate", then **rules** SHALL evaluate to "Indeterminate".

2881 If the **target** value is "No-match" or "Indeterminate" then the **policy** value SHALL be
2882 "NotApplicable" or "Indeterminate", respectively, regardless of the value of the **rules**. For these
2883 cases, therefore, the **rules** need not be evaluated in order to determine the **policy** value.

2884 If the **target** value is "Match" and the **rules** value is "At-least-one-applicable" or "Indeterminate",
2885 then the **rule-combining algorithm** specified in the **policy** SHALL determine the **policy** value.

2886

7.7. Policy Set evaluation

2887 The value of a **policy set** SHALL be determined by its contents, considered in relation to the
2888 contents of the **request context**. A **policy set's** value SHALL be determined by evaluation of the
2889 **policy set's target, policies** and **policy sets**, according to the specified **policy-combining**
2890 **algorithm**.

2891 The **policy set's target** SHALL be evaluated to determine the applicability of the **policy set**. If the
2892 **target** evaluates to "Match" then the value of the **policy set** SHALL be determined by evaluation of
2893 the **policy set's policies** and **policy sets**, according to the specified **policy-combining algorithm**.
2894 If the **target** evaluates to "Not-Match", then the value of the **policy set** shall be "NotApplicable". If
2895 the **target** evaluates to "Indeterminate", then the value of the **policy set** SHALL be "Indeterminate".

2896 The **policy set** truth table is shown in Table 3.

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"

2897

Table 3 – Policy set truth table

2898 A **policies** value of "At-least-one-applicable" SHALL be used if there are no contained or
2899 referenced **policies** or **policy sets**, or if one or more of the **policies** or **policy sets** contained in or
2900 referenced by the **policy set** is applicable to the **decision request** (i.e., returns a value determined
2901 by its **rule-combining algorithm**; see Section 7.6). A value of "None-applicable" SHALL be used if
2902 no **policy** or **policy set** contained in or referenced by the **policy set** is applicable to the request
2903 and if no **policy** or **policy set** contained in or referenced by the **policy set** returns a value of
2904 "Indeterminate". If no **policy** or **policy set** contained in or referenced by the **policy set** is
2905 applicable to the request but one or more **policy** or **policy set** returns a value of "Indeterminate",
2906 then **policies** SHALL evaluate to "Indeterminate".

2907 If the **target** value is "No-match" or "Indeterminate" then the **policy set** value SHALL be
2908 "NotApplicable" or "Indeterminate", respectively, regardless of the value of the **policies**. For these
2909 cases, therefore, the **policies** need not be evaluated in order to determine the **policy set** value.

2910 If the **target** value is "Match" and the **policies** value is "At-least-one-applicable" or "Indeterminate",
2911 then the **policy-combining algorithm** specified in the **policy set** SHALL determine the **policy set**
2912 value.

2913 7.8. Hierarchical resources

2914 It is often the case that a **resource** is organized as a hierarchy (e.g. file system, XML document).
2915 Some access requesters may request **access** to an entire subtree of a **resource** specified by a
2916 node. XACML allows the **PEP** (or **context handler**) to specify whether the **decision request** is

2917 just for a single **resource** or for a subtree below the specified **resource**. The latter is equivalent to
2918 repeating a single request for each node in the entire subtree. When a request **context** contains a
2919 resource attribute of type

2920 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2921 with a value of "Immediate", or if it does not contain that **attribute**, then the **decision request**
2922 SHALL be interpreted to apply to just the single **resource** specified by the
2923 "urn:oasis:names:tc:xacml:1.0:resource:resource-id" **attribute**.

2924 When the

2925 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2926 **attribute** has the value "Children", the **decision request** SHALL be interpreted to apply to the
2927 specified **resource** and its immediate children **resources**.

2928 When the

2929 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2930 **attribute** has the value "Descendants", the **decision request** SHALL be interpreted to apply to
2931 both the specified **resource** and all its descendant **resources**.

2932 In the case of "Children" and "Descendants", the **authorization decision** MAY include multiple
2933 results for the multiple sub-nodes in the **resource** sub-tree.

2934 An XACML **authorization response** MAY contain multiple <Result> elements.

2935 Note that the method by which the **PDP** discovers whether the **resource** is hierarchically organized
2936 or not is outside the scope of XACML.

2937 In the case where a child or descendant **resource** cannot be accessed, the <Result> element
2938 associated with the parent element SHALL contain a <StatusCode> Value of
2939 "urn:oasis:names:tc:xacml:1.0:status:processing-error".

2940 **7.9. Attributes**

2941 **Attributes** are specified in the request **context**, regardless of whether or not they appeared in the
2942 original **decision request**, and are referred to in the **policy** by **subject**, **resource**, **action** and
2943 **environment attribute** designators and **attribute** selectors. A **named attribute** is the term used for
2944 the criteria that the specific **subject**, **resource**, **action** and **environment attribute** designators and
2945 selectors use to refer to **attributes** in the **subject**, **resource**, **action** and **environment** elements of
2946 the request **context**, respectively.

2947 **7.9.1. Attribute Matching**

2948 A **named attribute** has specific criteria with which to match **attributes** in the **context**. An **attribute**
2949 specifies **AttributeId**, **DataType** and **Issuer** attributes, and each **named attribute** also
2950 specifies **AttributeId**, **DataType** and optional **Issuer** attributes. A **named attribute** SHALL
2951 match an **attribute** if the values of their respective **AttributeId**, **DataType** and optional **Issuer**
2952 attributes match within their particular element, e.g. **subject**, **resource**, **action** or **environment**, of
2953 the **context**. The **AttributeId** of the named **attribute** MUST match, by URI equality, the
2954 **AttributeId** of the context **attribute**. The **DataType** of the named **attribute** MUST match, by
2955 URI equality, the **DataType** of the same context **attribute**. If **Issuer** is supplied in the named
2956 **attribute**, then it MUST match, by URI equality, the **Issuer** of the same context **attribute**. If
2957 **Issuer** is not supplied in the **named attribute**, then the matching of the context **attribute** to the

2958 *named attribute* SHALL be governed by `AttributeId` and `DataType` alone, regardless of the
2959 presence, absence, or actual value of `Issuer`. In the case of an *attribute* selector, the matching
2960 of the *attribute* to the *named attribute* SHALL be governed by the XPath expression, `DataType`
2961 and `Issuer`.

2962 7.9.2. Attribute Retrieval

2963 The *PDP* SHALL request the values of *attributes* in the request *context* from the *context handler*.
2964 The *PDP* SHALL reference the *attributes* as if they were in a physical request *context* document,
2965 but the *context handler* is responsible for obtaining and supplying the requested values. The
2966 *context handler* SHALL return the values of *attributes* that match the *attribute* designator or
2967 *attribute* selector and form them into a *bag* of values with the specified data-type. If no *attributes*
2968 from the request *context* match, then the *attribute* SHALL be considered missing. If the *attribute*
2969 is missing, then `MustBePresent` governs whether the *attribute* designator or *attribute* selector
2970 returns an empty *bag* or an "Indeterminate" result. If `MustBePresent` is "False" (default value),
2971 then a missing *attribute* SHALL result in an empty *bag*. If `MustBePresent` is "True", then a
2972 missing *attribute* SHALL result in "Indeterminate". This "Indeterminate" result SHALL be handled
2973 in accordance with the specification of the encompassing expressions, *rules*, *policies* and *policy*
2974 *sets*. If the result is "Indeterminate", then the `AttributeId`, `DataType` and `Issuer` of the
2975 *attribute* MAY be listed in the *authorization decision* as described in Section 7.10. However, a
2976 *PDP* MAY choose not to return such information for security reasons.

2977 7.9.3. Environment Attributes

2978 *Environment attributes* are listed in Section B.8. If a value for one of these *attributes* is supplied
2979 in the *decision request*, then the *context handler* SHALL use that value. Otherwise, the *context*
2980 *handler* SHALL supply a value. For the date and time *attributes*, the supplied value SHALL have
2981 the semantics of "date and time that apply to the *decision request*".

2982 7.10. Authorization decision

2983 Given a valid XACML *policy* or *policy set*, a compliant XACML *PDP* MUST evaluate the *policy* as
2984 specified in Sections 5, 0 and 4.2. The *PDP* MUST return a response *context*, with one
2985 `<Decision>` element of value "Permit", "Deny", "Indeterminate" or "NotApplicable".

2986 If the *PDP* cannot make a decision, then an "Indeterminate" `<Decision>` element contents SHALL
2987 be returned. The *PDP* MAY return a `<Decision>` element contents of "Indeterminate" with a
2988 status code of:

2989 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

2990 signifying that more information is needed. In this case, the `<Status>` element MAY list the
2991 names and data-types of any *attributes* of the *subjects* and the *resource* that are needed by the
2992 *PDP* to refine its decision. A *PEP* MAY resubmit a refined request *context* in response to a
2993 `<Decision>` element contents of "Indeterminate" with a status code of

2994 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

2995 by adding *attribute* values for the *attribute* names that were listed in the previous response. When
2996 the *PDP* returns a `<Decision>` element contents of "Indeterminate", with a status code of

2997 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

2998 it MUST NOT list the names and data-types of any *attribute* of the *subject* or the *resource* for
2999 which values were supplied in the original request. Note, this requirement forces the *PDP* to

3000 eventually return an **authorization decision** of "Permit", "Deny" or "Indeterminate" with some other
3001 status code, in response to successively-refined requests.

3002 7.11. Obligations

3003 A **policy** or **policy set** may contain one or more **obligations**. When such a **policy** or **policy set** is
3004 evaluated, an **obligation** SHALL be passed up to the next level of evaluation (the enclosing or
3005 referencing **policy set** or **authorization decision**) only if the **effect** of the **policy** or **policy set**
3006 being evaluated matches the value of the `xacml:FulfillOn` attribute of the **obligation**.

3007
3008 As a consequence of this procedure, no **obligations** SHALL be returned to the **PEP** if the **policies**
3009 or **policy sets** from which they are drawn are not evaluated, or if their evaluated result is
3010 "Indeterminate" or "NotApplicable", or if the **decision** resulting from evaluating the **policy** or **policy**
3011 **set** does not match the **decision** resulting from evaluating an enclosing **policy set**.

3012
3013 If the **PDP's** evaluation is viewed as a tree of **policy sets** and **policies**, each of which returns
3014 "Permit" or "Deny", then the set of **obligations** returned by the **PDP** to the **PEP** will include only the
3015 **obligations** associated with those paths where the **effect** at each level of evaluation is the same as
3016 the **effect** being returned by the **PDP**.

3017 A **PEP** that receives a valid XACML response of "Permit" with **obligations** SHALL be responsible
3018 for fulfilling *all* of those **obligations**. A **PEP** that receives an XACML response of "Deny" with
3019 **obligations** SHALL be responsible for fulfilling all of the **obligations** that it *understands*.

3020 7.12. Unsupported functionality

3021 If the **PDP** attempts to evaluate a **policy set** or **policy** that contains an optional element type or
3022 feature that the **PDP** does not support, then the **PDP** SHALL return a `<Decision>` value of
3023 "Indeterminate". If a `<StatusCode>` element is also returned, then its value SHALL be
3024 "urn:oasis:names:tc:xacml:1.0:status:syntax-error" in the case of an unsupported element type, and
3025 "urn:oasis:names:tc:xacml:1.0:status:processing-error" in the case of an unsupported feature.

3026 7.13. Syntax and type errors

3027 If a **policy** that contains invalid syntax is evaluated by the XACML **PDP** at the time a **decision**
3028 **request** is received, then the result of that **policy** SHALL be "Indeterminate" with a `StatusCode`
3029 value of "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

3030 If a **policy** that contains invalid static data-types is evaluated by the XACML **PDP** at the time a
3031 **decision request** is received, then the result of that **policy** SHALL be "Indeterminate" with a
3032 `StatusCode` value of "urn:oasis:names:tc:xacml:1.0:status:processing-error".

3033 8. XACML extensibility points (non-normative)

3034 This section describes the points within the XACML model and schema where extensions can be
3035 added

3036 8.1. Extensible XML attribute types

3037 The following XML attributes have values that are URIs or QNames. These may be extended by
3038 the creation of new URIs or QNames associated with new semantics for these attributes.

3039 AttributeId,
3040 AttributeValue,
3041 DataType,
3042 FunctionId,
3043 MatchId,
3044 ObligationId,
3045 PolicyCombiningAlgId,
3046 RuleCombiningAlgId,
3047 StatusCode,
3048 SubjectCategory.
3049 See Section 5 for definitions of these attribute types.

3050 **8.2. Structured attributes**

3051 An XACML <AttributeValue> element MAY contain an instance of a structured XML data-type.
3052 Section A.3 describes a number of standard techniques to identify data items within such a
3053 structured attribute. Listed here are some additional techniques that require XACML extensions.

- 3054 1. For a given structured data-type, a community of XACML users MAY define new attribute
3055 identifiers for each leaf sub-element of the structured data-type that has a type conformant
3056 with one of the XACML-defined primitive data-types. Using these new attribute identifiers,
3057 the **PEPs** or **context handlers** used by that community of users can flatten instances of
3058 the structured data-type into a sequence of individual <Attribute> elements. Each such
3059 <Attribute> element can be compared using the XACML-defined functions. Using this
3060 method, the structured data-type itself never appears in an <AttributeValue> element.
- 3061 2. A community of XACML users MAY define a new function that can be used to compare a
3062 value of the structured data-type against some other value. This method may only be used
3063 by **PDPs** that support the new function.

3064 **9. Security and privacy considerations (non-** 3065 **normative)**

3066 This section identifies possible security and privacy compromise scenarios that should be
3067 considered when implementing an XACML-based system. The section is informative only. It is left
3068 to the implementer to decide whether these compromise scenarios are practical in their
3069 environment and to select appropriate safeguards.

3070 **9.1. Threat model**

3071 We assume here that the adversary has access to the communication channel between the
3072 XACML actors and is able to interpret, insert, delete and modify messages or parts of messages.

3073 Additionally, an actor may use information from a former transaction maliciously in subsequent
3074 transactions. It is further assumed that **rules** and **policies** are only as reliable as the actors that

3075 create and use them. Thus it is incumbent on each actor to establish appropriate trust in the other
3076 actors upon which it relies. Mechanisms for trust establishment are outside the scope of this
3077 specification.

3078 The messages that are transmitted between the actors in the XACML model are susceptible to
3079 attack by malicious third parties. Other points of vulnerability include the *PEP*, the *PDP* and the
3080 *PAP*. While some of these entities are not strictly within the scope of this specification, their
3081 compromise could lead to the compromise of **access control** enforced by the *PEP*.

3082 It should be noted that there are other components of a distributed system that may be
3083 compromised, such as an operating system and the domain-name system (DNS) that are outside
3084 the scope of this discussion of threat models. Compromise in these components may also lead to a
3085 policy violation.

3086 The following sections detail specific compromise scenarios that may be relevant to an XACML
3087 system.

3088 **9.1.1. Unauthorized disclosure**

3089 XACML does not specify any inherent mechanisms for confidentiality of the messages exchanged
3090 between actors. Therefore, an adversary could observe the messages in transit. Under certain
3091 security policies, disclosure of this information is a violation. Disclosure of **attributes** or the types
3092 of **decision requests** that a **subject** submits may be a breach of privacy policy. In the commercial
3093 sector, the consequences of unauthorized disclosure of personal data may range from
3094 embarrassment to the custodian to imprisonment and large fines in the case of medical or financial
3095 data.

3096 Unauthorized disclosure is addressed by confidentiality mechanisms.

3097 **9.1.2. Message replay**

3098 A message replay attack is one in which the adversary records and replays legitimate messages
3099 between XACML actors. This attack may lead to denial of service, the use of out-of-date
3100 information or impersonation.

3101 Prevention of replay attacks requires the use of message freshness mechanisms.

3102 Note that encryption of the message does not mitigate a replay attack since the message is just
3103 replayed and does not have to be understood by the adversary.

3104 **9.1.3. Message insertion**

3105 A message insertion attack is one in which the adversary inserts messages in the sequence of
3106 messages between XACML actors.

3107 The solution to a message insertion attack is to use mutual authentication and a message
3108 sequence integrity mechanism between the actors. It should be noted that just using SSL mutual
3109 authentication is not sufficient. This only proves that the other party is the one identified by the
3110 subject of the X.509 certificate. In order to be effective, it is necessary to confirm that the certificate
3111 subject is authorized to send the message.

3112 **9.1.4. Message deletion**

3113 A message deletion attack is one in which the adversary deletes messages in the sequence of
3114 messages between XACML actors. Message deletion may lead to denial of service. However, a

3115 properly designed XACML system should not render an incorrect authorization decision as a result
3116 of a message deletion attack.

3117 The solution to a message deletion attack is to use a message integrity mechanism between the
3118 actors.

3119 **9.1.5. Message modification**

3120 If an adversary can intercept a message and change its contents, then they may be able to alter an
3121 **authorization decision**. Message integrity mechanisms can prevent a successful message
3122 modification attack.

3123 **9.1.6. NotApplicable results**

3124 A result of "NotApplicable" means that the **PDP** did not have a policy whose target matched the
3125 information in the **decision request**. In general, we highly recommend using a "default-deny"
3126 policy, so that when a **PDP** would have returned "NotApplicable", a result of "Deny" is returned
3127 instead.

3128 In some security models, however, such as is common in many Web Servers, a result of
3129 "NotApplicable" is treated as equivalent to "Permit". There are particular security considerations
3130 that must be taken into account for this to be safe. These are explained in the following
3131 paragraphs.

3132 If "NotApplicable" is to be treated as "Permit", it is vital that the matching algorithms used by the
3133 policy to match elements in the decision request are closely aligned with the data syntax used by
3134 the applications that will be submitting the decision request. A failure to match will be treated as
3135 "Permit", so an unintended failure to match may allow unintended access.

3136 A common example of this is a Web Server. Commercial http responders allow a variety of
3137 syntaxes to be treated equivalently. The "%" can be used to represent characters by hex value.
3138 The URL path "../" provides multiple ways of specifying the same value. Multiple character sets
3139 may be permitted and, in some cases, the same printed character can be represented by different
3140 binary values. Unless the matching algorithm used by the policy is sophisticated enough to catch
3141 these variations, unintended access may be permitted.

3142 It is safe to treat "NotApplicable" as "Permit" only in a closed environment where all applications
3143 that formulate a decision request can be guaranteed to use the exact syntax expected by the
3144 policies used by the **PDP**. In a more open environment, where decision requests may be received
3145 from applications that may use any legal syntax, it is strongly recommended that "NotApplicable"
3146 NOT be treated as "Permit" unless matching rules have been very carefully designed to match all
3147 possible applicable inputs, regardless of syntax or type variations.

3148 **9.1.7. Negative rules**

3149 A negative **rule** is one that is based on a **predicate** not being "True". If not used with care,
3150 negative **rules** can lead to policy violation, therefore some authorities recommend that they not be
3151 used. However, negative **rules** can be extremely efficient in certain cases, so XACML has chosen
3152 to include them. Nevertheless, it is recommended that they be used with care and avoided if
3153 possible.

3154 A common use for negative **rules** is to deny **access** to an individual or subgroup when their
3155 membership in a larger group would otherwise permit them access. For example, we might want to
3156 write a **rule** that allows all Vice Presidents to see the unpublished financial data, except for Joe,
3157 who is only a Ceremonial Vice President and can be indiscreet in his communications. If we have
3158 complete control of the administration of **subject attributes**, a superior approach would be to
3159 define "Vice President" and "Ceremonial Vice President" as distinct groups and then define **rules**

3160 accordingly. However, in some environments this approach may not be feasible. (It is worth noting
3161 in passing that, generally speaking, referring to individuals in **rules** does not scale well. Generally,
3162 shared **attributes** are preferred.)

3163 If not used with care, negative **rules** can lead to policy violation in two common cases. They are:
3164 when **attributes** are suppressed and when the base group changes. An example of suppressed
3165 **attributes** would be if we have a policy that **access** should be permitted, *unless* the **subject** is a
3166 credit risk. If it is possible that the **attribute** of being a credit risk may be unknown to the **PDP** for
3167 some reason, then unauthorized **access** may be permitted. In some environments, the **subject**
3168 may be able to suppress the publication of **attributes** by the application of privacy controls, or the
3169 server or repository that contains the information may be unavailable for accidental or intentional
3170 reasons.

3171 An example of a changing base group would be if there is a policy that everyone in the engineering
3172 department may change software source code, except for secretaries. Suppose now that the
3173 department was to merge with another engineering department and the intent is to maintain the
3174 same policy. However, the new department also includes individuals identified as administrative
3175 assistants, who ought to be treated in the same way as secretaries. Unless the policy is altered,
3176 they will unintentionally be permitted to change software source code. Problems of this type are
3177 easy to avoid when one individual administers all **policies**, but when administration is distributed,
3178 as XACML allows, this type of situation must be explicitly guarded against.

3179 **9.2. Safeguards**

3180 **9.2.1. Authentication**

3181 Authentication provides the means for one party in a transaction to determine the identity of the
3182 other party in the transaction. Authentication may be in one direction, or it may be bilateral.

3183 Given the sensitive nature of **access control** systems, it is important for a **PEP** to authenticate the
3184 identity of the **PDP** to which it sends **decision requests**. Otherwise, there is a risk that an
3185 adversary could provide false or invalid **authorization decisions**, leading to a policy violation.

3186 It is equally important for a **PDP** to authenticate the identity of the **PEP** and assess the level of trust
3187 to determine what, if any, sensitive data should be passed. One should keep in mind that even
3188 simple "Permit" or "Deny" responses could be exploited if an adversary were allowed to make
3189 unlimited requests to a **PDP**.

3190 Many different techniques may be used to provide authentication, such as co-located code, a
3191 private network, a VPN or digital signatures. Authentication may also be performed as part of the
3192 communication protocol used to exchange the **contexts**. In this case, authentication may be
3193 performed at the message level or at the session level.

3194 **9.2.2. Policy administration**

3195 If the contents of **policies** are exposed outside of the **access control** system, potential **subjects**
3196 may use this information to determine how to gain unauthorized **access**.

3197 To prevent this threat, the repository used for the storage of **policies** may itself require **access**
3198 **control**. In addition, the <Status> element should be used to return values of missing **attributes**
3199 only when exposure of the identities of those **attributes** will not compromise security.

3200

9.2.3. Confidentiality

3201 Confidentiality mechanisms ensure that the contents of a message can be read only by the desired
3202 recipients and not by anyone else who encounters the message while it is in transit. There are two
3203 areas in which confidentiality should be considered: one is confidentiality during transmission; the
3204 other is confidentiality within a `<Policy>` element.

3205

9.2.3.1. Communication confidentiality

3206 In some environments it is deemed good practice to treat all data within an **access control** system
3207 as confidential. In other environments, **policies** may be made freely available for distribution,
3208 inspection and audit. The idea behind keeping **policy** information secret is to make it more difficult
3209 for an adversary to know what steps might be sufficient to obtain unauthorized **access**. Regardless
3210 of the approach chosen, the security of the **access control** system should not depend on the
3211 secrecy of the **policy**.

3212 Any security concerns or requirements related to transmitting or exchanging XACML `<policy>`
3213 elements are outside the scope of the XACML standard. While it is often important to ensure that
3214 the integrity and confidentiality of `<policy>` elements is maintained when they are exchanged
3215 between two parties, it is left to the implementers to determine the appropriate mechanisms for their
3216 environment.

3217 Communications confidentiality can be provided by a confidentiality mechanism, such as SSL.
3218 Using a point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points
3219 is compromised.

3220

9.2.3.2. Statement level confidentiality

3221 In some cases, an implementation may want to encrypt only parts of an XACML `<Policy>`
3222 element.

3223 The XML Encryption Syntax and Processing Candidate Recommendation from W3C can be used
3224 to encrypt all or parts of an XML document. This specification is recommended for use with
3225 XACML.

3226 It should go without saying that if a repository is used to facilitate the communication of cleartext
3227 (i.e., unencrypted) **policy** between the **PAP** and **PDP**, then a secure repository should be used to
3228 store this sensitive data.

3229

9.2.4. Policy integrity

3230 The XACML **policy**, used by the **PDP** to evaluate the request **context**, is the heart of the system.
3231 Therefore, maintaining its integrity is essential. There are two aspects to maintaining the integrity of
3232 the **policy**. One is to ensure that `<Policy>` elements have not been altered since they were
3233 originally created by the **PAP**. The other is to ensure that `<Policy>` elements have not been
3234 inserted or deleted from the set of **policies**.

3235 In many cases, both aspects can be achieved by ensuring the integrity of the actors and
3236 implementing session-level mechanisms to secure the communication between actors. The
3237 selection of the appropriate mechanisms is left to the implementers. However, when **policy** is
3238 distributed between organizations to be acted on at a later time, or when the **policy** travels with the
3239 protected resource, it would be useful to sign the **policy**. In these cases, the XML Signature
3240 Syntax and Processing standard from W3C is recommended to be used with XACML.

3241 Digital signatures should only be used to ensure the integrity of the statements. Digital signatures
3242 should not be used as a method of selecting or evaluating **policy**. That is, the **PDP** should not

3243 request a **policy** based on who signed it or whether or not it has been signed (as such a basis for
3244 selection would, itself, be a matter of policy). However, the **PDP** must verify that the key used to
3245 sign the **policy** is one controlled by the purported issuer of the **policy**. The means to do this are
3246 dependent on the specific signature technology chosen and are outside the scope of this document.

3247 **9.2.5. Policy identifiers**

3248 Since **policies** can be referenced by their identifiers, it is the responsibility of the **PAP** to ensure
3249 that these are unique. Confusion between identifiers could lead to misidentification of the
3250 **applicable policy**. This specification is silent on whether a **PAP** must generate a new identifier
3251 when a **policy** is modified or may use the same identifier in the modified **policy**. This is a matter of
3252 administrative practice. However, care must be taken in either case. If the identifier is reused,
3253 there is a danger that other **policies** or **policy sets** that reference it may be adversely affected.
3254 Conversely, if a new identifier is used, these other **policies** may continue to use the prior **policy**,
3255 unless it is deleted. In either case the results may not be what the **policy** administrator intends.

3256 **9.2.6. Trust model**

3257 Discussions of authentication, integrity and confidentiality mechanisms necessarily assume an
3258 underlying trust model: how can one actor come to believe that a given key is uniquely associated
3259 with a specific, identified actor so that the key can be used to encrypt data for that actor or verify
3260 signatures (or other integrity structures) from that actor? Many different types of trust model exist,
3261 including strict hierarchies, distributed authorities, the Web, the bridge and so on.

3262 It is worth considering the relationships between the various actors of the **access control** system in
3263 terms of the interdependencies that do and do not exist.

- 3264 • None of the entities of the authorization system are dependent on the **PEP**. They may
3265 collect data from it, for example authentication, but are responsible for verifying it.
- 3266 • The correct operation of the system depends on the ability of the **PEP** to actually enforce
3267 **policy** decisions.
- 3268 • The **PEP** depends on the **PDP** to correctly evaluate **policies**. This in turn implies that the
3269 **PDP** is supplied with the correct inputs. Other than that, the **PDP** does not depend on the
3270 **PEP**.
- 3271 • The **PDP** depends on the **PAP** to supply appropriate policies. The **PAP** is not dependent
3272 on other components.

3273 **9.2.7. Privacy**

3274 It is important to be aware that any transactions that occur with respect to **access control** may
3275 reveal private information about the actors. For example, if an XACML **policy** states that certain
3276 data may only be read by **subjects** with “Gold Card Member” status, then any transaction in which
3277 a **subject** is permitted **access** to that data leaks information to an adversary about the **subject's**
3278 status. Privacy considerations may therefore lead to encryption and/or to **access control policies**
3279 surrounding the enforcement of XACML **policy** instances themselves: confidentiality-protected
3280 channels for the request/response protocol messages, protection of **subject attributes** in storage
3281 and in transit, and so on.

3282 Selection and use of privacy mechanisms appropriate to a given environment are outside the scope
3283 of XACML. The decision regarding whether, how and when to deploy such mechanisms is left to
3284 the implementers associated with the environment.

3285

10. Conformance (normative)

3286

10.1. Introduction

3287

The XACML specification addresses the following aspect of conformance:

3288

1.The XACML specification defines a number of functions, etc. that have somewhat specialist

3289

application, therefore they are not required to be implemented in an implementation that claims to

3290

conform with the OASIS standard.

3291

10.2.Conformance tables

3292

This section lists those portions of the specification that **MUST** be included in an implementation of

3293

a **PDP** that claims to conform with XACML v1.0. A set of test cases has been created to assist in

3294

this process. These test cases are hosted by Sun Microsystems and can be located from the

3295

XACML Web page. The site hosting the test cases contains a full description of the test cases and

3296

how to execute them.

3297

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

3298

10.2.1. Schema elements

3299

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

Element name	M/O
xacml-context:Action	M
xacml-context:Attribute	M
xacml-context:AttributeValue	M
xacml-context:Decision	M
xacml-context:Environment	M
xacml-context:Obligations	O
xacml-context:Request	M
xacml-context:Resource	M
xacml-context:ResourceContent	O
xacml-context:Response	M
xacml-context:Result	M
xacml-context:Status	O
xacml-context:StatusCode	O
xacml-context:StatusDetail	O
xacml-context:StatusMessage	O
xacml-context:Subject	M
xacml:Action	M
xacml:ActionAttributeDesignator	M
xacml:ActionMatch	M
xacml:Actions	M
xacml:AnyAction	M
xacml:AnyResource	M
xacml:AnySubject	M
xacml:Apply	M
xacml:AttributeAssignment	O
xacml:AttributeSelector	O
xacml:AttributeValue	M
xacml:Condition	M
xacml:Description	M

xacml:EnvironmentAttributeDesignator	M
xacml:Function	M
xacml:Obligation	O
xacml:Obligations	O
xacml:Policy	M
xacml:PolicyDefaults	O
xacml:PolicyIdReference	M
xacml:PolicySet	M
xacml:PolicySetDefaults	O
xacml:PolicySetIdReference	M
xacml:Resource	M
xacml:ResourceAttributeDesignator	M
xacml:ResourceMatch	M
xacml:Resources	M
xacml:Rule	M
xacml:Subject	M
xacml:SubjectMatch	M
xacml:Subjects	M
xacml:Target	M
xacml:XPathVersion	O

3300 **10.2.2. Identifier Prefixes**

3301 The following identifier prefixes are reserved by XACML.

Identifier
urn:oasis:names:tc:xacml:1.0
urn:oasis:names:tc:xacml:1.0:conformance-test
urn:oasis:names:tc:xacml:1.0:context
urn:oasis:names:tc:xacml:1.0:example
urn:oasis:names:tc:xacml:1.0:function
urn:oasis:names:tc:xacml:1.0:policy
urn:oasis:names:tc:xacml:1.0:subject
urn:oasis:names:tc:xacml:1.0:resource
urn:oasis:names:tc:xacml:1.0:action

3302 **10.2.3. Algorithms**

3303 The implementation MUST include the rule- and policy-combining algorithms associated with the
3304 following identifiers that are marked "M".

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

3305 **10.2.4. Status Codes**

3306 Implementation support for the urn:oasis:names:tc:xacml:1.0:context:status element is optional, but
 3307 if the element is supported, then the following status codes must be supported and must be used in
 3308 the way XACML has specified.

Identifier	M/O
urn:oasis:names:tc:xacml:1.0:status:missing-attribute	M
urn:oasis:names:tc:xacml:1.0:status:ok	M
urn:oasis:names:tc:xacml:1.0:status:processing-error	M
urn:oasis:names:tc:xacml:1.0:status:syntax-error	M

3309 **10.2.5. Attributes**

3310 The implementation **MUST** support the attributes associated with the following attribute identifiers
 3311 as specified by XACML. If values for these **attributes** are not present in the **decision request**,
 3312 then their values **MUST** be supplied by the **PDP**. So, unlike most other **attributes**, their semantics
 3313 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

3314 **10.2.6. Identifiers**

3315 The implementation **MUST** use the attributes associated with the following identifiers in the way
 3316 XACML has defined. This requirement pertains primarily to implementations of a **PAP** or **PEP** that
 3317 use XACML, since the semantics of the attributes are transparent to the **PDP**.

Identifier	M/O
urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name	O
urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address	O
urn:oasis:names:tc:xacml:1.0:subject:authentication-method	O
urn:oasis:names:tc:xacml:1.0:subject:authentication-time	O
urn:oasis:names:tc:xacml:1.0:subject:key-info	O
urn:oasis:names:tc:xacml:1.0:subject:request-time	O
urn:oasis:names:tc:xacml:1.0:subject:session-start-time	O
urn:oasis:names:tc:xacml:1.0:subject:subject-id	O
urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier	O
urn:oasis:names:tc:xacml:1.0:subject-category:access-subject	M
urn:oasis:names:tc:xacml:1.0:subject-category:codebase	O
urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject	O
urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject	O
urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine	O
urn:oasis:names:tc:xacml:1.0:resource:resource-location	O
urn:oasis:names:tc:xacml:1.0:resource:resource-id	O
urn:oasis:names:tc:xacml:1.0:resource:scope	O
urn:oasis:names:tc:xacml:1.0:resource:simple-file-name	O
urn:oasis:names:tc:xacml:1.0:action:action-id	M
urn:oasis:names:tc:xacml:1.0:action:implied-action	M

3318 **10.2.7. Data-types**

3319 The implementation **MUST** support the data-types associated with the following identifiers marked
 3320 "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/xquery-operators#dayTimeDuration	M
http://www.w3.org/TR/xquery-operators#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

3321 **10.2.8. Functions**

3322 The implementation MUST properly process those functions associated with the identifiers marked
3323 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	
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-equal	
urn:oasis:names:tc:xacml:1.0:function:anyURI-equal	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-equal	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal	M
urn:oasis:names:tc:xacml:1.0:function:integer-add	M
urn:oasis:names:tc:xacml:1.0:function:double-add	M
urn:oasis:names:tc:xacml:1.0:function:integer-subtract	M
urn:oasis:names:tc:xacml:1.0:function:double-subtract	M
urn:oasis:names:tc:xacml:1.0:function:integer-multiply	M
urn:oasis:names:tc:xacml:1.0:function:double-multiply	M
urn:oasis:names:tc:xacml:1.0:function:integer-divide	M
urn:oasis:names:tc:xacml:1.0:function:double-divide	M
urn:oasis:names:tc:xacml:1.0:function:integer-mod	M
urn:oasis:names:tc:xacml:1.0:function:integer-abs	M
urn:oasis:names:tc:xacml:1.0:function:double-abs	M
urn:oasis:names:tc:xacml:1.0:function:round	M
urn:oasis:names:tc:xacml:1.0:function:floor	M
urn:oasis:names:tc:xacml:1.0:function:string-normalize-space	M
urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case	M
urn:oasis:names:tc:xacml:1.0:function:double-to-integer	M
urn:oasis:names:tc:xacml:1.0:function:integer-to-double	M
urn:oasis:names:tc:xacml:1.0:function:or	M
urn:oasis:names:tc:xacml:1.0:function:and	M
urn:oasis:names:tc:xacml:1.0:function:n-of	M
urn:oasis:names:tc:xacml:1.0:function:not	M

urn:oasis:names:tc:xacml:1.0:function:present	M
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:integer-less-than	M
urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:double-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:double-less-than	M
urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-dayTimeDuration	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:date-subtract-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:string-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:string-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:string-less-than	M
urn:oasis:names:tc:xacml:1.0:function:string-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:time-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:time-less-than	M
urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:date-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:date-less-than	M
urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:string-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:string-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:string-is-in	M
urn:oasis:names:tc:xacml:1.0:function:string-bag	M
urn:oasis:names:tc:xacml:1.0:function:boolean-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:boolean-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:boolean-is-in	M
urn:oasis:names:tc:xacml:1.0:function:boolean-bag	M
urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:integer-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:integer-is-in	M
urn:oasis:names:tc:xacml:1.0:function:integer-bag	M
urn:oasis:names:tc:xacml:1.0:function:double-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:double-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:double-is-in	M
urn:oasis:names:tc:xacml:1.0:function:double-bag	M
urn:oasis:names:tc:xacml:1.0:function:time-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:time-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:time-is-in	M
urn:oasis:names:tc:xacml:1.0:function:time-bag	M
urn:oasis:names:tc:xacml:1.0:function:date-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:date-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:date-is-in	M
urn:oasis:names:tc:xacml:1.0:function:date-bag	M

urn:oasis:names:tc:xacml:1.0:function:dateTime-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-is-in	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-bag	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-is-in	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-bag	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-is-in	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-bag	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-is-in	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-bag	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-is-in	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-bag	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-is-in	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-bag	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-is-in	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-bag	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-is-in	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-bag	M
urn:oasis:names:tc:xacml:1.0:function:any-of	M
urn:oasis:names:tc:xacml:1.0:function:all-of	M
urn:oasis:names:tc:xacml:1.0:function:any-of-any	M
urn:oasis:names:tc:xacml:1.0:function:all-of-any	M
urn:oasis:names:tc:xacml:1.0:function:any-of-all	M
urn:oasis:names:tc:xacml:1.0:function:all-of-all	M
urn:oasis:names:tc:xacml:1.0:function:map	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-match	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match	M
urn:oasis:names:tc:xacml:1.0:function:regexp-string-match	M
urn:oasis:names:tc:xacml:1.0:function:xpath-node-count	O
urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal	O
urn:oasis:names:tc:xacml:1.0:function:xpath-node-match	O
urn:oasis:names:tc:xacml:1.0:function:string-intersection	M
urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:string-union	M
urn:oasis:names:tc:xacml:1.0:function:string-subset	M
urn:oasis:names:tc:xacml:1.0:function:string-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:boolean-intersection	M
urn:oasis:names:tc:xacml:1.0:function:boolean-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:boolean-union	M
urn:oasis:names:tc:xacml:1.0:function:boolean-subset	M
urn:oasis:names:tc:xacml:1.0:function:boolean-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:integer-intersection	M
urn:oasis:names:tc:xacml:1.0:function:integer-at-least-one-member-of	M

urn:oasis:names:tc:xacml:1.0:function:integer-union	M
urn:oasis:names:tc:xacml:1.0:function:integer-subset	M
urn:oasis:names:tc:xacml:1.0:function:integer-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:double-intersection	M
urn:oasis:names:tc:xacml:1.0:function:double-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:double-union	M
urn:oasis:names:tc:xacml:1.0:function:double-subset	M
urn:oasis:names:tc:xacml:1.0:function:double-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:time-intersection	M
urn:oasis:names:tc:xacml:1.0:function:time-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:time-union	M
urn:oasis:names:tc:xacml:1.0:function:time-subset	M
urn:oasis:names:tc:xacml:1.0:function:time-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:date-intersection	M
urn:oasis:names:tc:xacml:1.0:function:date-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:date-union	M
urn:oasis:names:tc:xacml:1.0:function:date-subset	M
urn:oasis:names:tc:xacml:1.0:function:date-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-intersection	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-union	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-subset	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-intersection	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-union	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-subset	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-intersection	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-union	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-subset	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-intersection	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-union	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-subset	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-intersection	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-union	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-subset	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-intersection	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-union	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-subset	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-intersection	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-union	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-subset	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-intersection	M

urn:oasis:names:tc:xacml:1.0:function:rfc822Name-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-union	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-subset	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-set-equals	M

11. References

- 3324
- 3325 **[DS]** D. Eastlake et al., *XML-Signature Syntax and Processing*,
3326 <http://www.w3.org/TR/xmlsig-core/>, World Wide Web Consortium.
- 3327 **[Haskell]** Haskell, a purely functional language. Available at
3328 <http://www.haskell.org/>
- 3329 **[Hinton94]** Hinton, H, M, Lee,, E, S, The Compatibility of Policies, Proceedings 2nd
3330 ACM Conference on Computer and Communications Security, Nov 1994,
3331 Fairfax, Virginia, USA.
- 3332 **[IEEE754]** IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-
3333 7653-8, IEEE Product No. SH10116-TBR
- 3334 **[Kudo00]** Kudo M and Hada S, XML document security based on provisional
3335 authorization, Proceedings of the Seventh ACM Conference on Computer
3336 and Communications Security, Nov 2000, Athens, Greece, pp 87-96.
- 3337 **[LDAP-1]** RFC2256, A summary of the X500(96) User Schema for use with LDAPv3,
3338 section 5, M Wahl, December 1997 <http://www.ietf.org/rfc/rfc2798.txt>
- 3339 **[LDAP-2]** RFC2798, Definition of the inetOrgPerson, M. Smith, April 2000
3340 <http://www.ietf.org/rfc/rfc2798.txt>
- 3341 **[MathML]** Mathematical Markup Language (MathML), Version 2.0, W3C
3342 Recommendation, 21 February 2001. Available at:
3343 <http://www.w3.org/TR/MathML2/>
- 3344 **[Perritt93]** Perritt, H. Knowbots, Permissions Headers and Contract Law, Conference
3345 on Technological Strategies for Protecting Intellectual Property in the
3346 Networked Multimedia Environment, April 1993. Available at:
3347 <http://www.ifla.org/documents/infopol/copyright/perh2.txt>
- 3348 **[RBAC]** Role-Based Access Controls, David Ferraiolo and Richard Kuhn, 15th
3349 National Computer Security Conference, 1992. Available at:
3350 <http://csrc.nist.gov/rbac>
- 3351 **[RegEx]** XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001,
3352 Appendix D. Available at: <http://www.w3.org/TR/xmlschema-0/>
- 3353 **[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
3354 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997
- 3355 **[SAML]** Security Assertion Markup Language available from [http://www.oasis-
3356 open.org/committees/security/#documents](http://www.oasis-open.org/committees/security/#documents)
- 3357 **[Sloman94]** Sloman, M. Policy Driven Management for Distributed Systems. Journal
3358 of Network and Systems Management, Volume 2, part 4. Plenum Press.
3359 1994.
- 3360 **[XF]** XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft
3361 16 August 2002. Available at: <http://www.w3.org/TR/xquery-operators>
- 3362 **[XS]** XML Schema. Available at: [http://www.w3.org/TR/2001/REC-
3363 xmlschema-2-20010502/](http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/)
- 3364 **[XPath]** XML Path Language (XPath), Version 1.0, W3C Recommendation 16
3365 November 1999. Available at: <http://www.w3.org/TR/xpath>

3366	[XQO]	XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft
3367		15 November 2002. Available at: http://www.w3.org/TR/2002/WD-xquery-
3368		operators-20021115/
3369	[XSLT]	XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16
3370		November 1999. Available at: http://www.w3.org/TR/xslt
3371		

3372 **Appendix A. Standard data-types, functions and**
3373 **their semantics (normative)**

3374 **A.1. Introduction**

3375 This section contains a specification of the data-types and functions used in XACML to create
3376 **predicates** for a **rule's condition** and **target** matches.

3377 This specification combines the various standards set forth by IEEE and ANSI for string
3378 representation of numeric values, as well as the evaluation of arithmetic functions.

3379 This section describes the primitive data-types, **bags** and construction of expressions using
3380 XACML constructs. Finally, each standard function is named and its operational semantics are
3381 described.

3382 **A.2. Primitive types**

3383 Although XML instances represent all data-types as strings, an XACML **PDP** must reason about
3384 types of data that, while they have string representations, are not just strings. Types such as
3385 boolean, integer and double **MUST** be converted from their XML string representations to values
3386 that can be compared with values in their domain of discourse, such as numbers. The following
3387 primitive data-types are specified for use with XACML and have explicit data representations:

- 3388 • <http://www.w3.org/2001/XMLSchema#string>
- 3389 • <http://www.w3.org/2001/XMLSchema#boolean>
- 3390 • <http://www.w3.org/2001/XMLSchema#integer>
- 3391 • <http://www.w3.org/2001/XMLSchema#double>
- 3392 • <http://www.w3.org/2001/XMLSchema#time>
- 3393 • <http://www.w3.org/2001/XMLSchema#date>
- 3394 • <http://www.w3.org/2001/XMLSchema#dateTime>
- 3395 • <http://www.w3.org/2001/XMLSchema#anyURI>
- 3396 • <http://www.w3.org/2001/XMLSchema#hexBinary>
- 3397 • <http://www.w3.org/2001/XMLSchema#base64Binary>
- 3398 • <http://www.w3.org/TR/xquery-operators#dayTimeDuration>
- 3399 • <http://www.w3.org/TR/xquery-operators#yearMonthDuration>
- 3400 • <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
- 3401 • <urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name>

3402 A.3. Structured types

3403 An XACML <AttributeValue> element MAY contain an instance of a structured XML data-type,
3404 for example <ds:KeyInfo>. XACML 1.0 supports several ways for comparing such
3405 <AttributeValue> elements.

3406 1. In some cases, such an <AttributeValue> element MAY be compared using one of the
3407 XACML string functions, such as “regexp-string-match”, described below. This requires
3408 that the structured data <AttributeValue> be given the DataType="xsi:string". For example,
3409 a structured data-type that is actually a ds:KeyInfo/KeyName would appear in the Context
3410 as:

```
3411 <AttributeValue  
3412   DataType="http://www.w3.org/2001/XMLSchema-  
3413   instance#string">&lt;ds:KeyName&gt;jhibbert-key&lt;/ds:KeyName&gt;  
3414 </AttributeValue>
```

3415 In general, this method will not be adequate unless the structured data-type is quite simple.

3416 2. An <AttributeSelector> element MAY be used to select the value of a leaf sub-
3417 element of the structured data-type by means of an XPath expression. That value MAY
3418 then be compared using one of the supported XACML functions appropriate for its primitive
3419 data-type. This method requires support by the **PDP** for the optional XPath expressions
3420 feature.

3421 3. An <AttributeSelector> element MAY be used to select the value of any node in the
3422 structured data-type by means of an XPath expression. This node MAY then be compared
3423 using one of the XPath-based functions described in Section A14.13. This method requires
3424 support by the **PDP** for the optional XPath expressions and XPath functions features.

3425 A.4. Representations

3426 An XACML **PDP** SHALL be capable of converting string representations into various primitive data-
3427 types. For integers and doubles, XACML SHALL use the conversions described in [IEEE754].

3428 This document combines the various standards set forth by IEEE and ANSI for string
3429 representation of numeric values.

3430 XACML defines two additional data-types; these are “urn:oasis:names:tc:xacml:1.0:data-
3431 type:x500Name” and “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name”. These types
3432 represent identifiers for **subjects** and appear in several standard applications, such as TLS/SSL
3433 and electronic mail.

3434 The “urn:oasis:names:tc:xacml:1.0:data-type:x500Name” primitive type represents an X.500
3435 Distinguished Name. The string representation of an X.500 distinguished name is specified in IETF
3436 RFC 2253 “Lightweight Directory Access Protocol (v3): UTF-8 String Representation of
3437 Distinguished Names”.¹

3438 The “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name” primitive type represents electronic mail
3439 addresses, and its string representation is specified by RFC 822.

1 An earlier RFC, RFC 1779 “A String Representation of Distinguished Names”, is less restrictive, so urn:oasis:names:tc:xacml:1.0:data-type:x500Name uses the syntax in RFC 2253 for better interoperability.

3440 An RFC822 name consists of a *local-part* followed by "@" followed by a *domain-part*. The *local-*
3441 *part* is case-sensitive, while the *domain-part* (which is usually a DNS host name) is not case-
3442 sensitive.²

3443 A.5. Bags

3444 XACML defines implicit collections of its primitive types. XACML refers to a collection of values that
3445 are of a single primitive type as a **bag**. **Bags** of primitive types are needed because selections of
3446 nodes from an XML **resource** or XACML request **context** may return more than one value.

3447 The <AttributeSelector> element uses an XPath expression to specify the selection of data
3448 from an XML **resource**. The result of an XPath expression is termed a *node-set*, which contains all
3449 the leaf nodes from the XML **resource** that match the predicate in the XPath expression. Based on
3450 the various indexing functions provided in the XPath specification, it SHALL be implied that a
3451 resultant node-set is the collection of the matching nodes. XACML also defines the
3452 <AttributeDesignator> **element** to have the same matching methodology for attributes in the
3453 XACML request **context**.

3454 The values in a **bag** are not ordered, and some of the values may be duplicates. There SHALL be
3455 no notion of a **bag** containing **bags**, or a **bag** containing values of differing types. I.e. a **bag** in
3456 XACML SHALL contain only values that are of the same primitive type.

3457 A.6. Expressions

3458 XACML specifies expressions in terms of the following elements. Each expression evaluates to
3459 one of the primitive types, or a **bag** of one of the primitive types. In addition, XACML defines an
3460 evaluation result of "Indeterminate", which is said to be the result of an invalid expression, or an
3461 operational error occurring during the evaluation of the expression.

3462 XACML defines the following elements to be legal XACML expressions:

- 3463 • <AttributeValue>
- 3464 • <SubjectAttributeDesignator>
- 3465 • <SubjectAttributeSelector>
- 3466 • <ResourceAttributeDesignator>
- 3467 • <ActionAttributeDesignator>
- 3468 • <EnvironmentAttributeDesignator>
- 3469 • <AttributeSelector>
- 3470 • <Apply>
- 3471 • <Condition>

2 According to IETF RFC822 and its successor specifications [RFC2821], case is significant in the *local-part*. However, many mail systems, as well as the IETF PKIX specification, treat the *local-part* as case-insensitive. This is considered an error by mail-system designers and is not encouraged.

- 3472 • <Function>

3473 A.7. Element <AttributeValue>

3474 The <AttributeValue> element SHALL represent an explicit value of a primitive type. For
3475 example:

```
3476 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-equal">  
3477   <AttributeValue  
3478   DataType="http://www.w3.org/2001/XMLSchema#integer">123</AttributeValue>  
3479   <AttributeValue  
3480   DataType="http://www.w3.org/2001/XMLSchema#integer">123</AttributeValue>  
3481 </Apply>
```

3482 A.8. Elements <AttributeDesignator> and 3483 <AttributeSelector>

3484 The <AttributeDesignator> and <AttributeSelector> elements SHALL evaluate to a **bag**
3485 of a specific primitive type. The type SHALL be inferred from the function in which it appears. Each
3486 element SHALL contain a URI or XPath expression, respectively, to identify the required **attribute**
3487 values. If an operational error were to occur while finding the values, the value of the element
3488 SHALL be set to "Indeterminate". If the required **attribute** cannot be located, then the value of the
3489 element SHALL be set to an empty **bag** of the inferred primitive type.

3490 A.9. Element <Apply>

3491 XACML function calls are represented by the <Apply> element. The function to be applied is
3492 named in the `FunctionId` attribute of this element. The value of the <Apply> element SHALL be
3493 set to either a primitive data-type or a **bag** of a primitive type, whose data-type SHALL be inferred
3494 from the `FunctionId`. The arguments of a function SHALL be the values of the XACML
3495 expressions that are contained as ordered elements in an <Apply> element. The legal number of
3496 arguments within an <Apply> element SHALL depend upon the `functionId`.

3497 A.10. Element <Condition>

3498 The <Condition> element MAY appear in the <Rule> element as the premise for emitting the
3499 corresponding **effect** of the **rule**. The <Condition> element has the same structure as the
3500 <Apply> element, with the restriction that its result SHALL be of data-type
3501 "http://www.w3.org/2001/XMLSchema#boolean". The evaluation of the <Condition> element
3502 SHALL follow the same evaluation semantics as those of the <Apply> element.

3503 A.11. Element <Function>

3504 The <Function> element names a standard XACML function or an extension function in its
3505 FunctionId attribute. The <Function> element MAY be used as an argument in functions that
3506 take a function as an argument.

3507 A.12. Matching elements

3508 Matching elements appear in the <Target> element of *rules*, *policies* and *policy sets*. They are
3509 the following:

3510 <SubjectMatch>

3511 <ResourceMatch>

3512 <ActionMatch>

3513 These elements represent boolean expressions over attributes of the subject, resource, and action,
3514 respectively. A matching element contains a MatchId attribute that specifies the function to be
3515 used in performing the match evaluation, an **attribute value**, and an <AttributeDesignator>
3516 or <AttributeSelector> element that specifies the **attribute** in the **context** that is to be
3517 matched against the specified value.

3518 The MatchId attribute SHALL specify a function that compares two arguments, returning a result
3519 type of "http://www.w3.org/2001/XMLSchema#boolean". The **attribute** value specified in the
3520 matching element SHALL be supplied to the MatchId function as its first argument. An element of
3521 the **bag** returned by the <AttributeDesignator> or <AttributeSelector> element SHALL
3522 be supplied to the MatchId function as its second argument. The data-type of the **attribute** value
3523 SHALL match the data-type of the first argument expected by the MatchId function. The data-type
3524 of the <AttributeDesignator> or <AttributeSelector> element SHALL match the data-
3525 type of the second argument expected by the MatchId function.

3526 The XACML standard functions that meet the requirements for use as a MatchId attribute value
3527 are:

3528 urn:oasis:names:tc:xacml:1.0:function:-type-equal

3529 urn:oasis:names:tc:xacml:1.0:function:-type-greater-than

3530 urn:oasis:names:tc:xacml:1.0:function:-type-greater-than-or-equal

3531 urn:oasis:names:tc:xacml:1.0:function:-type-less-than

3532 urn:oasis:names:tc:xacml:1.0:function:-type-less-than-or-equal

3533 urn:oasis:names:tc:xacml:1.0:function:-type-match

3534 In addition, functions that are strictly within an extension to XACML MAY appear as a value for the
3535 MatchId attribute, and those functions MAY use data-types that are also extensions, so long as
3536 the extension function returns a boolean result and takes an **attribute** value as its first argument
3537 and an <AttributeDesignator> or <AttributeSelector> as its second argument. The
3538 function used as the value for the MatchId attribute SHOULD be easily indexable. Use of non-
3539 indexable or complex functions may prevent efficient evaluation of **decision requests**.

3540 The evaluation semantics for a matching element is as follows. If an operational error were to
3541 occur while evaluating the <AttributeDesignator> or <AttributeSelector> element, then

3542 the result of the entire expression SHALL be "Indeterminate". If the <AttributeDesignator> or
3543 <AttributeSelector> element were to evaluate to an empty **bag**, then the result of the
3544 expression SHALL be "False". Otherwise, the MatchId function SHALL be applied between the
3545 explicit **attribute** value and each element of the **bag** returned from the <AttributeDesignator>
3546 or <AttributeSelector> element. If at least one of those function applications were to evaluate
3547 to "True", then the result of the entire expression SHALL be "True". Otherwise, if at least one of the
3548 function applications results in "Indeterminate", then the result SHALL be "Indeterminate". Finally,
3549 only if all function applications evaluate to "False", the result of the entire expression SHALL be
3550 "False".

3551 It is possible to express the semantics of a **target** matching element in a **condition**. For instance,
3552 the **target** match expression that compares a "subject-name" starting with the name "John" can be
3553 expressed as follows:

```
3554 <SubjectMatch  
3555     MatchId="urn:oasis:names:tc:xacml:1.0:function:regexp-string-match">  
3556   <SubjectAttributeDesignator  
3557     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3558     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3559   <AttributeValue  
3560     DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>  
3561 </SubjectMatch>
```

3562 Alternatively, the same match semantics can be expressed as an <Apply> element in a **condition**
3563 by using the "urn:oasis:names:tc:xacml:1.0:function:any-of" function, as follows:

```
3564 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">  
3565   <Function  
3566     FunctionId="urn:oasis:names:tc:xacml:1.0:function:regexp-string-match"/>  
3567   <AttributeValue  
3568     DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>  
3569   <SubjectAttributeDesignator  
3570     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3571     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3572 </Apply>
```

3573

3574 This expression of the semantics is NOT normative.

3575 **A.13. Arithmetic evaluation**

3576 IEEE 754 [IEEE 754] specifies how to evaluate arithmetic functions in a context, which specifies
3577 defaults for precision, rounding, etc. XACML SHALL use this specification for the evaluation of all
3578 integer and double functions relying on the *Extended Default Context*, enhanced with double
3579 precision:

3580 flags - all set to 0

3581 trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the "division-by-zero" trap
3582 enabler, which SHALL be set to 1

3583 precision - is set to the designated double precision

3584 rounding - is set to round-half-even (IEEE 854 §4.1)

3585 **A.14. XACML standard functions**

3586 XACML specifies the following functions that are prefixed with the
3587 "urn:oasis:names:tc:xacml:1.0:function:" relative name space identifier.

3588 **A14.1 Equality predicates**

3589 The following functions are the *equality* functions for the various primitive types. Each function for a
3590 particular data-type follows a specified standard convention for that data-type. If an argument of
3591 one of these functions were to evaluate to "Indeterminate", then the function SHALL be set to
3592 "Indeterminate".

- 3593 • string-equal

3594 This function SHALL take two arguments of "http://www.w3.org/2001/XMLSchema#string"
3595 and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". The function
3596 SHALL return "True" if and only if the value of both of its arguments are of equal length and
3597 each string is determined to be equal byte-by-byte according to the function "integer-equal".

- 3598 • boolean-equal

3599 This function SHALL take two arguments of
3600 "http://www.w3.org/2001/XMLSchema#boolean" and SHALL return "True" if and only if both
3601 values are equal.

- 3602 • integer-equal

3603 This function SHALL take two arguments of data-type
3604 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return an
3605 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on
3606 integers according to IEEE 754 [IEEE 754].

- 3607 • double-equal

3608 This function SHALL take two arguments of data-type
3609 "http://www.w3.org/2001/XMLSchema#double" and SHALL return an
3610 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on
3611 doubles according to IEEE 754 [IEEE 754].

- 3612 • date-equal

3613 This function SHALL take two arguments of data-type
3614 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
3615 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
3616 according to the "op:date-equal" function [XQO Section 8.3.11].

- 3617 • time-equal

3618 This function SHALL take two arguments of data-type
3619 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
3620 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according
3621 to the "op:time-equal" function [XQO Section 8.3.14].

- 3622 • dateTime-equal

3623 This function SHALL take two arguments of data-type
3624 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an

3625 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
3626 according to the "op:dateTime-equal" function [XQO Section 8.3.8].

3627 • **dayTimeDuration-equal**

3628 This function SHALL take two arguments of data-type "http://www.w3.org/TR/xquery-
3629 operators#dayTimeDuration" and SHALL return an
3630 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
3631 according to the "op:dayTimeDuration-equal" function [XQO Section 8.3.5]. Note that the
3632 lexical representation of each argument MUST be converted to a value expressed in
3633 fractional seconds [XQO Section 8.2.2].

3634 • **yearMonthDuration-equal**

3635 This function SHALL take two arguments of data-type "http://www.w3.org/TR/xquery-
3636 operators#yearMonthDuration" and SHALL return an
3637 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
3638 according to the "op:yearMonthDuration-equal" function [XQO Section 8.3.2]. Note that the
3639 lexical representation of each argument MUST be converted to a value expressed in
3640 integer months [XQO Section 8.2.1].

3641 • **anyURI-equal**

3642 This function SHALL take two arguments of data-type
3643 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return an
3644 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
3645 according to the "op:anyURI-equal" function [XQO Section 10.2.1].

3646 • **x500Name-equal**

3647 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-
3648 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It
3649 shall return "True" if and only if each Relative Distinguished Name (RDN) in the two
3650 arguments matches. Two RDNs shall be said to match if and only if the result of the
3651 following operations is "True"³.

- 3652 1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory
3653 Access Protocol (v3): UTF-8 String Representation of Distinguished Names".
- 3654 2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute
3655 ValuePairs in that RDN in ascending order when compared as octet strings
3656 (described in ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").
- 3657 3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key
3658 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section
3659 4.1.2.4 "Issuer".

3660 • **rfc822Name-equal**

3661 This function SHALL take two arguments of data-type "urn:oasis:names:tc:xacml:1.0:data-
3662 type:rfc822Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean".
3663 This function SHALL determine whether two "urn:oasis:names:tc:xacml:1.0:data-
3664 type:rfc822Name" arguments are equal. An RFC822 name consists of a *local-part* followed
3665 by "@" followed by a *domain-part*. The *local-part* is case-sensitive, while the *domain-part*
3666 (which is usually a DNS host name) is not case-sensitive. Perform the following
3667 operations:

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.

- 3668 1. Normalize the *domain*-part of each argument to lower case
- 3669 2. Compare the expressions by applying the function
- 3670 “urn:oasis:names:tc:xacml:1.0:function:string-equal” to the normalized arguments.

3671 • hexBinary-equal

3672 This function SHALL take two arguments of data-type

3673 “http://www.w3.org/2001/XMLSchema#hexBinary” and SHALL return an

3674 “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL return "True" if the

3675 octet sequences represented by the value of both arguments have equal length and are

3676 equal in a conjunctive, point-wise, comparison using the

3677 “urn:oasis:names:tc:xacml:1.0:function:integer-equal”. The conversion from the string

3678 representation to an octet sequence SHALL be as specified in [XS Section 8.2.15]

3679 • base64Binary-equal

3680 This function SHALL take two arguments of data-type

3681 “http://www.w3.org/2001/XMLSchema#base64Binary” and SHALL return an

3682 “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL return "True" if the

3683 octet sequences represented by the value of both arguments have equal length and are

3684 equal in a conjunctive, point-wise, comparison using the

3685 “urn:oasis:names:tc:xacml:1.0:function:integer-equal”. The conversion from the string

3686 representation to an octet sequence SHALL be as specified in [XS Section 8.2.16]

3687 **A14.2 Arithmetic functions**

3688 All of the following functions SHALL take two arguments of the specified *data-type*, integer or

3689 double, and SHALL return an element of integer or double data-type, respectively. However, the

3690 “add” functions MAY take more than two arguments. Each function evaluation SHALL proceed as

3691 specified by their logical counterparts in IEEE 754 [IEEE 754]. In an expression that contains any

3692 of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to

3693 "Indeterminate". In the case of the divide functions, if the divisor is zero, then the function SHALL

3694 evaluate to “Indeterminate”.

3695 • integer-add

3696 This function MAY have two or more arguments.

3697 • double-add

3698 This function MAY have two or more arguments.

3699 • integer-subtract

3700 • double-subtract

3701 • integer-multiply

3702 • double-multiply

3703 • integer-divide

3704 • double-divide

3705 • integer-mod

3706 The following functions SHALL take a single argument of the specified *data-type*. The round and

3707 floor functions SHALL take a single argument of data-type

3708 “http://www.w3.org/2001/XMLSchema#double” and return data-type

3709 "http://www.w3.org/2001/XMLSchema#double". In an expression that contains any of these
3710 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3711 "Indeterminate".

3712 • integer-abs

3713 • double-abs

3714 • round

3715 • floor

3716 **A14.3 String conversion functions**

3717 The following functions convert between values of the XACML
3718 "http://www.w3.org/2001/XMLSchema#string" primitive types. In an expression that contains any of
3719 these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3720 "Indeterminate".

3721 • string-normalize-space

3722 This function SHALL take one argument of data-type
3723 "http://www.w3.org/2001/XMLSchema#string" and SHALL normalize the value by stripping
3724 off all leading and trailing whitespace characters.

3725 • string-normalize-to-lower-case

3726 This function SHALL take one argument of "http://www.w3.org/2001/XMLSchema#string"
3727 and SHALL normalize the value by converting each upper case character to its lower case
3728 equivalent.

3729 **A14.4 Numeric data-type conversion functions**

3730 The following functions convert between the XACML
3731 "http://www.w3.org/2001/XMLSchema#integer" and "http://www.w3.org/2001/XMLSchema#double"
3732 primitive types. In any expression in which the functions defined below are applied, if any argument
3733 while being evaluated results in "Indeterminate", the expression SHALL return "Indeterminate".

3734 • double-to-integer

3735 This function SHALL take one argument of data-type
3736 "http://www.w3.org/2001/XMLSchema#double" and SHALL truncate its numeric value to a
3737 whole number and return an element of data-type
3738 "http://www.w3.org/2001/XMLSchema#integer".

3739 • integer-to-double

3740 This function SHALL take one argument of data-type
3741 "http://www.w3.org/2001/XMLSchema#integer" and SHALL promote its value to an element
3742 of data-type "http://www.w3.org/2001/XMLSchema#double" of the same numeric value.

3743 **A14.5 Logical functions**

3744 This section contains the specification for logical functions that operate on arguments of the
3745 "http://www.w3.org/2001/XMLSchema#boolean" data-type.

3746 • or

3747 This function SHALL return "False" if it has no arguments and SHALL return "True" if one of
3748 its arguments evaluates to "True". The order of evaluation SHALL be from first argument to
3749 last. The evaluation SHALL stop with a result of "True" if any argument evaluates to "True",
3750 leaving the rest of the arguments unevaluated. In an expression that contains any of these
3751 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3752 "Indeterminate".

3753 • and

3754 This function SHALL return "True" if it has no arguments and SHALL return "False" if one of
3755 its arguments evaluates to "False". The order of evaluation SHALL be from first argument
3756 to last. The evaluation SHALL stop with a result of "False" if any argument evaluates to
3757 "False", leaving the rest of the arguments unevaluated. In an expression that contains any
3758 of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate
3759 to "Indeterminate".

3760 • n-of

3761 The first argument to this function SHALL be of data-type
3762 "http://www.w3.org/2001/XMLSchema#integer", specifying the number of the remaining
3763 arguments that MUST evaluate to "True" for the expression to be considered "True". If the
3764 first argument is 0, the result SHALL be "True". If the number of arguments after the first
3765 one is less than the value of the first argument, then the expression SHALL result in
3766 "Indeterminate". The order of evaluation SHALL be: first evaluate the integer value, then
3767 evaluate each subsequent argument. The evaluation SHALL stop and return "True" if the
3768 specified number of arguments evaluate to "True". The evaluation of arguments SHALL
3769 stop if it is determined that evaluating the remaining arguments will not satisfy the
3770 requirement. In an expression that contains any of these functions, if any argument is
3771 "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

3772 • not

3773 This function SHALL take one logical argument. If the argument evaluates to "True", then
3774 the result of the expression SHALL be "False". If the argument evaluates to "False", then
3775 the result of the expression SHALL be "True". In an expression that contains any of these
3776 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3777 "Indeterminate".

3778 **A14.6 Arithmetic comparison functions**

3779 These functions form a minimal set for comparing two numbers, yielding a boolean result. They
3780 SHALL comply with the rules governed by IEEE 754 [IEEE 754]. In an expression that contains
3781 any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3782 "Indeterminate".

3783 • integer-greater-than

3784 • integer-greater-than-or-equal

3785 • integer-less-than

3786 • integer-less-than-or-equal

3787 • double-greater-than

3788 • double-greater-than-or-equal

3789 • double-less-than

- 3790 • double-less-than-or-equal

3791 **A14.7 Date and time arithmetic functions**

3792 These functions perform arithmetic operations with the date and time. In an expression that
3793 contains any of these functions, if any argument is "Indeterminate", then the expression SHALL
3794 evaluate to "Indeterminate".

- 3795 • dateTime-add-dayTimeDuration

3796 This function SHALL take two arguments, the first is of data-type
3797 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is of data-type
3798 "http://www.w3.org/TR/xquery-operators#dayTimeDuration". It SHALL return a result of
3799 "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL return the value by
3800 adding the second argument to the first argument according to the specification of adding
3801 durations to date and time [XS Appendix E].

- 3802 • dateTime-add-yearMonthDuration

3803 This function SHALL take two arguments, the first is a
3804 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a
3805 "http://www.w3.org/TR/xquery-operators#yearMonthDuration". It SHALL return a result of
3806 "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL return the value by
3807 adding the second argument to the first argument according to the specification of adding
3808 durations to date and time [XS Appendix E].

- 3809 • dateTime-subtract-dayTimeDuration

3810 This function SHALL take two arguments, the first is a
3811 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a
3812 "http://www.w3.org/TR/xquery-operators#dayTimeDuration". It SHALL return a result of
3813 "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument is a positive
3814 duration, then this function SHALL return the value by adding the corresponding negative
3815 duration, as per the specification [XS Appendix E]. If the second argument is a negative
3816 duration, then the result SHALL be as if the function
3817 "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration" had been applied
3818 to the corresponding positive duration.

- 3819 • dateTime-subtract-yearMonthDuration

3820 This function SHALL take two arguments, the first is a
3821 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a
3822 "http://www.w3.org/TR/xquery-operators#yearMonthDuration". It SHALL return a result of
3823 "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument is a positive
3824 duration, then this function SHALL return the value by adding the corresponding negative
3825 duration, as per the specification [XS Appendix E]. If the second argument is a negative
3826 duration, then the result SHALL be as if the function
3827 "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration" had been
3828 applied to the corresponding positive duration.

- 3829 • date-add-yearMonthDuration

3830 This function SHALL take two arguments, the first is a
3831 "http://www.w3.org/2001/XMLSchema#date" and the second is a
3832 "http://www.w3.org/TR/xquery-operators#yearMonthDuration". It return a result of
3833 "http://www.w3.org/2001/XMLSchema#date". This function SHALL return the value by
3834 adding the second argument to the first argument according to the specification of adding
3835 durations to date [XS Appendix E].

- 3836 • date-subtract-yearMonthDuration

3837 This function SHALL take two arguments, the first is a
3838 "http://www.w3.org/2001/XMLSchema#date" and the second is a
3839 "http://www.w3.org/TR/xquery-operators#yearMonthDuration". It SHALL return a result of
3840 "http://www.w3.org/2001/XMLSchema#date". If the second argument is a positive duration,
3841 then this function SHALL return the value by adding the corresponding negative duration,
3842 as per the specification [XS Appendix E]. If the second argument is a negative duration,
3843 then the result SHALL be as if the function "urn:oasis:names:tc:xacml:1.0:function:date-
3844 add-yearMonthDuration" had been applied to the corresponding positive duration.

3845 **A14.8 Non-numeric comparison functions**

3846 These functions perform comparison operations on two arguments of non-numerical types. In an
3847 expression that contains any of these functions, if any argument is "Indeterminate", then the
3848 expression SHALL evaluate to "Indeterminate".

- 3849 • string-greater-than

3850 This function SHALL take two arguments of data-type
3851 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
3852 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
3853 arguments are compared byte by byte and, after an initial prefix of corresponding bytes
3854 from both arguments that are considered equal by
3855 "urn:oasis:names:tc:xacml:1.0:function:integer-equal", the next byte by byte comparison is
3856 such that the byte from the first argument is greater than the byte from the second
3857 argument by the use of the function "urn:oasis:names:tc:xacml:1.0:function:integer-equal".

- 3858 • string-greater-than-or-equal

3859 This function SHALL take two arguments of data-type
3860 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
3861 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return a result as if evaluated
3862 with the logical function "urn:oasis:names:tc:xacml:1.0:function:or" with two arguments
3863 containing the functions "urn:oasis:names:tc:xacml:1.0:function:string-greater-than" and
3864 "urn:oasis:names:tc:xacml:1.0:function:string-equal" containing the original arguments

- 3865 • string-less-than

3866 This function SHALL take two arguments of data-type
3867 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
3868 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
3869 arguments are compared byte by byte and, after an initial prefix of corresponding bytes
3870 from both arguments are considered equal by
3871 "urn:oasis:names:tc:xacml:1.0:function:integer-equal", the next byte by byte comparison is
3872 such that the byte from the first argument is less than the byte from the second argument
3873 by the use of the function "urn:oasis:names:tc:xacml:1.0:function:integer-less-than".

- 3874 • string-less-than-or-equal

3875 This function SHALL take two arguments of data-type
3876 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
3877 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return a result as if evaluated
3878 with the function "urn:oasis:names:tc:xacml:1.0:function:or" with two arguments containing
3879 the functions "urn:oasis:names:tc:xacml:1.0:function:string-less-than" and
3880 "urn:oasis:names:tc:xacml:1.0:function:string-equal" containing the original arguments.

- 3881 • time-greater-than

3882 This function SHALL take two arguments of data-type
3883 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
3884 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3885 argument is greater than the second argument according to the order relation specified for
3886 "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

3887 • time-greater-than-or-equal

3888 This function SHALL take two arguments of data-type
3889 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
3890 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3891 argument is greater than or equal to the second argument according to the order relation
3892 specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

3893 • time-less-than

3894 This function SHALL take two arguments of data-type
3895 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
3896 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3897 argument is less than the second argument according to the order relation specified for
3898 "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

3899 • time-less-than-or-equal

3900 This function SHALL take two arguments of data-type
3901 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
3902 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3903 argument is less than or equal to the second argument according to the order relation
3904 specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

3905 • dateTime-greater-than

3906 This function SHALL take two arguments of data-type
3907 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
3908 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3909 argument is greater than the second argument according to the order relation specified for
3910 "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

3911 • dateTime-greater-than-or-equal

3912 This function SHALL take two arguments of data-type
3913 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
3914 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3915 argument is greater than or equal to the second argument according to the order relation
3916 specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

3917 • dateTime-less-than

3918 This function SHALL take two arguments of data-type
3919 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
3920 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3921 argument is less than the second argument according to the order relation specified for
3922 "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

3923 • dateTime-less-than-or-equal

3924 This function SHALL take two arguments of data-type
3925 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
3926 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first

- 3927 argument is less than or equal to the second argument according to the order relation
3928 specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].
- 3929 • date-greater-than
- 3930 This function SHALL take two arguments of data-type
3931 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
3932 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3933 argument is greater than the second argument according to the order relation specified for
3934 "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].
- 3935 • date-greater-than-or-equal
- 3936 This function SHALL take two arguments of data-type
3937 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
3938 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3939 argument is greater than or equal to the second argument according to the order relation
3940 specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].
- 3941 • date-less-than
- 3942 This function SHALL take two arguments of data-type
3943 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
3944 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3945 argument is less than the second argument according to the order relation specified for
3946 "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].
- 3947 • date-less-than-or-equal
- 3948 This function SHALL take two arguments of data-type
3949 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
3950 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3951 argument is less than or equal to the second argument according to the order relation
3952 specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

3953 **A14.9 Bag functions**

- 3954 These functions operate on a **bag** of *type* values, where *data-type* is one of the primitive types. In
3955 an expression that contains any of these functions, if any argument is "Indeterminate", then the
3956 expression SHALL evaluate to "Indeterminate". Some additional conditions defined for each
3957 function below SHALL cause the expression to evaluate to "Indeterminate".
- 3958 • *type-one-and-only*
- 3959 This function SHALL take an argument of a **bag** of *type* values and SHALL return a value
3960 of *data-type*. It SHALL return the only value in the **bag**. If the **bag** does not have one and
3961 only one value, then the expression SHALL evaluate to "Indeterminate".
- 3962 • *type-bag-size*
- 3963 This function SHALL take a **bag** of *type* values as an argument and SHALL return an
3964 "http://www.w3.org/2001/XMLSchema#integer" indicating the number of values in the **bag**.
- 3965 • *type-is-in*
- 3966 This function SHALL take an argument of data-type *type* as the first argument and a **bag** of
3967 *type* values as the second argument. The expression SHALL evaluate to "True" if the first
3968 argument matches by the "urn:oasis:names:tc:xacml:1.0:function:type-equal" to any value
3969 in the **bag**.

- 3970 • *type-bag*
- 3971 This function SHALL take any number of arguments of a single data-type and return a **bag**
3972 of *type* values containing the values of the arguments. An application of this function to
3973 zero arguments SHALL produce an empty **bag** of the specified data-type.

3974 **A14.10 Set functions**

3975 These functions operate on **bags** mimicking sets by eliminating duplicate elements from a **bag**. In
3976 an expression that contains any of these functions, if any argument is "Indeterminate", then the
3977 expression SHALL evaluate to "Indeterminate".

- 3978 • *type-intersection*
- 3979 This function SHALL take two arguments that are both a **bag** of *type* values. The
3980 expression SHALL return a **bag** of *type* values such that it contains only elements that are
3981 common between the two **bags**, which is determined by
3982 "urn:oasis:names:tc:xacml:1.0:function:type-equal". No duplicates as determined by
3983 "urn:oasis:names:tc:xacml:1.0:function:type-equal" SHALL exist in the result.
- 3984 • *type-at-least-one-member-of*
- 3985 This function SHALL take two arguments that are both a **bag** of *type* values. The
3986 expression SHALL evaluate to "True" if at least one element of the first argument is
3987 contained in the second argument as determined by
3988 "urn:oasis:names:tc:xacml:1.0:function:type-is-in".
- 3989 • *type-union*
- 3990 This function SHALL take two arguments that are both a **bag** of *type* values. The
3991 expression SHALL return a **bag** of *type* such that it contains all elements of both **bags**. No
3992 duplicates as determined by "urn:oasis:names:tc:xacml:1.0:function:type-equal" SHALL
3993 exist in the result.
- 3994 • *type-subset*
- 3995 This function SHALL take two arguments that are both a **bag** of *type* values. It SHALL
3996 return "True" if the first argument is a subset of the second argument. Each argument is
3997 considered to have its duplicates removed as determined by
3998 "urn:oasis:names:tc:xacml:1.0:function:type-equal" before subset calculation.
- 3999 • *type-set-equals*
- 4000 This function SHALL take two arguments that are both a **bag** of *type* values and SHALL
4001 return the result of applying "urn:oasis:names:tc:xacml:1.0:function:and" to the application
4002 of "urn:oasis:names:tc:xacml:1.0:function:type-subset" to the first and second arguments
4003 and the application of "urn:oasis:names:tc:xacml:1.0:function:type-subset" to the second
4004 and first arguments.

4005 **A14.11 Higher-order bag functions**

4006 This section describes functions in XACML that perform operations on **bags** such that functions
4007 may be applied to the **bags** in general.

4008 In this section, a general-purpose functional language called Haskell [**Haskell**] is used to formally
4009 specify the semantics of these functions. Although the English description is adequate, a formal
4010 specification of the semantics is helpful.

4011 For a quick summary, in the following Haskell notation, a function definition takes the form of
4012 clauses that are applied to patterns of structures, namely lists. The symbol “[]” denotes the empty
4013 list, whereas the expression “(x:xs)” matches against an argument of a non-empty list of which “x”
4014 represents the first element of the list, and “xs” is the rest of the list, which may be an empty list. We
4015 use the Haskell notion of a list, which is an ordered collection of elements, to model the XACML
4016 **bags** of values.

4017 A simple Haskell definition of a familiar function “urn:oasis:names:tc:xacml:1.0:function:and” that
4018 takes a list of booleans is defined as follows:

4019 and:: [Bool] -> Bool

4020 and [] = "True"

4021 and (x:xs) = x && (and xs)

4022 The first definition line denoted by a “::” formally describes the data-type of the function, which takes
4023 a list of booleans, denoted by “[Bool]”, and returns a boolean, denoted by “Bool”. The second
4024 definition line is a clause that states that the function “and” applied to the empty list is “True”. The
4025 second definition line is a clause that states that for a non-empty list, such that the first element is
4026 “x”, which is a value of data-type Bool, the function “and” applied to x SHALL be combined with,
4027 using the logical conjunction function, which is denoted by the infix symbol “&&”, the result of
4028 recursively applying the function “and” to the rest of the list. Of course, an application of the “and”
4029 function is “True” if and only if the list to which it is applied is empty or every element of the list is
4030 “True”. For example, the evaluation of the following Haskell expressions,

4031 (and []), (and ["True"]), (and ["True", "True"]), (and ["True", "True", "False"])

4032 evaluate to "True", "True", "True", and "False", respectively.

4033 In an expression that contains any of these functions, if any argument is "Indeterminate", then the
4034 expression SHALL evaluate to "Indeterminate".

4035 • any-of

4036 This function applies a boolean function between a specific primitive value and a **bag** of
4037 values, and SHALL return "True" if and only if the predicate is "True" for at least one
4038 element of the **bag**.

4039 This function SHALL take three arguments. The first argument SHALL be a <Function>
4040 element that names a boolean function that takes two arguments of primitive types. The
4041 second argument SHALL be a value of a primitive data-type. The third argument SHALL
4042 be a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function
4043 named in the <Function> element is applied to the second argument and each element
4044 of the third argumane (the **bag**) and the results are combined with
4045 “urn:oasis:names:tc:xacml:1.0:function:or”.

4046 In Haskell, the semantics of this operation are as follows:

4047 any_of :: (a -> b -> Bool) -> a -> [b] -> Bool

4048 any_of f a [] = "False"

4049 any_of f a (x:xs) = (f a x) || (any_of f a xs)

4050 In the above notation, “f” is the function name to be applied, “a” is the primitive value, and
4051 “(x:xs)” represents the first element of the list as “x” and the rest of the list as “xs”.

4052 For example, the following expression SHALL return "True":


```

4053 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
4054   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
4055   <AttributeValue
4056     DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4057   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4058     <AttributeValue
4059       DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4060     <AttributeValue
4061       DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4062     <AttributeValue
4063       DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4064     <AttributeValue
4065       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4066   </Apply>
4067 </Apply>

```

4068 This expression is "True" because the first argument is equal to at least one of the
4069 elements of the **bag**.

4070 • all-of

4071 This function applies a boolean function between a specific primitive value and a **bag** of
4072 values, and returns "True" if and only if the predicate is "True" for every element of the **bag**.

4073 This function SHALL take three arguments. The first argument SHALL be a <Function>
4074 element that names a boolean function that takes two arguments of primitive types. The
4075 second argument SHALL be a value of a primitive data-type. The third argument SHALL
4076 be a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function
4077 named in the <Function> element were applied to the second argument and each
4078 element of the third argument (the **bag**) and the results were combined using
4079 "urn:oasis:names:tc:xacml:1.0:function:and".

4080 In Haskell, the semantics of this operation are as follows:

```

4081 all_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
4082 all_of f a [] = "False"
4083 all_of f a (x:xs) = (f a x) && (all_of f a xs)

```

4084 In the above notation, "f" is the function name to be applied, "a" is the primitive value, and
4085 "(x:xs)" represents the first element of the list as "x" and the rest of the list as "xs".

4086 For example, the following expression SHALL evaluate to "True":

```

4087 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">
4088   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4089     greater" />
4090   <AttributeValue
4091     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4092   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4093     <AttributeValue
4094       DataType="http://www.w3.org/2001/XMLSchema#integer">9</AttributeValue>
4095     <AttributeValue
4096       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4097     <AttributeValue
4098       DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4099     <AttributeValue
4100       DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4101   </Apply>
4102 </Apply>

```

4103 This expression is "True" because the first argument is greater than *all* of the elements of
4104 the **bag**.

4105 • any-of-any

4106 This function applies a boolean function between each element of a **bag** of values and
4107 each element of another **bag** of values, and returns "True" if and only if the predicate is
4108 "True" for at least one comparison.

4109 This function SHALL take three arguments. The first argument SHALL be a <Function>
4110 element that names a boolean function that takes two arguments of primitive types. The
4111 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be
4112 a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function
4113 named in the <Function> element were applied between every element in the second
4114 argument and every element of the third argument (the **bag**) and the results were
4115 combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics are that the
4116 result of the expression SHALL be "True" if and only if the applied predicate is "True" for
4117 any comparison of elements from the two **bags**.

4118 In Haskell, taking advantage of the "any_of" function defined above, the semantics of the
4119 "any_of_any" function are as follows:

```
4120 any_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4121 any_of_any f [] ys = "False"
4122 any_of_any f (x:xs) ys = (any_of f x ys) || (any_of_any f xs ys)
```

4123 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4124 element of the list as "x" and the rest of the list as "xs".

4125 For example, the following expression SHALL evaluate to "True":

```
4126 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">
4127   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
4128   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4129     <AttributeValue
4130       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4131     <AttributeValue
4132       DataType="http://www.w3.org/2001/XMLSchema#string">Mary</AttributeValue>
4133   </Apply>
4134   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4135     <AttributeValue
4136       DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4137     <AttributeValue
4138       DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4139     <AttributeValue
4140       DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4141     <AttributeValue
4142       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4143   </Apply>
4144 </Apply>
```

4145 This expression is "True" because at least one of the elements of the first **bag**, namely
4146 "Ringo", is equal to at least one of the string values of the second **bag**.

4147 • all-of-any

4148 This function applies a boolean function between the elements of two **bags**. The
4149 expression is "True" if and only if the predicate is "True" between each and all of the
4150 elements of the first **bag** collectively against at least one element of the second **bag**.

4151 This function SHALL take three arguments. The first argument SHALL be a <Function>
4152 element that names a boolean function that takes two arguments of primitive types. The
4153 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be
4154 a **bag** of a primitive data-type. The expression SHALL be evaluated as if function named in
4155 the <Function> element were applied between every element in the second argument

4156 and every element of the third argument (the **bag**) using
4157 "urn:oasis:names:tc:xacml:1.0:function:and". The semantics are that the result of the
4158 expression SHALL be "True" if and only if the applied predicate is "True" for each element
4159 of the first **bag** and any element of the second **bag**.

4160 In Haskell, taking advantage of the "any_of" function defined in Haskell above, the
4161 semantics of the "all_of_any" function are as follows:

```
4162 all_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4163 all_of_any f [] ys = "False"
4164 all_of_any f (x:xs) ys = (any_of f x ys) && (all_of_any f xs ys)
```

4165 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4166 element of the list as "x" and the rest of the list as "xs".

4167 For example, the following expression SHALL evaluate to "True":

```
4168 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-any">
4169 <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4170 greater" />
4171 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4172 <AttributeValue
4173 DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4174 <AttributeValue
4175 DataType="http://www.w3.org/2001/XMLSchema#integer">20</AttributeValue>
4176 </Apply>
4177 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4178 <AttributeValue
4179 DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4180 <AttributeValue
4181 DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4182 <AttributeValue
4183 DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4184 <AttributeValue
4185 DataType="http://www.w3.org/2001/XMLSchema#integer">21</AttributeValue>
4186 </Apply>
4187 </Apply>
```

4188 This expression is "True" because all of the elements of the first **bag**, each "10" and "20",
4189 are greater than at least one of the integer values "1", "3", "5", "21" of the second **bag**.

4190 • any-of-all

4191 This function applies a boolean function between the elements of two **bags**. The
4192 expression SHALL be "True" if and only if the predicate is "True" between at least one of
4193 the elements of the first **bag** collectively against all the elements of the second **bag**.

4194 This function SHALL take three arguments. The first argument SHALL be a <Function>
4195 element that names a boolean function that takes two arguments of primitive types. The
4196 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be
4197 a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function
4198 named in the <Function> element were applied between *every* element in the second
4199 argument and *every* element of the third argument (the **bag**) and the results were
4200 combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics are that the
4201 result of the expression SHALL be "True" if and only if the applied predicate is "True" for
4202 any element of the first **bag** compared to *all* the elements of the second **bag**.

4203 In Haskell, taking advantage of the "all_of" function defined in Haskell above, the semantics
4204 of the "any_of_all" function are as follows:

```

4205         any_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4206         any_of_all f []      ys = "False"
4207         any_of_all f (x:xs) ys = (all_of f x ys) || ( any_of_all f xs ys)

```

4208 In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first
4209 element of the list as “x” and the rest of the list as “xs”.

4210 For example, the following expression SHALL evaluate to "True":

```

4211 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-all">
4212   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4213   greater"/>
4214   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4215     <AttributeValue
4216     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4217     <AttributeValue
4218     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4219   </Apply>
4220   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4221     <AttributeValue
4222     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4223     <AttributeValue
4224     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4225     <AttributeValue
4226     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4227     <AttributeValue
4228     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4229   </Apply>
4230 </Apply>

```

4231 This expression is "True" because at least one element of the first **bag**, namely “5”, is
4232 greater than all of the integer values “1”, “2”, “3”, “4” of the second **bag**.

4233 • all-of-all

4234 This function applies a boolean function between the elements of two **bags**. The
4235 expression SHALL be "True" if and only if the predicate is "True" between each and all of
4236 the elements of the first **bag** collectively against all the elements of the second **bag**.

4237 This function SHALL take three arguments. The first argument SHALL be a <Function>
4238 element that names a boolean function that takes two arguments of primitive types. The
4239 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be
4240 a **bag** of a primitive data-type. The expression is evaluated as if the function named in the
4241 <Function> element were applied between *every* element in the second argument and
4242 *every* element of the third argument (the **bag**) and the results were combined using
4243 “urn:oasis:names:tc:xacml:1.0:function:and”. The semantics are that the result of the
4244 expression is "True" if and only if the applied predicate is "True" for *all* elements of the first
4245 **bag** compared to *all* the elements of the second **bag**.

4246 In Haskell, taking advantage of the “all_of” function defined in Haskell above, the semantics
4247 of the “all_of_all” function is as follows:

```

4248         all_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4249         all_of_all f []      ys = "False"
4250         all_of_all f (x:xs) ys = (all_of f x ys) && (all_of_all f xs ys)

```

4251 In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first
4252 element of the list as “x” and the rest of the list as “xs”.

4253 For example, the following expression SHALL evaluate to "True":

```

4254 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-all">
4255   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4256   greater"/>
4257   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4258     <AttributeValue
4259     DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>
4260     <AttributeValue
4261     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4262   </Apply>
4263   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4264     <AttributeValue
4265     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4266     <AttributeValue
4267     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4268     <AttributeValue
4269     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4270     <AttributeValue
4271     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4272   </Apply>
4273 </Apply>

```

4274 This expression is "True" because all elements of the first **bag**, "5" and "6", are each
4275 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4276 • map

4277 This function converts a **bag** of values to another **bag** of values.

4278 This function SHALL take two arguments. The first function SHALL be a <Function>
4279 element naming a function that takes a single argument of a primitive data-type and returns
4280 a value of a primitive data-type. The second argument SHALL be a **bag** of a primitive data-
4281 type. The expression SHALL be evaluated as if the function named in the <Function>
4282 element were applied to each element in the **bag** resulting in a **bag** of the converted value.
4283 The result SHALL be a **bag** of the primitive data-type that is the same data-type that is
4284 returned by the function named in the <Function> element.

4285 In Haskell, this function is defined as follows:

```

4286     map:: (a -> b) -> [a] -> [b]
4287     map f []     = []
4288     map f (x:xs) = (f x) : (map f xs)

```

4289 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4290 element of the list as "x" and the rest of the list as "xs".

4291 For example, the following expression,

```

4292 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:map">
4293   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
4294   normalize-to-lower-case">
4295   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4296     <AttributeValue
4297     DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>
4298     <AttributeValue
4299     DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>
4300   </Apply>
4301 </Apply>

```

4302 evaluates to a **bag** containing "hello" and "world!".

4303 **A14.12 Special match functions**

4304 These functions operate on various types and evaluate to
4305 "http://www.w3.org/2001/XMLSchema#boolean" based on the specified standard matching
4306 algorithm. In an expression that contains any of these functions, if any argument is "Indeterminate",
4307 then the expression SHALL evaluate to "Indeterminate".

4308 • regex-string-match

4309 This function decides a regular expression match. It SHALL take two arguments of
4310 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4311 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4312 expression and the second argument SHALL be a general string. The function
4313 specification SHALL be that of the "http://www.w3.org/TR/xquery-operators#match" function
4314 with the arguments reversed [XF Section 6.3.15.1].

4315 • x500Name-match

4316 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-
4317 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It
4318 shall return "True" if and only if the first argument matches some terminal sequence of
4319 RDNs from the second argument when compared using x500Name-equal.

4320 • rfc822Name-match

4321 This function SHALL take two arguments, the first is of data-type
4322 "http://www.w3.org/2001/XMLSchema#string" and the second is of data-type
4323 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an
4324 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if
4325 the first argument matches the second argument according to the following specification.

4326 An RFC822 name consists of a local-part followed by "@" followed by domain-part. The
4327 local-part is case-sensitive, while the domain-part (which is usually a DNS name) is not
4328 case-sensitive.⁴

4329 The second argument contains a complete rfc822Name. The first argument is a complete
4330 or partial rfc822Name used to select appropriate values in the second argument as follows.

4331 In order to match a particular mailbox in the second argument, the first argument must
4332 specify the complete mail address to be matched. For example, if the first argument is
4333 "Anderson@sun.com", this matches a value in the second argument of
4334 "Anderson@sun.com" and "Anderson@SUN.COM", but not "Anne.Anderson@sun.com",
4335 "anderson@sun.com" or "Anderson@east.sun.com".

4336 In order to match any mail address at a particular domain in the second argument, the first
4337 argument must specify only a domain name (usually a DNS name). For example, if the first
4338 argument is "sun.com", this matches a value in the first argument of "Anderson@sun.com"
4339 or "Baxter@SUN.COM", but not "Anderson@east.sun.com".

4340 In order to match any mail address in a particular domain in the second argument, the first
4341 argument must specify the desired domain-part with a leading ".". For example, if the first
4342 argument is ".east.sun.com", this matches a value in the second argument of

4 According to IETF RFC822 and its successor specifications [RFC2821], case is significant in the *local-part*. Many mail systems, as well as the IETF PKIX specification, treat the *local-part* as case-insensitive. This anomaly is considered an error by mail-system designers and is not encouraged. For this reason, rfc822Name-match treats *local-part* as case sensitive.

4343 "Anderson@east.sun.com" and "anne.anderson@ISRG.EAST.SUN.COM" but not
4344 "Anderson@sun.com".

4345 **A14.13 XPath-based functions**

4346 This section specifies functions that take XPath expressions for arguments. An XPath expression
4347 evaluates to a *node-set*, which is a set of XML nodes that match the expression. A node or node-
4348 set is not in the formal data-type system of XACML. All comparison or other operations on node-
4349 sets are performed in the isolation of the particular function specified. The XPath expressions in
4350 these functions are restricted to the XACML request **context**. The following functions are defined:

- 4351 • xpath-node-count

4352 This function SHALL take an "http://www.w3.org/2001/XMLSchema#string" as an
4353 argument, which SHALL be interpreted as an XPath expression, and evaluates to an
4354 "http://www.w3.org/2001/XMLSchema#integer". The value returned from the function
4355 SHALL be the count of the nodes within the node-set that matches the given XPath
4356 expression.

- 4357 • xpath-node-equal

4358 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,
4359 which SHALL be interpreted as XPath expressions, and SHALL return an
4360 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if any
4361 XML node from the node-set matched by the first argument equals according to the
4362 "op:node-equal" function [XQO] any XML node from the node-set matched by the second
4363 argument.

- 4364 • xpath-node-match

4365 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,
4366 which SHALL be interpreted as XPath expressions and SHALL return an
4367 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL first extend the first
4368 argument to match an XML document in a hierarchical fashion. If *a* is an XPath expression
4369 and it is specified as the first argument, it SHALL be interpreted to mean match the set of
4370 nodes specified by the enhanced XPath expression "*a* | *a*/* | *a*/*@*". In other words, the
4371 expression *a* SHALL match all elements and attributes below the element specified by *a*.
4372 This function SHALL evaluate to "True" if any XML node that matches the enhanced XPath
4373 expression is equal according to "op:node-equal" [XQO] to any XML node from the node-
4374 set matched by the second argument.

4375 **A14.14 Extension functions and primitive types**

4376 Functions and primitive types are specified by string identifiers allowing for the introduction of
4377 functions in addition to those specified by XACML. This approach allows one to extend the XACML
4378 module with special functions and special primitive data-types.

4379 In order to preserve some integrity to the XACML evaluation strategy, the result of all function
4380 applications SHALL depend only on the values of its arguments. Global and hidden parameters
4381 SHALL NOT affect the evaluation of an expression. Functions SHALL NOT have side effects, as
4382 evaluation order cannot be guaranteed in a standard way.

4383 Appendix B. XACML identifiers (normative)

4384 This section defines standard identifiers for commonly used entities. All XACML-defined identifiers
4385 have the common base:

4386 `urn:oasis:names:tc:xacml:1.0`

4387 B.1. XACML namespaces

4388 There are currently two defined XACML namespaces.

4389 Policies are defined using this identifier.

4390 `urn:oasis:names:tc:xacml:1.0:policy`

4391 Request and response **contexts** are defined using this identifier.

4392 `urn:oasis:names:tc:xacml:1.0:context`

4393 B.2. Access subject categories

4394 This identifier indicates the system entity that is directly requesting **access**. That is, the final entity
4395 in a request chain. If **subject** category is not specified, this is the default value.

4396 `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

4397 This identifier indicates the system entity that will receive the results of the request. Used when it is
4398 distinct from the access-subject.

4399 `urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject`

4400 This identifier indicates a system entity through which the **access** request was passed. There may
4401 be more than one. No means is provided to specify the order in which they passed the message.

4402 `urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject`

4403 This identifier indicates a system entity associated with a local or remote codebase that generated
4404 the request. Corresponding **subject attributes** might include the URL from which it was loaded
4405 and/or the identity of the code-signer. There may be more than one. No means is provided to
4406 specify the order they processed the request.

4407 `urn:oasis:names:tc:xacml:1.0:subject-category:codebase`

4408 This identifier indicates a system entity associated with the computer that initiated the **access**
4409 request. An example would be an IPsec identity.

4410 `urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine`

4411 B.3. XACML functions

4412 This identifier is the base for all the identifiers in the table of functions. See Section A.1.

4413 `urn:oasis:names:tc:xacml:1.0:function`

4414 B.4. Data-types

4415 The following identifiers indicate useful data-types.

4416 X.500 distinguished name

4417 urn:oasis:names:tc:xacml:1.0:data-type:x500Name

4418 An x500Name contains an ITU-T Rec. X.520 Distinguished Name. The valid syntax for such a
 4419 name is described in IETF RFC 2253 "Lightweight Directory Access Protocol (v3): UTF-8 String
 4420 Representation of Distinguished Names".

4421 RFC822 Name

4422 urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name

4423 An rfc822Name contains an "e-mail name". The valid syntax for such a name is described in IETF
 4424 RFC 2821, Section 4.1.2, Command Argument Syntax, under the term "Mailbox".

4425 The following data-type identifiers are defined by XML Schema.

4426 <http://www.w3.org/2001/XMLSchema#string>
 4427 <http://www.w3.org/2001/XMLSchema#boolean>
 4428 <http://www.w3.org/2001/XMLSchema#integer>
 4429 <http://www.w3.org/2001/XMLSchema#double>
 4430 <http://www.w3.org/2001/XMLSchema#time>
 4431 <http://www.w3.org/2001/XMLSchema#date>
 4432 <http://www.w3.org/2001/XMLSchema#dateTime>
 4433 <http://www.w3.org/2001/XMLSchema#anyURI>
 4434 <http://www.w3.org/2001/XMLSchema#hexBinary>
 4435 <http://www.w3.org/2001/XMLSchema#base64Binary>

4436 The following data-type identifiers correspond to the dayTimeDuration and yearMonthDuration
 4437 data-types defined in the XQuery specification [XQO Sections 8.2.2 and 8.2.1, respectively].

4438 <http://www.w3.org/2002/08/xquery-functions#dayTimeDuration>
 4439 <http://www.w3.org/2002/08/xquery-functions#yearMonthDuration>

4440 B.5. Subject attributes

4441 These identifiers indicate **attributes** of a **subject**. When used, they SHALL appear within a
 4442 <Subject> element of the request **context**. They SHALL be accessed via a
 4443 <SubjectAttributeDesignator> or an <AttributeSelector> element pointing into a
 4444 <Subject> element of the request **context**.

4445 At most one of each of these attributes is associated with each subject. Each attribute associated
 4446 with authentication included within a single <Subject> element relates to the same authentication
 4447 event.

4448 This identifier indicates the name of the **subject**. The default format is
 4449 <http://www.w3.org/2001/XMLSchema#string>. To indicate other formats, use `DataType` attributes
 4450 listed in B.4

4451 urn:oasis:names:tc:xacml:1.0:subject:subject-id

4452 This identifier indicates the **subject** category. "access-subject" is the default.

4453 urn:oasis:names:tc:xacml:1.0:subject-category

4454 This identifier indicates the security domain of the **subject**. It identifies the administrator and policy
 4455 that manages the name-space in which the **subject** id is administered.

4456 urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier

4457 This identifier indicates a public key used to confirm the **subject's** identity.

4458 urn:oasis:names:tc:xacml:1.0:subject:key-info

4459 This identifier indicates the time at which the **subject** was authenticated.

4460 urn:oasis:names:tc:xacml:1.0:subject:authentication-time

4461 This identifier indicates the method used to authenticate the **subject**.

4462 urn:oasis:names:tc:xacml:1.0:subject:authentication-method

4463 This identifier indicates the time at which the **subject** initiated the **access** request, according to the
4464 **PEP**.

4465 urn:oasis:names:tc:xacml:1.0:subject:request-time

4466 This identifier indicates the time at which the **subject's** current session began, according to the
4467 **PEP**.

4468 urn:oasis:names:tc:xacml:1.0:subject:session-start-time

4469 The following identifiers indicate the location where authentication credentials were activated. They
4470 are intended to support the corresponding entities from the SAML authentication statement.

4471 This identifier indicates that the location is expressed as an IP address.

4472 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address

4473 This identifier indicates that the location is expressed as a DNS name.

4474 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name

4475 Where a suitable attribute is already defined in LDAP [LDAP-1, LDAP-2], the XACML identifier
4476 SHALL be formed by adding the **attribute** name to the URI of the LDAP specification. For
4477 example, the **attribute** name for the userPassword defined in the rfc2256 SHALL be:

4478 http://www.ietf.org/rfc/rfc2256.txt#userPassword

4479 B.6. Resource attributes

4480 These identifiers indicate **attributes** of the **resource**. When used, they SHALL appear within the
4481 <Resource> element of the request **context**. They SHALL be accessed via a
4482 <ResourceAttributeDesignator> or an <AttributeSelector> element pointing into the
4483 <Resource> element of the request **context**.

4484 This identifier indicates the entire URI of the **resource**.

4485 urn:oasis:names:tc:xacml:1.0:resource:resource-id

4486 A **resource attribute** used to indicate values extracted from the **resource**.

4487 urn:oasis:names:tc:xacml:1.0:resource:resource-content

4488 This identifier indicates the last (rightmost) component of the file name. For example, if the URI is:
4489 "file://home/my/status#pointer", the simple-file-name is "status".

4490 urn:oasis:names:tc:xacml:1.0:resource:simple-file-name

4491 This identifier indicates that the **resource** is specified by an XPath expression.

4492 urn:oasis:names:tc:xacml:1.0:resource:xpath

4493 This identifier indicates a UNIX file-system path.

4494 urn:oasis:names:tc:xacml:1.0:resource:ufs-path

4495 This identifier indicates the scope of the **resource**, as described in Section 7.8.

4496 urn:oasis:names:tc:xacml:1.0:resource:scope

4497 The allowed value for this attribute is of data-type http://www.w3.org/2001/XMLSchema#string, and
4498 is either "Immediate", "Children" or "Descendants".

4499 B.7. Action attributes

4500 These identifiers indicate **attributes** of the **action** being requested. When used, they SHALL appear
4501 within the <Action> element of the request **context**. They SHALL be accessed via an
4502 <ActionAttributeDesignator> or an <AttributeSelector> element pointing into the
4503 <Action> element of the request **context**.

4504 urn:oasis:names:tc:xacml:1.0:action:action-id
4505 Action namespace
4506 urn:oasis:names:tc:xacml:1.0:action:action-namespace
4507 Implied action. This is the value for action-id attribute when action is implied.
4508 urn:oasis:names:tc:xacml:1.0:action:implied-action

4509 B.8. Environment attributes

4510 These identifiers indicate *attributes* of the *environment* within which the *decision request* is to be
4511 evaluated. When used in the *decision request*, they SHALL appear in the <Environment>
4512 element of the request *context*. They SHALL be accessed via an
4513 <EnvironmentAttributeDesignator> or an <AttributeSelector> element pointing into
4514 the <Environment> element of the request *context*.

4515 This identifier indicates the current time at the *PDP*. In practice it is the time at which the request
4516 *context* was created.

4517 urn:oasis:names:tc:xacml:1.0:environment:current-time
4518 urn:oasis:names:tc:xacml:1.0:environment:current-date
4519 urn:oasis:names:tc:xacml:1.0:environment:current-dateTime

4520 B.9. Status codes

4521 The following status code identifiers are defined.

4522 This identifier indicates success.

4523 urn:oasis:names:tc:xacml:1.0:status:ok

4524 This identifier indicates that attributes necessary to make a policy decision were not available.

4525 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

4526 This identifier indicates that some attribute value contained a syntax error, such as a letter in a
4527 numeric field.

4528 urn:oasis:names:tc:xacml:1.0:status:syntax-error

4529 This identifier indicates that an error occurred during policy evaluation. An example would be
4530 division by zero.

4531 urn:oasis:names:tc:xacml:1.0:status:processing-error

4532 B.10. Combining algorithms

4533 The deny-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

4534 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

4535 The deny-overrides policy-combining algorithm has the following value for
4536 policyCombiningAlgId:

4537 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides

4538 The permit-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

4539 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides

4540 The permit-overrides policy-combining algorithm has the following value for
4541 policyCombiningAlgId:

4542 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides

4543 The first-applicable rule-combining algorithm has the following value for ruleCombiningAlgId:
4544 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable
4545 The first-applicable policy-combining algorithm has the following value for
4546 policyCombiningAlgId:
4547 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable
4548 The only-one-applicable-policy policy-combining algorithm has the following value for
4549 policyCombiningAlgId:
4550 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable

4551

Appendix C. Combining algorithms (normative)

4552
4553

This section contains a description of the rule-combining and policy-combining algorithms specified by XACML.

4554

C.1. Deny-overrides

4555

The following specification defines the “Deny-overrides” *rule-combining algorithm* of a *policy*.

4556
4557
4558
4559
4560
4561

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".

4562
4563
4564
4565

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.

4566
4567
4568

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".

4569

The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

4570
4571
4572
4573
4574
4575
4576
4577
4578
4579
4580
4581
4582
4583
4584
4585
4586
4587
4588
4589
4590
4591
4592
4593
4594
4595
4596
4597
4598
4599

```
Decision denyOverridesRuleCombiningAlgorithm(Rule rule[])
{
    Boolean atLeastOneError = false;
    Boolean potentialDeny = false;
    Boolean atLeastOnePermit = false;
    for( i=0 ; i < lengthOf(rules) ; i++ )
    {
        Decision decision = evaluate(rule[i]);
        if (decision == Deny)
        {
            return Deny;
        }
        if (decision == Permit)
        {
            atLeastOnePermit = true;
            continue;
        }
        if (decision == NotApplicable)
        {
            continue;
        }
        if (decision == Indeterminate)
        {
            atLeastOneError = true;

            if (effect(rule[i]) == Deny)
            {
                potentialDeny = true;
            }
            continue;
        }
    }
}
```

```

4600     }
4601     }
4602     if (potentialDeny)
4603     {
4604         return Indeterminate;
4605     }
4606     if (atLeastOnePermit)
4607     {
4608         return Permit;
4609     }
4610     if (atLeastOneError)
4611     {
4612         return Indeterminate;
4613     }
4614     return NotApplicable;
4615 }

```

4616 The following specification defines the “Deny-overrides” *policy-combining algorithm* of a *policy*
4617 **set**.

4618 In the entire set of *policies* in the *policy set*, if any *policy* evaluates to "Deny", then the
4619 result of the *policy* combination SHALL be "Deny". In other words, "Deny" takes
4620 precedence, regardless of the result of evaluating any of the other *policies* in the *policy*
4621 **set**. If all *policies* are found to be "NotApplicable" to the *decision request*, then the
4622 *policy set* SHALL evaluate to "NotApplicable".

4623 If an error occurs while evaluating the *target* of a *policy*, or a reference to a *policy* is
4624 considered invalid or the *policy* evaluation results in "Indeterminate", then the *policy set*
4625 SHALL evaluate to "Deny".

4626 The following pseudo-code represents the evaluation strategy of this *policy-combining algorithm*.

```

4627 Decision denyOverridesPolicyCombiningAlgorithm(Policy policy[])
4628 {
4629     Boolean atLeastOnePermit = false;
4630     for( i=0 ; i < lengthOf(policy) ; i++ )
4631     {
4632         Decision decision = evaluate(policy[i]);
4633         if (decision == Deny)
4634         {
4635             return Deny;
4636         }
4637         if (decision == Permit)
4638         {
4639             atLeastOnePermit = true;
4640             continue;
4641         }
4642         if (decision == NotApplicable)
4643         {
4644             continue;
4645         }
4646         if (decision == Indeterminate)
4647         {
4648             return Deny;
4649         }
4650     }
4651     if (atLeastOnePermit)
4652     {
4653         return Permit;
4654     }
4655     return NotApplicable;
4656 }

```

4657 **Obligations** of the individual *policies* shall be combined as described in Section 3.3.2.3.

4658 C.2. Permit-overrides

4659 The following specification defines the "Permit-overrides" *rule-combining algorithm* of a *policy*.

4660 In the entire set of *rules* in the *policy*, if any *rule* evaluates to "Permit", then the result of
4661 the *rule* combination SHALL be "Permit". If any *rule* evaluates to "Deny" and all other
4662 *rules* evaluate to "NotApplicable", then the *policy* SHALL evaluate to "Deny". In other
4663 words, "Permit" takes precedence, regardless of the result of evaluating any of the other
4664 *rules* in the *policy*. If all *rules* are found to be "NotApplicable" to the *decision request*,
4665 then the *policy* SHALL evaluate to "NotApplicable".

4666 If an error occurs while evaluating the *target* or *condition* of a *rule* that contains an *effect*
4667 of "Permit" then the evaluation SHALL continue looking for a result of "Permit". If no other
4668 *rule* evaluates to "Permit", then the *policy* SHALL evaluate to "Indeterminate", with the
4669 appropriate error status.

4670 If at least one *rule* evaluates to "Deny", all other *rules* that do not have evaluation errors
4671 evaluate to "Deny" or "NotApplicable" and all *rules* that do have evaluation errors contain
4672 an *effect* value of "Deny", then the *policy* SHALL evaluate to "Deny".

4673 The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

```
4674 Decision permitOverridesRuleCombiningAlgorithm(Rule rule[])
4675 {
4676     Boolean atLeastOneError = false;
4677     Boolean potentialPermit = false;
4678     Boolean atLeastOneDeny = false;
4679     for( i=0 ; i < lengthOf(rule) ; i++ )
4680     {
4681         Decision decision = evaluate(rule[i]);
4682         if (decision == Deny)
4683         {
4684             atLeastOneDeny = true;
4685             continue;
4686         }
4687         if (decision == Permit)
4688         {
4689             return Permit;
4690         }
4691         if (decision == NotApplicable)
4692         {
4693             continue;
4694         }
4695         if (decision == Indeterminate)
4696         {
4697             atLeastOneError = true;
4698
4699             if (effect(rule[i]) == Permit)
4700             {
4701                 potentialPermit = true;
4702             }
4703             continue;
4704         }
4705     }
4706     if (potentialPermit)
4707     {
4708         return Indeterminate;
4709     }
4710     if (atLeastOneDeny)
4711     {
4712         return Deny;
```

```

4713     }
4714     if (atLeastOneError)
4715     {
4716         return Indeterminate;
4717     }
4718     return NotApplicable;
4719 }

```

4720 The following specification defines the "Permit-overrides" *policy-combining algorithm* of a *policy*
4721 **set**.

4722 In the entire set of *policies* in the *policy set*, if any *policy* evaluates to "Permit", then the
4723 result of the *policy* combination SHALL be "Permit". In other words, "Permit" takes
4724 precedence, regardless of the result of evaluating any of the other *policies* in the *policy*
4725 **set**. If all *policies* are found to be "NotApplicable" to the *decision request*, then the
4726 *policy set* SHALL evaluate to "NotApplicable".

4727 If an error occurs while evaluating the *target* of a *policy*, a reference to a *policy* is
4728 considered invalid or the *policy* evaluation results in "Indeterminate", then the *policy set*
4729 SHALL evaluate to "Indeterminate", with the appropriate error status, provided no other
4730 *policies* evaluate to "Permit" or "Deny".

4731 The following pseudo-code represents the evaluation strategy of this *policy-combining algorithm*.

```

4732 Decision permitOverridesPolicyCombiningAlgorithm(Policy policy[])
4733 {
4734     Boolean atLeastOneError = false;
4735     Boolean atLeastOneDeny = false;
4736     for( i=0 ; i < lengthOf(policy) ; i++ )
4737     {
4738         Decision decision = evaluate(policy[i]);
4739         if (decision == Deny)
4740         {
4741             atLeastOneDeny = true;
4742             continue;
4743         }
4744         if (decision == Permit)
4745         {
4746             return Permit;
4747         }
4748         if (decision == NotApplicable)
4749         {
4750             continue;
4751         }
4752         if (decision == Indeterminate)
4753         {
4754             atLeastOneError = true;
4755             continue;
4756         }
4757     }
4758     if (atLeastOneDeny)
4759     {
4760         return Deny;
4761     }
4762     if (atLeastOneError)
4763     {
4764         return Indeterminate;
4765     }
4766     return NotApplicable;
4767 }

```

4768 **Obligations** of the individual policies shall be combined as described in Section 3.3.2.3.

4769 C.3. First-applicable

4770 The following specification defines the "First-Applicable " **rule-combining algorithm** of a **policy**.

4771 Each **rule** SHALL be evaluated in the order in which it is listed in the **policy**. For a
4772 particular **rule**, if the **target** matches and the **condition** evaluates to "True", then the
4773 evaluation of the **policy** SHALL halt and the corresponding **effect** of the **rule** SHALL be the
4774 result of the evaluation of the **policy** (i.e. "Permit" or "Deny"). For a particular **rule** selected
4775 in the evaluation, if the **target** evaluates to "False" or the **condition** evaluates to "False",
4776 then the next **rule** in the order SHALL be evaluated. If no further **rule** in the order exists,
4777 then the **policy** SHALL evaluate to "NotApplicable".

4778 If an error occurs while evaluating the **target** or **condition** of a **rule**, then the evaluation
4779 SHALL halt, and the **policy** shall evaluate to "Indeterminate", with the appropriate error
4780 status.

4781 The following pseudo-code represents the evaluation strategy of this **rule-combining algorithm**.

```
4782 Decision firstApplicableEffectRuleCombiningAlgorithm(Rule rule[])  
4783 {  
4784     for( i = 0 ; i < lengthOf(rule) ; i++ )  
4785     {  
4786         Decision decision = evaluate(rule[i]);  
4787         if (decision == Deny)  
4788         {  
4789             return Deny;  
4790         }  
4791         if (decision == Permit)  
4792         {  
4793             return Permit;  
4794         }  
4795         if (decision == NotApplicable)  
4796         {  
4797             continue;  
4798         }  
4799         if (decision == Indeterminate)  
4800         {  
4801             return Indeterminate;  
4802         }  
4803     }  
4804     return NotApplicable;  
4805 }
```

4806 The following specification defines the "First-applicable" **policy-combining algorithm** of a **policy**
4807 **set**.

4808 Each **policy** is evaluated in the order that it appears in the **policy set**. For a particular
4809 **policy**, if the **target** evaluates to "True" and the **policy** evaluates to a determinate value of
4810 "Permit" or "Deny", then the evaluation SHALL halt and the **policy set** SHALL evaluate to
4811 the **effect** value of that **policy**. For a particular **policy**, if the **target** evaluate to "False", or
4812 the **policy** evaluates to "NotApplicable", then the next **policy** in the order SHALL be
4813 evaluated. If no further **policy** exists in the order, then the **policy set** SHALL evaluate to
4814 "NotApplicable".

4815 If an error were to occur when evaluating the **target**, or when evaluating a specific **policy**,
4816 the reference to the **policy** is considered invalid, or the **policy** itself evaluates to
4817 "Indeterminate", then the evaluation of the **policy-combining algorithm** shall halt, and the
4818 **policy set** shall evaluate to "Indeterminate" with an appropriate error status.

4819 The following pseudo-code represents the evaluation strategy of this **policy-combination**
4820 **algorithm**.

```
4821 Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy policy[])
4822 {
4823     for( i = 0 ; i < lengthOf(policy) ; i++ )
4824     {
4825         Decision decision = evaluate(policy[i]);
4826         if(decision == Deny)
4827         {
4828             return Deny;
4829         }
4830         if(decision == Permit)
4831         {
4832             return Permit;
4833         }
4834         if (decision == NotApplicable)
4835         {
4836             continue;
4837         }
4838         if (decision == Indeterminate)
4839         {
4840             return Indeterminate;
4841         }
4842     }
4843     return NotApplicable;
4844 }
```

4845 **Obligations** of the individual policies shall be combined as described in Section 3.3.2.3

4846 C.4. Only-one-applicable

4847 The following specification defines the "Only-one-applicable" **policy-combining algorithm** of a
4848 **policy set**.

4849 In the entire set of policies in the **policy set**, if no **policy** is considered applicable by virtue of their
4850 **targets**, then the result of the policy combination algorithm SHALL be "NotApplicable". If more than
4851 one policy is considered applicable by virtue of their **targets**, then the result of the policy
4852 combination algorithm SHALL be "Indeterminate".

4853 If only one **policy** is considered applicable by evaluation of the **policy targets**, then the result of
4854 the **policy-combining algorithm** SHALL be the result of evaluating the **policy**.

4855 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is considered
4856 invalid or the **policy** evaluation results in "Indeterminate", then the **policy set** SHALL evaluate to
4857 "Indeterminate", with the appropriate error status.

4858 The following pseudo-code represents the evaluation strategy of this policy combining algorithm.

```
4859 Decision onlyOneApplicablePolicyPolicyCombiningAlgoGrithm(Policy policy[])
4860 {
4861     Boolean          atLeastOne      = false;
4862     Policy           selectedPolicy = null;
4863     ApplicableResult appResult;
4864
4865     for ( i = 0; i < lengthOf(policy) ; i++ )
4866     {
4867         appResult = isApplicable(policy[i]);
4868
4869         if ( appResult == Indeterminate )
4870         {
```

```
4871     return Indeterminate;
4872 }
4873 if( appResult == Applicable )
4874 {
4875     if ( atLeastOne )
4876     {
4877         return Indeterminate;
4878     }
4879     else
4880     {
4881         atLeastOne      = true;
4882         selectedPolicy = policy[i];
4883     }
4884 }
4885 if ( appResult == NotApplicable )
4886 {
4887     continue;
4888 }
4889 }
4890 if ( atLeastOne )
4891 {
4892     return evaluate(selectedPolicy);
4893 }
4894 else
4895 {
4896     return NotApplicable;
4897 }
4898 }
4899
```

4900

Appendix D. Acknowledgments

4901 The following individuals were voting members of the XACML committee at the time that this
4902 version of the specification was issued:

4903 Anne Anderson, Sun Microsystems, Anne.Anderson@Sun.com
4904 Bill Parducci, Overxeer, bill.parducci@overxeer.com
4905 Carlisle Adams, Entrust, carlisle.adams@entrust.com
4906 Daniel Engovatov, CrossLogix, dengovatov@crosslogix.com
4907 Don Flinn, Quadrasis, Don.Flinn@hitachisoftware.com
4908 Gerald Brose, Xtradyne, Gerald.Brose@xtradyne.com
4909 Hal Lockhart, Entegrity, hal.lockhart@entegrity.com
4910 Ken Yagen, CrossLogix, kyagen@crosslogix.com
4911 Konstantin Beznosov, Quadrasis, konstantin.beznosov@quadrasis.com
4912 Michiharu Kudo, IBM, kudo@jp.ibm.com
4913 Polar Humenn, Self, polar@syr.edu
4914 Simon Godik, Overxeer, simon.godik@overxeer.com
4915 Steve Andersen, OpenNetwork, sanderson@opennetwork.com
4916 Steve Crocker, Pervasive Security Systems, steve.crocker@pervasivesec.com

4917
4918

Appendix E. Tim Moses, Entrust, tim.moses@entrust.com Revision history

Rev	Date	By whom	What
CS V1.0	6 Nov 2002	XACML Technical Committee	First committee specification.
Draft OS V1.0	29 Nov 2002	XACML Technical Committee	Incorporates changes: http://lists.oasis-open.org/archives/xacml/200211/msg00166.html ; 0001, 0002, 0003a, 0003b, 0003c, 0003d, 0003e, 0003f, 0003g, 0003h, 0003i, 0004, 0005, 0006, 0007a, 0007b, 0008a, 0008b, 0009, 0010, 0011b, 0012, 0013, 0014, 0015, 0016, 0017, 0018a, 0018b, 0018c, 0019, 0020, 0021, 0022, 0023, 0024, 0025, 0026, 0027, 0028, 0029, 0030, 0031, 0032a, 0032b, 0032c, 0032d, 0032f, 0034, 0035, 0037, 0038, 0041, 0042, 0043, 0046, 0047, 0049, 0050, 0051, 0053, 0054a, 0054b, 0055, 0056.
Draft OS V1.0	6 Dec 2002	XACML Technical Committee	Incorporates changes: http://lists.oasis-open.org/archives/xacml-comment/200212/msg00036.html ; 0003j, 0011a, 0033, 0036, 0039, 0040, 0044, 0045, 0048, 0052a, 0052b, 0052c, 0052d, 0057, 0058, 0059a, 0059b, 0060, 0061, 0062, 0063a, 0063b, 0064, 0065a, 0065b, 0066, 0067, 0070
Draft OS V1.0	10 Dec 2002	XACML Technical Committee	Incorporates changes: http://lists.oasis-open.org/archives/xacml/200212/msg00069.html ; 0052b, 0069, 0071b, 0073a
Draft OS V1.0	11 Dec 2002	XACML Technical Committee	Incorporates changes: http://lists.oasis-open.org/archives/xacml/200212/msg00076.html ; 0052b (more precise), 0071a, 0072a, 0072b, 0072c, 0072d, 0073a (more complete), 0073b, 0073e, 0073f, 0073g, 0073h

4919

Appendix F. Notices

4920

4921 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
4922 that might be claimed to pertain to the implementation or use of the technology described in this
4923 document or the extent to which any license under such rights might or might not be available;
4924 neither does it represent that it has made any effort to identify any such rights. Information on
4925 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
4926 website. Copies of claims of rights made available for publication and any assurances of licenses to
4927 be made available, or the result of an attempt made to obtain a general license or permission for
4928 the use of such proprietary rights by implementors or users of this specification, can be obtained
4929 from the OASIS Executive Director.

4930 OASIS has been notified of intellectual property rights claimed in regard to some or all of the
4931 contents of this specification. For more information consult the online list of claimed rights.

4932 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
4933 applications, or other proprietary rights which may cover technology that may be required to
4934 implement this specification. Please address the information to the OASIS Executive Director.

4935 Copyright (C) OASIS Open 2002. All Rights Reserved.

4936 This document and translations of it may be copied and furnished to others, and derivative works
4937 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
4938 published and distributed, in whole or in part, without restriction of any kind, provided that the above
4939 copyright notice and this paragraph are included on all such copies and derivative works. However,
4940 this document itself may not be modified in any way, such as by removing the copyright notice or
4941 references to OASIS, except as needed for the purpose of developing OASIS specifications, in
4942 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights
4943 document must be followed, or as required to translate it into languages other than English.

4944 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
4945 successors or assigns.

4946 This document and the information contained herein is provided on an "AS IS" basis and OASIS
4947 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
4948 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY
4949 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
4950 PARTICULAR PURPOSE.