



1

2

3

eXtensible Access Control Markup Language (XACML) Version 1.0

4

OASIS Standard, 18 February 2003

5

Document identifier: oasis-####-xacml-1.0.pdf

6

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

7

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

8

Editors:

9

Simon Godik, Overxeer (simon.godik@overxeer.com)

10

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

11

Committee members:

12

Anne Anderson, Sun Microsystems, Anne.Anderson@Sun.com

13

Bill Parducci, Overxeer, bill.parducci@overxeer.com

14

Carlisle Adams, Entrust, carlisle.adams@entrust.com

15

Don Flinn, Quadrasis, Don.Flinn@hitachisoftware.com

16

Gerald Brose, Xtradyne, Gerald.Brose@xtradyne.com

17

Hal Lockhart, Entegriy, hal.lockhart@entegriy.com

18

Konstantin Beznosov, Quadrasis, konstantin.beznosov@quadrasis.com

19

Michiharu Kudo, IBM, kudo@jp.ibm.com

20

Polar Humenn, Self, polar@syr.edu

21

Simon Godik, Overxeer, simon.godik@overxeer.com

22

Steve Andersen, OpenNetwork, sanderson@opennetwork.com

23

Steve Crocker, Pervasive Security Systems, steve.crocker@pervasivesec.com

24

Tim Moses, Entrust, tim.moses@entrust.com

25

26

Abstract:

27

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

28

language.

29

30

Status:

31

This version of the specification is an OASIS standard.

32

If you are on the xacml@lists.oasis-open.org list for committee members, send comments

33

there. If you are not on that list, subscribe to the xacml-comment@lists.oasis-open.org list

[oasis-####-xacml-1.0.pdf](#)

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

36

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

38 Table of contents

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

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

111	5.36. Element <AttributeAssignment>	63
112	6. Context syntax (normative with the exception of the schema fragments)	64
113	6.1. Element <Request>	64
114	6.2. Element <Subject>	65
115	6.3. Element <Resource>	65
116	6.4. Element <ResourceContent>	66
117	6.5. Element <Action>	66
118	6.6. Element <Environment>	66
119	6.7. Element <Attribute>	67
120	6.8. Element <AttributeValue>	68
121	6.9. Element <Response>	68
122	6.10. Element <Result>	68
123	6.11. Element <Decision>	69
124	6.12. Element <Status>	70
125	6.13. Element <StatusCode>	70
126	6.14. Element <StatusMessage>	70
127	6.15. Element <StatusDetail>	71
128	7. Functional requirements (normative)	71
129	7.1. Policy enforcement point	71
130	7.2. Base policy	72
131	7.3. Target evaluation	72
132	7.4. Condition evaluation	72
133	7.5. Rule evaluation	72
134	7.6. Policy evaluation	73
135	7.7. Policy Set evaluation	74
136	7.8. Hierarchical resources	75
137	7.9. Attributes	75
138	7.9.1. Attribute Matching	76
139	7.9.2. Attribute Retrieval	76
140	7.9.3. Environment Attributes	76
141	7.10. Authorization decision	76
142	7.11. Obligations	77
143	7.12. Unsupported functionality	77
144	7.13. Syntax and type errors	77
145	8. XACML extensibility points (non-normative)	78
146	8.1. Extensible XML attribute types	78
147	8.2. Structured attributes	78

148	9. Security and privacy considerations (non-normative)	79
149	9.1. Threat model	79
150	9.1.1. Unauthorized disclosure	79
151	9.1.2. Message replay	79
152	9.1.3. Message insertion	80
153	9.1.4. Message deletion	80
154	9.1.5. Message modification	80
155	9.1.6. NotApplicable results	80
156	9.1.7. Negative rules	81
157	9.2. Safeguards	81
158	9.2.1. Authentication	81
159	9.2.2. Policy administration	82
160	9.2.3. Confidentiality	82
161	9.2.4. Policy integrity	82
162	9.2.5. Policy identifiers	83
163	9.2.6. Trust model	83
164	9.2.7. Privacy	83
165	10. Conformance (normative)	84
166	10.1. Introduction	84
167	10.2. Conformance tables	84
168	10.2.1. Schema elements	84
169	10.2.2. Identifier Prefixes	85
170	10.2.3. Algorithms	85
171	10.2.4. Status Codes	86
172	10.2.5. Attributes	86
173	10.2.6. Identifiers	86
174	10.2.7. Data-types	87
175	10.2.8. Functions	87
176	11. References	91
177	Appendix A. Standard data-types, functions and their semantics (normative)	93
178	A.1. Introduction	93
179	A.2. Primitive types	93
180	A.3. Structured types	94
181	A.4. Representations	94
182	A.5. Bags	95
183	A.6. Expressions	95
184	A.7. Element <AttributeValue>	96

185	A.8.	Elements <AttributeDesignator> and <AttributeSelector>	96
186	A.9.	Element <Apply>	96
187	A.10.	Element <Condition>	96
188	A.11.	Element <Function>	97
189	A.12.	Matching elements	97
190	A.13.	Arithmetic evaluation	98
191	A.14.	XACML standard functions	99
192	A14.1	Equality predicates	99
193	A14.2	Arithmetic functions	101
194	A14.3	String conversion functions	102
195	A14.4	Numeric data-type conversion functions	102
196	A14.5	Logical functions	102
197	A14.6	Arithmetic comparison functions	103
198	A14.7	Date and time arithmetic functions	104
199	A14.8	Non-numeric comparison functions	105
200	A14.9	Bag functions	107
201	A14.10	Set functions	108
202	A14.11	Higher-order bag functions	109
203	A14.12	Special match functions	116
204	A14.13	XPath-based functions	117
205	A14.14	Extension functions and primitive types	117
206		Appendix B. XACML identifiers (normative)	118
207	B.1.	XACML namespaces	118
208	B.2.	Access subject categories	118
209	B.3.	XACML functions	118
210	B.4.	Data-types	118
211	B.5.	Subject attributes	119
212	B.6.	Resource attributes	120
213	B.7.	Action attributes	120
214	B.8.	Environment attributes	121
215	B.9.	Status codes	121
216	B.10.	Combining algorithms	121
217		Appendix C. Combining algorithms (normative)	123
218	C.1.	Deny-overrides	123
219	C.2.	Permit-overrides	125
220	C.3.	First-applicable	127
221	C.4.	Only-one-applicable	128

222	Appendix D. Acknowledgments	130
223	Appendix E. Revision history	131
224	Appendix F. Notices	132
225		

226 **Errata**

227 Errata can be found at the following location:

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

230 1. Introduction (non-normative)

231 1.1. Glossary

232 1.1.1 Preferred terms

233 **Access** - Performing an *action*

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

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

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

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

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

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

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

246 **Conjunctive sequence** - a sequence of boolean elements combined using the logical 'AND'
247 operation

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

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

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

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

254 **Disjunctive sequence** - a sequence of boolean elements combined using the logical 'OR'
255 operation

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

280 1.1.2 Related terms

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

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

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

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

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

288 1.2. Notation

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

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

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

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

300 Listings of XACML schemas appear like this.

301
302 Example code listings appear like this.

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

- 306 • The prefix `xacml`: stands for the XACML policy namespace.
- 307 • The prefix `xacml-context`: stands for the XACML context namespace.
- 308 • The prefix `ds`: stands for the W3C XML Signature namespace [DS].
- 309 • The prefix `xs`: stands for the W3C XML Schema namespace [XS].
- 310 • The prefix `xf`: stands for the XQuery 1.0 and XPath 2.0 Function and Operators
311 specification namespace [XF].

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

315 1.3. Schema organization and namespaces

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

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

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

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

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

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

323 2. Background (non-normative)

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

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

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

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

350 2.1. Requirements

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

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

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

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

375 2.2. Rule and policy combining

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

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

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

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

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

395 2.3. Combining algorithms

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

- 403 • Deny-overrides,
- 404 • Permit-overrides,
- 405 • First applicable and
- 406 • Only-one-applicable.

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

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

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

420 **2.4. Multiple subjects**

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

427 **2.5. Policies based on subject and resource attributes**

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

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

444 **2.6. Multi-valued attributes**

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

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

453

2.7. Policies based on resource contents

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

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

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

464

2.8. Operators

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

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

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

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

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

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

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

494

2.9. Policy distribution

495 In a distributed system, individual **policy** statements may be written by several policy writers and
496 enforced at several enforcement points. In addition to facilitating the collection and combination of

497 independent **policy** components, this approach allows **policies** to be updated as required. XACML
498 **policy** statements may be distributed in any one of a number of ways. But, XACML does not
499 describe any normative way to do this. Regardless of the means of distribution, **PDPs** are
500 expected to confirm, by examining the **policy's** <Target> element that the policy is applicable to
501 the **decision request** that it is processing.

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

506 2.10. Policy indexing

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

512 Two approaches are supported:

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

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

522 2.11. Abstraction layer

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

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

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

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

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

545 **2.12. Actions performed in conjunction with enforcement**

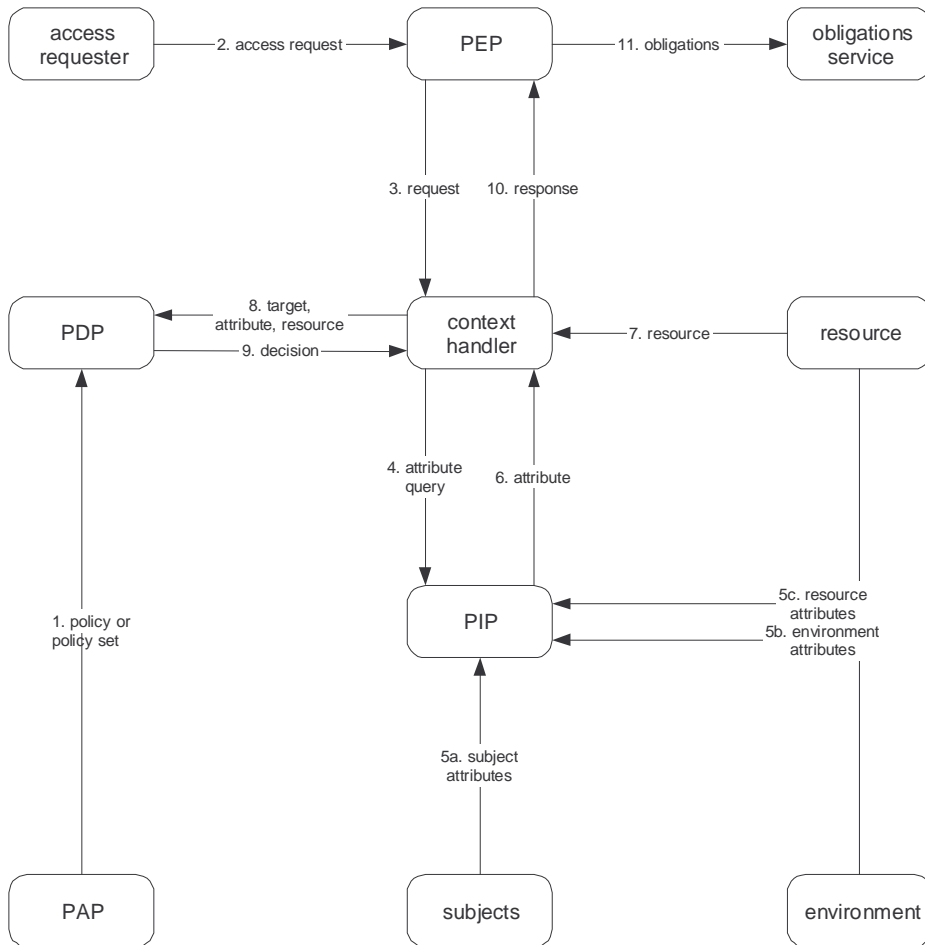
546 In many applications, policies specify actions that **MUST** be performed, either instead of, or in
547 addition to, actions that **MAY** be performed. This idea was described by Sloman [Sloman94].
548 XACML provides facilities to specify actions that **MUST** be performed in conjunction with policy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

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

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

557 **3.1. Data-flow model**

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



559

560

Figure 1 - Data-flow diagram

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

566 The model operates by the following steps.

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

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

570 3. The **PEP** sends the request for **access** to the **context handler** in its native request format,
 571 optionally including **attributes** of the **subjects**, **resource** and **action**. The **context handler**
 572 constructs an XACML request **context** in accordance with steps 4,5,6 and 7.

573 4. **Subject**, **resource** and **environment attributes** may be requested from a **PIP**.

574 5. The **PIP** obtains the requested **attributes**.

575 6. The **PIP** returns the requested **attributes** to the **context handler**.

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

3.2. XACML context

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

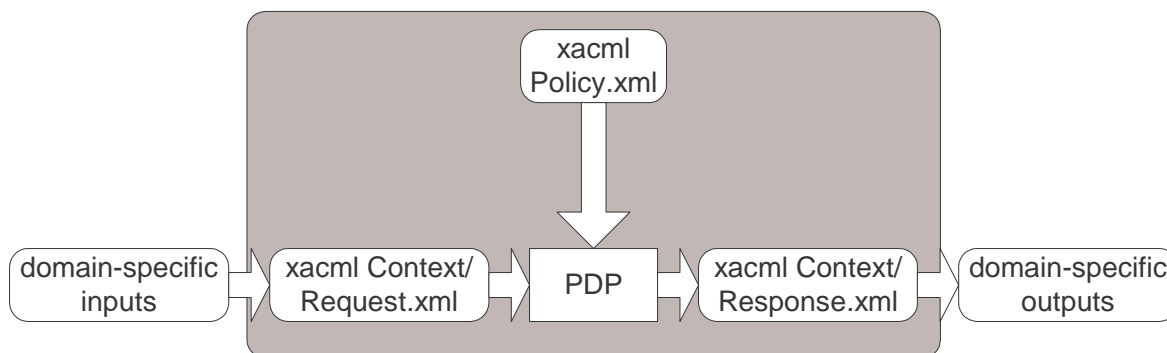


Figure 2 - XACML context

599

600

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

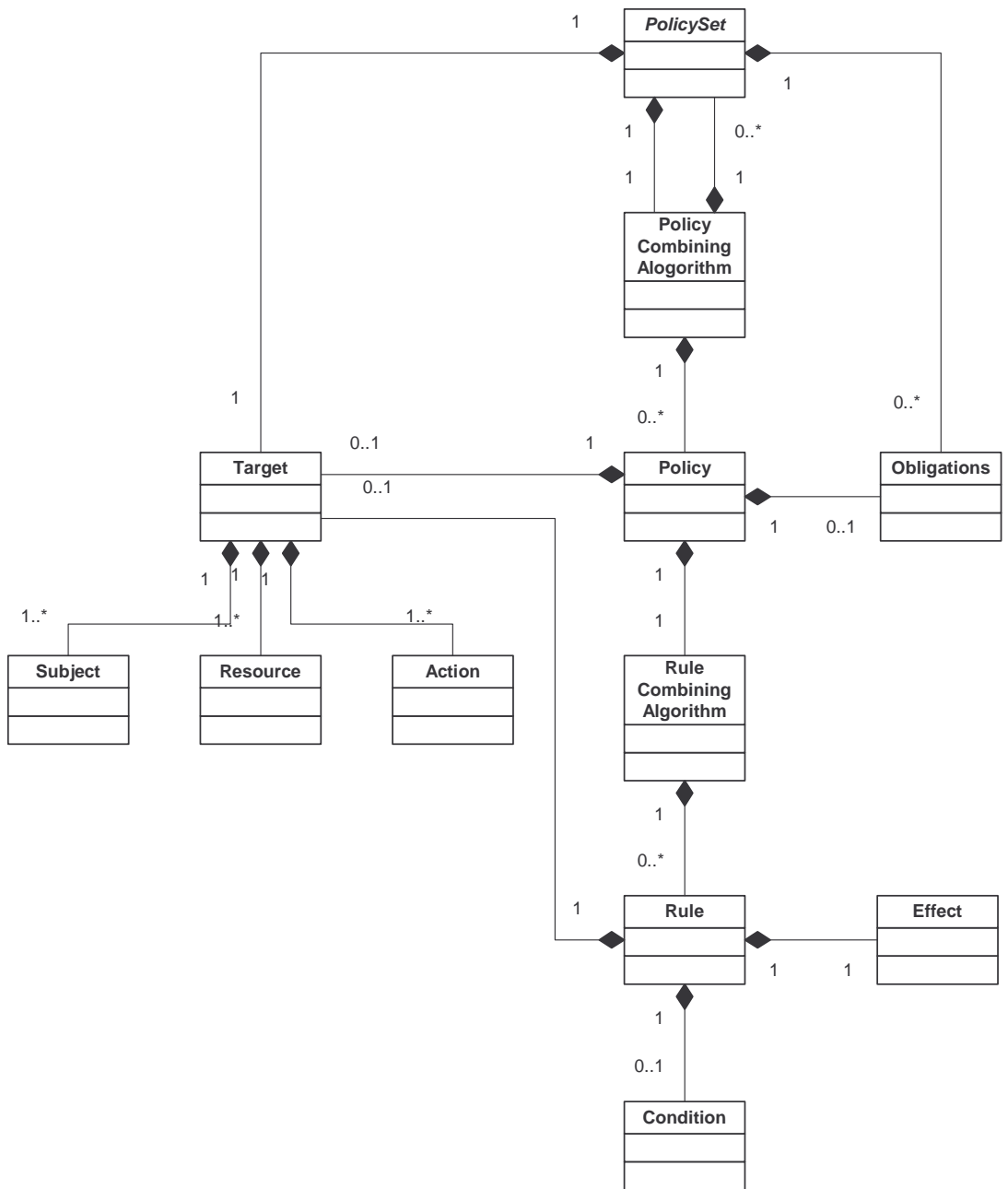
602 See Section 7.9 for a more detailed discussion of the request **context**.

3.3. Policy language model

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

- 605 • **Rule**;
- 606 • **Policy**; and

607 • **Policy set.**
 608 These are described in the following sub-sections.



609

610

Figure 3 - Policy language model

611

3.3.1 Rule

612

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

615

- 616 • a **target**,
 - 617 • an **effect**, and
 - 618 • a **condition**.
- 619 These are discussed in the following sub-sections.

620 3.3.1.1. Rule target

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

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

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

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

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

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

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

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

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

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

658 All three options are supported in XACML.

659 **3.3.1.2. Effect**

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

662 **3.3.1.3. Condition**

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

665 **3.3.2 Policy**

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

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

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

674 **3.3.2.1. Policy target**

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

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

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

696 3.3.2.2. Rule-combining algorithm

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

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

702 3.3.2.3. Obligations

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

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

707 3.3.3 Policy set

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

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

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

715 3.3.3.1. Policy-combining algorithm

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

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

721 3.3.3.2. Obligations

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

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

726

727

4. Examples (non-normative)

728

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

729

730

731

732

4.1. Example one

733

4.1.1 Example policy

734

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

735

736

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

737

738

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

739

740

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">
```

741

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

742

743

[p02] introduces the XACML Policy itself.

744

[p03-p05] are XML namespace declarations.

745

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

746

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

747

748

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

749

750

751

752

753

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

754

[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>
```

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

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

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

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

763 [p24] says what **effect** this **rule** has if the **rule** evaluates to “True”. **Rules** can have an **effect** of
764 either “Permit” or “Deny”. In this case, the rule will evaluate to “Permit”, meaning that, as far as this
765 one **rule** is concerned, the requested **access** should be permitted. If a **rule** evaluates to “False”,
766 then it returns a result of “NotApplicable”. If an error occurs when evaluating the **rule**, the **rule**
767 returns a result of “Indeterminate”. As mentioned above, the **rule-combining algorithm** for the
768 **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>
```

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

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

770 [p29] introduces the **target** of the **rule**. As described above for the **target** of a policy, the **target** of
771 a **rule** describes the **decision requests** to which this **rule** applies. If the **subject, resource** and
772 **action** in a **decision request** do not match the values specified in the **rule target**, then the
773 remainder of the **rule** does not need to be evaluated, and a value of “NotApplicable” is returned to
774 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>
```

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

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

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

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

788 4.1.2 Example request context

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

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

794 In XACML, the information in the **decision request** is formatted into a **request context** statement
795 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">
```

796 [c01-c05] are the header for the **request context**, and are used the same way as the header for the
797 **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>
```

798 The <Subject> element contains one or more **attributes** of the entity making the **access** request.
799 There can be multiple **subjects**, and each **subject** can have multiple **attributes**. In this case, in
800 [c06-c11], there is only one **subject**, and the **subject** has only one **attribute**: the **subject's** identity,
801 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>
```

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

804 per **decision request**. Lines [c13-c16] contain the one **attribute** of the **resource**
805 to which Bart Simpson has requested **access**: the **resource** unix file-system path-
806 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>
```

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

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

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

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

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

821 4.1.3 Example response context

822 As a result, there is no **rule** in this **policy** that returns a "Permit" result for this request. The **rule-**
823 **combining algorithm** for the **policy** specifies that, in this case, a result of "NotApplicable" should
824 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">
```

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

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

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

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

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

831 4.2. Example two

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

835

4.2.1 Example medical record instance

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

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

```

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

```

923 4.2.2 Example request context

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

```

927 [01] <?xml version="1.0" encoding="UTF-8"?>
928 [02] <Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
929 [03] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
930 [04] <Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-
931 category:access-subject">
932 [05]   <Attribute AttributeId=
933 [06]     "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
934 [07]     DataType=
935 [08]       "urn:oasis:names:tc:xacml:1.0:data-type:x500name"
936 [09]     Issuer="www.medico.com"
937 [10]     IssueInstant="2001-12-17T09:30:47-05:00">
938 [11]       <AttributeValue>CN=Julius Hibbert</AttributeValue>
939 [12]     </Attribute>
940 [13]     <Attribute AttributeId=
941 [14]       "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
942 [15]       DataType="http://www.w3.org/2001/XMLSchema#string"
943 [16]       Issuer="www.medico.com"
944 [17]       IssueInstant="2001-12-17T09:30:47-05:00">
945 [18]         <AttributeValue>physician</AttributeValue>
946 [19]       </Attribute>
947 [20]     <Attribute AttributeId=
948 [21]       "urn:oasis:names:tc:xacml:1.0:example:attribute:physician-id"
949 [22]       DataType="http://www.w3.org/2001/XMLSchema#string"
950 [23]       Issuer="www.medico.com"
951 [24]       IssueInstant="2001-12-17T09:30:47-05:00">
952 [25]         <AttributeValue>jh1234</AttributeValue>
953 [26]       </Attribute>
954 [27]     </Subject>
955 [28] </Resource>

```

```

956 [29] <ResourceContent>
957 [30]   <md:record
958 [31]     xmlns:md="//http:www.medico.com/schemas/record.xsd">
959 [32]       <md:patient>
960 [33]         <md:patientDoB>1992-03-21</md:patientDoB>
961 [34]       </md:patient>
962 [35]       <!-- other fields -->
963 [36]     </md:record>
964 [37]   </ResourceContent>
965 [38]   <Attribute AttributeId=
966 [39]     "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
967 [40]     DataType="http://www.w3.org/2001/XMLSchema#string">
968 [41]     <AttributeValue>
969 [42]       //medico.com/records/bart-simpson.xml#
970 [43]       xmlns(md="//http:www.medico.com/schemas/record.xsd)
971 [44]       xpointer(/md:record/md:patient/md:patientDoB)
972 [45]     </AttributeValue>
973 [46]   </Attribute>
974 [47]   <Attribute AttributeId=
975 [48]     "urn:oasis:names:tc:xacml:1.0:resource:xpath"
976 [49]     DataType="http://www.w3.org/2001/XMLSchema#string">
977 [50]     <AttributeValue>
978 [51]       xmlns(md=http:www.medico.com/schemas/record.xsd)
979 [52]       xpointer(/md:record/md:patient/md:patientDoB)
980 [53]     </AttributeValue>
981 [54]   </Attribute>
982 [55]   <Attribute AttributeId=
983 [56]     "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
984 [57]     DataType="http://www.w3.org/2001/XMLSchema#string">
985 [58]     <AttributeValue>
986 [59]       http://www.medico.com/schemas/record.xsd
987 [60]     </AttributeValue>
988 [61]   </Attribute>
989 [62] </Resource>
990 [63] <Action>
991 [64]   <Attribute AttributeId=
992 [65]     "urn:oasis:names:tc:xacml:1.0:action:action-id"
993 [66]     DataType="http://www.w3.org/2001/XMLSchema#string">
994 [67]     <AttributeValue>read</AttributeValue>
995 [68]   </Attribute>
996 [69] </Action>
997 [70] </Request>

```

998 [02]-[03] Standard namespace declarations.

999 [04]-[27] **Subject** attributes are placed in the Subject section of the Request. Each **attribute**
1000 consists of the **attribute** meta-data and the **attribute** value.

1001 [04] Each Subject element has SubjectCategory xml attribute. The value of this attribute
1002 describes the role that the **subject** plays in making the **decision request**. The value of "access-
1003 subject" denotes the identity for which the request was issued.

1004 [05]-[12] **Subject** subject-id **attribute**.

1005 [13]-[19] **Subject** role **attribute**.

1006 [20]-[26] **Subject** physician-id **attribute**.

1007 [28]-[62] **Resource** attributes are placed in the Resource section of the Request. Each **attribute**
1008 consists of **attribute** meta-data and an **attribute** value.

1009 [29]-[36] **Resource** content. The XML document that is being requested is placed here.

- 1010 [38]-[46] **Resource** identifier.
- 1011 [47]-[61] The **Resource** is identified with an Xpointer expression that names the URI of the file that
 1012 is accessed, the target namespace of the document, and the XPath location path to the specific
 1013 element.
- 1014 [47]-[54] The XPath location path in the “resource-id” attribute is extracted and placed in the
 1015 xpath attribute.
- 1016 [55]-[61] **Resource** target-namespace **attribute**.
- 1017 [63]-[69] **Action attributes** are placed in the Action section of the Request.
- 1018 [64]-[68] **Action** identifier.

4.2.3 Example plain-language rules

1019 The following plain-language rules are to be enforced:

1021 Rule 1: A person, identified by his or her patient number, may read any record for which he
 1022 or she is the designated patient.

1023 Rule 2: A person may read any record for which he or she is the designated parent or
 1024 guardian, and for which the patient is under 16 years of age.

1025 Rule 3: A physician may write to any medical element for which he or she is the designated
 1026 primary care physician, provided an email is sent to the patient.

1027 Rule 4: An administrator shall not be permitted to read or write to medical elements of a
 1028 patient record.

1029 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

1032 Rule 1 illustrates a simple **rule** with a single <Condition> element. The following XACML
 1033 <Rule> instance expresses Rule 1:

```

1034 [01] <?xml version="1.0" encoding="UTF-8"?>
1035 [02] <Rule
1036 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1037 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1038 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1039 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1040 [07]   RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"
1041 [08]   Effect="Permit">
1042 [09]   <Description>
1043 [10]     A person may read any medical record in the
1044 [11]     http://www.medico.com/schemas/record.xsd namespace
1045 [12]     for which he or she is a designated patient
1046 [13]   </Description>
1047 [14]   <Target>
1048 [15]     <Subjects>
1049 [16]       <AnySubject/>
1050 [17]     </Subjects>
1051 [18]   <Resources>
1052 [20]     <Resource>
```



```

1053 [21] <!-- match document target namespace -->
1054 [22] <ResourceMatch
1055 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1056 [23] <AttributeValue
1057 DataType="http://www.w3.org/2001/XMLSchema#string">
1058 [24] http://www.medico.com/schemas/record.xsd
1059 [25] </AttributeValue>
1060 [26] <ResourceAttributeDesignator AttributeId=
1061 [27] "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1062 [28] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1063 </ResourceMatch>
1064 [29] <!-- match requested xml element -->
1065 [30] <ResourceMatch
1066 MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1067 [31] <AttributeValue
1068 DataType="http://www.w3.org/2001/XMLSchema#string">/md:record</AttributeV
1069 alue>
1070 [32] <ResourceAttributeDesignator AttributeId=
1071 [33] "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1072 [34] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1073 </ResourceMatch>
1074 [35] </Resource>
1075 [36] </Resources>
1076 [37] <Actions>
1077 [38] <Action>
1078 [39] <ActionMatch
1079 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1080 [40] <AttributeValue
1081 DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1082 [41] <ActionAttributeDesignator AttributeId=
1083 [42] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1084 [43] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1085 </ActionMatch>
1086 [44] </Action>
1087 [45] </Actions>
1088 [46] </Target>
1089 [47] <!-- compare policy number in the document with
1090 [48] policy-number attribute -->
1091 [49] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1092 equal">
1093 [50] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1094 and-only">
1095 [51] <!-- policy-number attribute -->
1096 [52] <SubjectAttributeDesignator AttributeId=
1097 [53] "urn:oasis:names:tc:xacml:1.0:examples:attribute:policy-number"
1098 [54] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1099 </Apply>
1100 [55] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1101 and-only">
1102 [56] <!-- policy number in the document -->
1103 [57] <AttributeSelector RequestContextPath=
1104 [58] "//md:record/md:patient/md:patient-number/text()"
1105 [59] DataType="http://www.w3.org/2001/XMLSchema#string">
1106 </AttributeSelector>
1107 [60] </Apply>
1108 [61] </Condition>
1109 [62] </Rule>

```

1110 [02]-[06]. XML namespace declarations.

1111 [07] **Rule** identifier.

1112 [08]. When a **rule** evaluates to 'True' it emits the value of the `Effect` attribute. This value is
1113 combined with the `Effect` values of other rules according to the **rule-combining algorithm**.

1114 [09]-[13] Free form description of the **rule**.

1115 [14]-[46]. A **rule target** defines a set of **decision requests** that are applicable to the **rule**. A
1116 **decision request**, such that the value of the
1117 "urn:oasis:names:tc:xacml:1.0:resource:target-namespace" **resource attribute** is
1118 equal to "http://www.medico.com/schema/records.xsd" and the value of the
1119 "urn:oasis:names:tc:xacml:1.0:resource:xpath" **resource attribute** matches the XPath
1120 expression "/md:record" and the value of the
1121 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** is equal to "read",
1122 matches the **target** of this **rule**.

1123 [15]-[17]. The `Subjects` element may contain either a **disjunctive sequence** of `Subject`
1124 elements or `AnySubject` element.

1125 [16] The `AnySubject` element is a special element that matches any **subject** in the request
1126 **context**.

1127 [18]-[36]. The `Resources` element may contain either a **disjunctive sequence** of `Resource`
1128 elements or `AnyResource` element.

1129 [20]-[35] The `Resource` element encloses the **conjunctive sequence** of `ResourceMatch`
1130 elements.

1131 [22]-[28] The `ResourceMatch` element compares its first and second child elements according to
1132 the matching function. A match is positive if the value of the first argument matches any of the
1133 values selected by the second argument. This match compares the target namespace of the
1134 requested document with the value of "http://www.medico.com/schema.records.xsd".

1135 [22] The `MatchId` attribute names the matching function.

1136 [23]-[25] Literal attribute value to match.

1137 [26]-[27] The `ResourceAttributeDesignator` element selects the **resource attribute** values
1138 from the request **context**. The **attribute** name is specified by the `AttributeId`. The selection
1139 result is a **bag** of values.

1140 [30]-[34] The `ResourceMatch`. This match compares the results of two XPath expressions. The
1141 first XPath expression is `/md:record` and the second XPath expression is the location path to the
1142 requested xml element. The "xpath-node-match" function evaluates to "True" if the requested XML
1143 element is below the `/md:record` element.

1144 [30] `MatchId` attribute names the matching function.

1145 [31] The literal XPath expression to match. The `md` prefix is resolved using a standard namespace
1146 declaration.

1147 [32]-[33] The `ResourceAttributeDesignator` selects the **bag** of values for the
1148 "urn:oasis:names:tc:xacml:1.0:xpath" **resource attribute**. Here, there is just one
1149 element in the **bag**, which is the location path for the requested XML element.

1150 [37]-[45] The `Actions` element may contain either a **disjunctive sequence** of `Action` elements
1151 or an `AnyAction` element.

1152 [38]-[44] The `Action` element contains a **conjunctive sequence** of `ActionMatch` elements.

1153 [39]-[43] The `ActionMatch` element compares its first and second child elements according to the
1154 matching function. Match is positive if the value of the first argument matches any of the values
1155 selected by the second argument. In this case, the value of the `action-id` action attribute in the
1156 request **context** is compared with the value “read”.

1157 [39] The `MatchId` attribute names the matching function.

1158 [40] The **Attribute** value to match. This is an **action** name.

1159 [41]-[42] The `ActionAttributeDesignator` selects **action attribute** values from the request
1160 **context**. The **attribute** name is specified by the `AttributeId`. The selection result is a **bag** of
1161 values. “urn:oasis:names:tc:xacml:1.0:action:action-id” is the predefined name for
1162 the action identifier.

1163 [49]-[61] The `<Condition>` element. A **condition** must evaluate to “True” for the **rule** to be
1164 applicable. This condition evaluates the truth of the statement: the `patient-number` **subject**
1165 **attribute** is equal to the patient-number in the XML document.

1166 [49] The `FunctionId` attribute of the `<Condition>` element names the function to be used for
1167 comparison. In this case, comparison is done with
1168 urn:oasis:names:tc:xacml:1.0:function:string-equal; this function takes two
1169 arguments of the “http://www.w3.org/2001/XMLSchema#string” data-type.

1170 [50] The first argument to the urn:oasis:names:tc:xacml:1.0:function:string-equal
1171 in the `Condition`. Functions can take other functions as arguments. The `Apply` element
1172 encodes the function call with the `FunctionId` attribute naming the function. Since
1173 urn:oasis:names:tc:xacml:1.0:function:string-equal takes arguments of the
1174 “http://www.w3.org/2001/XMLSchema#string” data-type and
1175 `SubjectAttributeDesignator` selects a **bag** of
1176 “http://www.w3.org/2001/XMLSchema#string” values,
1177 “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used. This
1178 function guarantees that its argument evaluates to a **bag** containing one and only one
1179 “http://www.w3.org/2001/XMLSchema#string” element.

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

1182 [55] The second argument to the “urn:oasis:names:tc:xacml:1.0:function:string-
1183 equal” in the `Condition`. Functions can take other functions as arguments. The `Apply` element
1184 encodes function call with the `FunctionId` attribute naming the function. Since
1185 “urn:oasis:names:tc:xacml:1.0:function:string-equal” takes arguments of the
1186 “http://www.w3.org/2001/XMLSchema#string” data-type and the `AttributeSelector`
1187 selects a **bag** of “http://www.w3.org/2001/XMLSchema#string” values,
1188 “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used. This
1189 function guarantees that its argument evaluates to a **bag** containing one and only one
1190 “http://www.w3.org/2001/XMLSchema#string” element.

1191 [57] The `AttributeSelector` element selects a **bag** of values from the request **context**. The
1192 `AttributeSelector` is a free-form XPath pointing device into the request **context**. The
1193 `RequestContextPath` attribute specifies an XPath expression over the content of the requested
1194 XML document, selecting the policy number. Note that the namespace prefixes in the XPath
1195 expression are resolved with the standard XML namespace declarations.

4.2.4.2. Rule 2

1197 Rule 2 illustrates the use of a mathematical function, i.e. the <Apply> element with functionId
 1198 "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" to calculate date. It also
 1199 illustrates the use of *predicate* expressions, with the functionId
 1200 "urn:oasis:names:tc:xacml:1.0:function:and".

```

1201 [01] <?xml version="1.0" encoding="UTF-8"?>
1202 [02] <Rule
1203 [03] xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1204 [04] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1205 [05] xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1206 [06] xmlns:md="http://www.medico.com/schemas/record.xsd"
1207 [07] RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
1208 [08] Effect="Permit">
1209 [09] <Description>
1210 [10]   A person may read any medical record in the
1211 [11]   http://www.medico.com/records.xsd namespace
1212 [12]   for which he or she is the designated parent or guardian,
1213 [13]   and for which the patient is under 16 years of age
1214 [14] </Description>
1215 [15] <Target>
1216 [16]   <Subjects>
1217 [17]     <AnySubject/>
1218 [18]   </Subjects>
1219 [19]   <Resources>
1220 [20]     <Resource>
1221 [21]       <!-- match document target namespace -->
1222 [22]       <ResourceMatch
1223 [23]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1224 [24]           <AttributeValue
1225 [25]             DataType="http://www.w3.org/2001/XMLSchema#string">
1226 [26]               http://www.medico.com/schemas/record.xsd
1227 [27]             </AttributeValue>
1228 [28]             <ResourceAttributeDesignator AttributeId=
1229 [29]               "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1230 [30]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1231 [31]             </ResourceMatch>
1232 [32]           <!-- match requested xml element -->
1233 [33]           <ResourceMatch
1234 [34]             MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1235 [35]               <AttributeValue
1236 [36]                 DataType="http://www.w3.org/2001/XMLSchema#string">/md:record</AttributeV
1237 [37]                 alue>
1238 [38]               <ResourceAttributeDesignator AttributeId=
1239 [39]                 "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1240 [40]               DataType="http://www.w3.org/2001/XMLSchema#string"/>
1241 [41]             </ResourceMatch>
1242 [42]           </Resource>
1243 [43]     </Resources>
1244 [44]   <Actions>
1245 [45]     <Action>
1246 [46]       <!-- match 'read' action -->
1247 [47]       <ActionMatch
1248 [48]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1249 [49]           <AttributeValue
1250 [50]             DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1251 [51]           <ActionAttributeDesignator AttributeId=
1252 [52]             "urn:oasis:names:tc:xacml:1.0:action:action-id"
1253 [53]           DataType="http://www.w3.org/2001/XMLSchema#string"/>
1254 [54]           </ActionMatch>
1255 [55]         </Action>
1256 [56]       </Actions>

```

```

1257 [47] </Target>
1258 [48] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
1259 [49]   <!-- compare parent-guardian-id subject attribute with
1260 [50]     the value in the document -->
1261 [51]   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1262 [52]     equal">
1263 [53]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1264 [54]     and-only">
1265 [55]       <!-- parent-guardian-id subject attribute -->
1266 [56]       <SubjectAttributeDesignator AttributeId=
1267 [57]         "urn:oasis:names:tc:xacml:1.0:examples:attribute:
1268 [58]         parent-guardian-id"
1269 [59]         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1270 [60]       </Apply>
1271 [61]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1272 [62]     and-only">
1273 [63]       <!-- parent-guardian-id element in the document -->
1274 [64]       <AttributeSelector RequestContextPath=
1275 [65]         "//md:record/md:parentGuardian/md:parentGuardianId/text()"
1276 [66]         DataType="http://www.w3.org/2001/XMLSchema#string">
1277 [67]       </AttributeSelector>
1278 [68]     </Apply>
1279 [69]   </Apply>
1280 [70]   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-or-
1281 [71]     equal">
1282 [72]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-
1283 [73]     and-only">
1284 [74]       <EnvironmentAttributeDesignator AttributeId=
1285 [75]         "urn:oasis:names:tc:xacml:1.0:environment:current-date"
1286 [76]         DataType="http://www.w3.org/2001/XMLSchema#date"/>
1287 [77]       </Apply>
1288 [78]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-
1289 [79]     yearMonthDuration">
1290 [80]       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-
1291 [81]     one-and-only">
1292 [82]       <!-- patient dob recorded in the document -->
1293 [83]       <AttributeSelector RequestContextPath=
1294 [84]         "//md:record/md:patient/md:patientDoB/text()"
1295 [85]         DataType="http://www.w3.org/2001/XMLSchema#date">
1296 [86]       </AttributeSelector>
1297 [87]     </Apply>
1298 [88]   </Apply>
1299 [89]   <AttributeValue DataType="http://www.w3.org/TR/2002/WD-xquery-
1300 [90]     operators-20020816#yearMonthDuration">
1301 [91]     P16Y
1302 [92]   </AttributeValue>
1303 [93] </Apply>
1304 [94] </Apply>
1305 [95] </Condition>
1306 [96] </Rule>

```

1306 [02]-[47] **Rule** declaration and **rule target**. See Rule 1 in Section 4.2.4.1 for the detailed
1307 explanation of these elements.

1308 [48]-[82] The Condition element. **Condition** must evaluate to “True” for the **rule** to be applicable.
1309 This **condition** evaluates the truth of the statement: the requestor is the designated parent or
1310 guardian and the patient is under 16 years of age.

1311 [48] The Condition is using the “urn:oasis:names:tc:xacml:1.0:function:and”
1312 function. This is a boolean function that takes one or more boolean arguments (2 in this case) and
1313 performs the logical “AND” operation to compute the truth value of the expression.

1314 [51]-[65] The truth of the first part of the condition is evaluated: The requestor is the designated
1315 parent or guardian. The Apply element contains a function invocation. The function name is

1316 contained in the `FunctionId` attribute. The comparison is done with
1317 `"urn:oasis:names:tc:xacml:1.0:function:string-equal"` that takes 2 arguments of
1318 `"http://www.w3.org/2001/XMLSchema#string"` data-type.

1319 [52] Since `"urn:oasis:names:tc:xacml:1.0:function:string-equal"` takes arguments
1320 of the `"http://www.w3.org/2001/XMLSchema#string"` data-type,
1321 `"urn:oasis:names:tc:xacml:1.0:function:string-one-and-only"` is used to ensure
1322 that the **subject attribute** `"urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id"` in
1323 the request **context** contains one and only one value.
1324 `"urn:oasis:names:tc:xacml:1.0:function:string-equal"` takes an argument
1325 expression that evaluates to a **bag** of `"http://www.w3.org/2001/XMLSchema#string"`
1326 values.

1327 [54] Value of the **subject attribute**
1328 `"urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id"` is
1329 selected from the request **context** with the `<SubjectAttributeDesignator>` element. This
1330 expression evaluates to a bag of `"http://www.w3.org/2001/XMLSchema#string"` values.

1331 [58] `"urn:oasis:names:tc:xacml:1.0:function:string-one-and-only"` is used to
1332 ensure that the **bag** of values selected by its argument contains one and only one value of data-
1333 type `"http://www.w3.org/2001/XMLSchema#string"`.

1334 [60] The value of the `md:parentGuardianId` element is selected from the **resource** content with
1335 the `AttributeSelector` element. `AttributeSelector` is a free-form XPath expression,
1336 pointing into the request **context**. The `RequestContextPath` XML attribute contains an XPath
1337 expression over the request **context**. Note that all namespace prefixes in the XPath expression
1338 are resolved with standard namespace declarations. The `AttributeSelector` evaluates to the
1339 **bag** of values of data-type `"http://www.w3.org/2001/XMLSchema#string"`.

1340 [66]-[83] The expression: "the patient is under 16 years of age" is evaluated. The patient is under
1341 16 years of age if the current date is less than the date computed by adding 16 to the patient's date
1342 of birth.

1343 [66] `"urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal"` is used to
1344 compute the difference of two dates.

1345 [67] `"urn:oasis:names:tc:xacml:1.0:function:date-one-and-only"` is used to ensure
1346 that the **bag** of values selected by its argument contains one and only one value of data-type
1347 `"http://www.w3.org/2001/XMLSchema#date"`.

1348 [68]-[69] Current date is evaluated by selecting the
1349 `"urn:oasis:names:tc:xacml:1.0:environment:current-date"` **environment attribute**.

1350 [71] `"urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration"` is
1351 used to compute the date by adding 16 to the patient's date of birth. The first argument is a
1352 `"http://www.w3.org/2001/XMLSchema#date"`, and the second argument is an
1353 `"http://www.w3.org/TR/2002/WD-xquery-operators-
1354 20020816#yearMonthDuration"`.

1355 [73] `"urn:oasis:names:tc:xacml:1.0:function:date-one-and-only"` is used to ensure
1356 that the **bag** of values selected by its argument contains one and only one value of data-type
1357 `"http://www.w3.org/2001/XMLSchema#date"`.

1358 [75]-[76] The `<AttributeSelector>` element selects the patient's date of birth by taking the
1359 XPath expression over the document content.

1360 [79]-[81] Year Month Duration of 16 years.

1361

4.2.4.3. Rule 3

1362

Rule 3 illustrates the use of an **obligation**. The XACML <Rule> element syntax does not include

1363

an element suitable for carrying an **obligation**, therefore Rule 3 has to be formatted as a

1364

<Policy> element.

1365

```
[01] <?xml version="1.0" encoding="UTF-8"?>
1366 [02] <Policy
1367 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1368 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1369 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1370 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1371 [07]   PolicyId="urn:oasis:names:tc:xacml:examples:policyid:3"
1372 [08]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
1373 [09]     rule-combining-algorithm:deny-overrides">
1374 [10] <Description>
1375 [11]   Policy for any medical record in the
1376 [12]   http://www.medico.com/schemas/record.xsd namespace
1377 [13] </Description>
1378 [14] <Target>
1379 [15]   <Subjects>
1380 [16]     <AnySubject/>
1381 [17]   </Subjects>
1382 [18]   <Resources>
1383 [19]     <Resource>
1384 [20]       <!-- match document target namespace -->
1385 [21]       <ResourceMatch
1386 [22]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1387 [23]           <AttributeValue
1388 [24]             DataType="http://www.w3.org/2001/XMLSchema#string">
1389 [25]               http://www.medico.com/schemas/record.xsd
1390 [26]             </AttributeValue>
1391 [27]             <ResourceAttributeDesignator AttributeId=
1392 [28]               "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1393 [29]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1394 [30]           </ResourceMatch>
1395 [31]         </Resource>
1396 [32]       </Resources>
1397 [33]     </Subjects>
1398 [34]   <Actions>
1399 [35]     <AnyAction/>
1400 [36]   </Actions>
1401 [37] </Target>
1402 [38] <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:3"
1403 [39]   Effect="Permit">
1404 [40]   <Description>
1405 [41]     A physician may write any medical element in a record
1406 [42]     for which he or she is the designated primary care
1407 [43]     physician, provided an email is sent to the patient
1408 [44]   </Description>
1409 [45]   <Target>
1410 [46]     <Subjects>
1411 [47]       <Subject>
1412 [48]         <!-- match subject group attribute -->
1413 [49]         <SubjectMatch
1414 [50]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1415 [51]             <AttributeValue
1416 [52]               DataType="http://www.w3.org/2001/XMLSchema#string">physician</AttributeVa
1417 [53]             lue>
1418 [54]             <SubjectAttributeDesignator AttributeId=
1419 [55]               "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
1420 [56]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1421 [57]           </SubjectMatch>
1422 [58]         </Subject>
1423 [59]       </Subjects>
1424 [60]     </Target>
1425 [61]   </Rule>
1426 [62] </Policy>
```

```

1422 [51] </Subjects>
1423 [52] <Resources>
1424 [53] <Resource>
1425 [54] <!-- match requested xml element -->
1426 [55] <ResourceMatch
1427 MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1428 [56] <AttributeValue
1429 DataType="http://www.w3.org/2001/XMLSchema#string">
1430 /md:record/md:medical
1431 [57] </AttributeValue>
1432 [58] <ResourceAttributeDesignator AttributeId=
1433 [59] "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1434 [60] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1435 [61] </ResourceMatch>
1436 [62] </Resource>
1437 [63] </Resources>
1438 [64] <Actions>
1439 [65] <Action>
1440 [66] <!-- match action -->
1441 [67] <ActionMatch
1442 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1443 [68] <AttributeValue
1444 [69] DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
1445 [70] <ActionAttributeDesignator AttributeId=
1446 [71] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1447 [72] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1448 [73] </ActionMatch>
1449 [74] </Action>
1450 [75] </Actions>
1451 [76] </Target>
1452 [77] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1453 equal">
1454 [78] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1455 and-only">
1456 [79] <!-- physician-id subject attribute -->
1457 [80] <SubjectAttributeDesignator AttributeId=
1458 [81] "urn:oasis:names:tc:xacml:1.0:example:
1459 [82] attribute:physician-id"
1460 [83] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1461 [84] </Apply>
1462 [85] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1463 and-only">
1464 [86] <AttributeSelector RequestContextPath=
1465 [87] "//md:record/md:primaryCarePhysician/md:registrationID/text()"
1466 [88] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1467 [89] </Apply>
1468 [90] </Condition>
1469 [91] </Rule>
1470 [92] <Obligations>
1471 [93] <!-- send e-mail message to the document owner -->
1472 [94] <Obligation ObligationId=
1473 [95] "urn:oasis:names:tc:xacml:example:obligation:email"
1474 [96] FulfillOn="Permit">
1475 [97] <AttributeAssignment AttributeId=
1476 [98] "urn:oasis:names:tc:xacml:1.0:example:attribute:mailto"
1477 [99] DataType="http://www.w3.org/2001/XMLSchema#string">
1478 [100] <AttributeSelector RequestContextPath=
1479 [101] "//md:/record/md:patient/md:patientContact/md:email"
1480 [102] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1481 [103] </AttributeAssignment>
1482 [104] <AttributeAssignment AttributeId=
1483 [105] "urn:oasis:names:tc:xacml:1.0:example:attribute:text"
1484 [106] DataType="http://www.w3.org/2001/XMLSchema#string">

```



```

1485 [105] <AttributeValue>
1486 [106]     Your medical record has been accessed by:
1487 [107] </AttributeValue>
1488 [108] </AttributeAssignment>
1489 [109] <AttributeAssignment AttributeId=
1490 [110]     "urn:oasis:names:tc:xacml:example:attribute:text"
1491 [111]     DataType="http://www.w3.org/2001/XMLSchema#string">
1492 [112] <SubjectAttributeDesignator AttributeId=
1493 [113]     "urn:osasis:names:tc:xacml:1.0:subject:subject-id"
1494 [114]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1495 [114] </AttributeAssignment>
1496 [115] </Obligation>
1497 [116] </Obligations>
1498 [117] </Policy>

```

1499 [01]-[09] The `Policy` element includes standard namespace declarations as well as policy specific
1500 parameters, such as `PolicyId` and `RuleCombiningAlgId`.

1501 [07] **Policy** identifier. This parameter is used for the inclusion of the `Policy` in the `PolicySet`
1502 element.

1503 [08]-[09] **Rule combining algorithm** identifier. This parameter is used to compute the combined
1504 outcome of **rule effects** for **rules** that are applicable to the **decision request**.

1505 [10-13] Free-form description of the **policy**.

1506 [14]-[33] **Policy target**. The **policy target** defines a set of applicable decision requests. The
1507 structure of the `Target` element in the `Policy` is identical to the structure of the `Target` element
1508 in the `Rule`. In this case, the **policy target** is a set of all XML documents conforming to the
1509 "http://www.medico.com/schemas/record.xsd" target namespace. For the detailed description of
1510 the `Target` element see Rule 1, Section 4.2.4.1.

1511 [34]-[89] The only `Rule` element included in this `Policy`. Two parameters are specified in the **rule**
1512 header: `RuleId` and `Effect`. For the detailed description of the `Rule` structure see Rule 1,
1513 Section 4.2.4.1.

1514 [41]-[74] A **rule target** narrows down a **policy target**. **Decision requests** with the value of
1515 "urn:oasis:names:tc:xacml:1.0:exampe:attribute:role" **subject attribute** equal to
1516 "physician" [42]-[51], and that access elements of the medical record that "xpath-node-match"
1517 the "/md:record/md:medical" XPath expression [52]-[63], and that have the value of the
1518 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** equal to "read".

1519 [65]-[73] match the **target** of this **rule**. For a detailed description of the rule target see example 1,
1520 Section 4.2.4.1.

1521 [75]-[87] The `Condition` element. For the **rule** to be applicable to the authorization request,
1522 **condition** must evaluate to True. This **rule condition** compares the value of the
1523 "urn:oasis:names:tc:xacml:1.0:examples:attribute:physician-id" **subject**
1524 **attribute** with the value of the `physician id` element in the medical record that is being
1525 accessed. For a detailed explanation of rule condition see Rule 1, Section 4.2.4.1.

1526 [90]-[116] The `Obligations` element. **Obligations** are a set of operations that must be
1527 performed by the **PEP** in conjunction with an **authorization decision**. An **obligation** may be
1528 associated with a positive or negative **authorization decision**.

1529 [92]-[115] The `Obligation` element consists of the `ObligationId`, the authorization decision
1530 value for which it must fulfill, and a set of attribute assignments.

1531 [92]-[93] `ObligationId` identifies an **obligation**. **Obligation** names are not interpreted by the
1532 **PDP**.

1533 [94] FulfillOn attribute defines an **authorization decision** value for which this **obligation** must
1534 be fulfilled.

1535 [95]-[101] **Obligation** may have one or more parameters. The **obligation** parameter
1536 “urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto” is assigned the value
1537 from the content of the xml document.

1538 [95-96] AttributeId declares
1539 “urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto” **obligation** parameter.

1540 [97] The **obligation** parameter data-type is defined.

1541 [98]-[100] The **obligation** parameter value is selected from the content of the XML document that is
1542 being accessed with the XPath expression over request **context**.

1543 [102]-[108] The **obligation** parameter
1544 “urn:oasis:names:tc:xacml:1.0:examples:attribute:text” of data-type
1545 “http://www.w3.org/2001/XMLSchema#string” is assigned the literal value “Your
1546 medical record has been accessed by:”

1547 [109]-[114] The **obligation** parameter
1548 “urn:oasis:names:tc:xacml:1.0:examples:attribute:text” of the
1549 “http://www.w3.org/2001/XMLSchema#string” data-type is assigned the value of the
1550 “urn:oasis:names:tc:xacml:1.0:subject:subject-id” **subject attribute**.

1551 4.2.4.4. Rule 4

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

```

1553 [01] <?xml version="1.0" encoding="UTF-8"?>
1554 [02] <Rule
1555 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1556 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1557 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1558 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1559 [07]   RuleId="urn:oasis:names:tc:xacml:example:ruleid:4"
1560 [08]   Effect="Deny">
1561 [09]   <Description>
1562 [10]     An Administrator shall not be permitted to read or write
1563 [11]     medical elements of a patient record in the
1564 [12]     http://www.medico.com/records.xsd namespace.
1565 [13]   </Description>
1566 [14]   <Target>
1567 [15]     <Subjects>
1568 [16]       <Subject>
1569 [17]         <!-- match role subject attribute -->
1570 [18]         <SubjectMatch
1571 [19]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1572 [20]           <AttributeValue
1573 [21]             DataType="http://www.w3.org/2001/XMLSchema#string">administra
1574 [22]             r</AttributeValue>
1575 [23]           <SubjectAttributeDesignator AttributeId=
1576 [24]             "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
1577 [25]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1578 [26]           </SubjectMatch>
1579 [27]         </Subject>
1580 [28]       </Subjects>
1581 [29]     <Resources>
1582 [30]       <Resource>
1583 [31]         <!-- match document target namespace -->

```

```

1584 [28] <ResourceMatch
1585     MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1586 [29]     <AttributeValue
1587         DataType="http://www.w3.org/2001/XMLSchema#string">
1588 [30]         http://www.medico.com/schemas/record.xsd
1589 [31]     </AttributeValue>
1590 [32]     <ResourceAttributeDesignator AttributeId=
1591 [33]         "urn:oasis:names:tc:xacml:1.0:resource:target-namespcae"
1592         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1593 [34]     </ResourceMatch>
1594 [35]     <!-- match requested xml element -->
1595 [36]     <ResourceMatch
1596         MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1597 [37]         <AttributeValue
1598             DataType="http://www.w3.org/2001/XMLSchema#string">
1599 [38]             /md:record/md:medical
1600 [39]         </AttributeValue>
1601 [40]         <ResourceAttributeDesignator AttributeId=
1602 [41]             "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1603             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1604 [42]         </ResourceMatch>
1605 [43]     </Resource>
1606 [44] </Resources>
1607 [45] <Actions>
1608 [46]     <Action>
1609 [47]         <!-- match 'read' action -->
1610 [48]         <ActionMatch
1611             MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1612 [49]             <AttributeValue
1613                 DataType="http://www.w3.org/2001/XMLSchema#string">
1614                 read
1615             </AttributeValue>
1616 [50]             <ActionAttributeDesignator AttributeId=
1617 [51]                 "urn:oasis:names:tc:xacml:1.0:action:action-id"
1618                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1619 [52]             </ActionMatch>
1620 [53]         </Action>
1621 [54]     <Action>
1622 [55]         <!-- match 'write' action -->
1623 [56]         <ActionMatch
1624             MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1625 [57]             <AttributeValue
1626                 DataType="http://www.w3.org/2001/XMLSchema#string">
1627                 write
1628             </AttributeValue>
1629 [58]             <ActionAttributeDesignator AttributeId=
1630 [59]                 "urn:oasis:names:tc:xacml:1.0:action:action-id"
1631                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1632 [60]             </ActionMatch>
1633 [61]         </Action>
1634 [62]     </Actions>
1635 [63] </Target>
1636 [64] </Rule>

```

1637 [01]-[08] The Rule element declaration. The most important parameter here is `Effect`. See Rule
1638 1, Section 4.2.4.1 for a detailed explanation of the Rule structure.

1639 [08] **Rule** `Effect`. Every **rule** that evaluates to “True” emits **rule effect** as its value that will be
1640 combined later on with other **rule effects** according to the **rule combining algorithm**. This **rule**
1641 `Effect` is “Deny” meaning that according to this rule, access must be denied.

1642 [09]-[13] Free form description of the **rule**.

1643 [14]-[63] **Rule target.** The **Rule target** defines a set of **decision requests** that are applicable to
1644 the **rule**. This **rule** is matched by:

- 1645 • a **decision request** with **subject attribute**
1646 "urn:oasis:names:tc:xacml:1.0:examples:attribute:role" equal to
1647 "administrator";
- 1648 • the value of **resource attribute**
1649 "urn:oasis:names:tc:xacml:1.0:resource:target-namespace" is equal to
1650 "http://www.medico.com/schemas/record.xsd"
- 1651 • the value of the requested XML element matches the XPath expression
1652 "/md:record/md:medical";
- 1653 • the value of **action attribute** "urn:oasis:names:tc:xacml:1.0:action:action-id" is equal to
1654 "read"

1655 See Rule 1, Section 4.2.4.1 for the detailed explanation of the `Target` element.

1656 This **rule** does not have a `Condition` element.

1657 **4.2.4.5. Example PolicySet**

1658 This section uses the examples of the previous sections to illustrate the process of combining
1659 **policies**. The policy governing read access to medical elements of a record is formed from each of
1660 the four **rules** described in Section 4.2.3. In plain language, the combined rule is:

- 1661 • Either the requestor is the patient; or
- 1662 • the requestor is the parent or guardian and the patient is under 16; or
- 1663 • the requestor is the primary care physician and a notification is sent to the patient; and
- 1664 • the requestor is not an administrator.

1665 The following XACML `<PolicySet>` illustrates the combined **policies**. **Policy 3** is included by
1666 reference and **policy 2** is explicitly included.

```
1667 [01] <?xml version="1.0" encoding="UTF-8"?>
1668 [02] <PolicySet
1669 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1670 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1671 [05]   PolicySetId=
1672 [06]     "urn:oasis:names:tc:xacml:1.0:examples:policysetid:1"
1673 [07]   PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
1674 [071] policy-combining-algorithm:deny-overrides"/>
1675 [08] <Description>
1676 [09]   Example policy set.
1677 [10] </Description>
1678 [11] <Target>
1679 [12]   <Subjects>
1680 [13]     <Subject>
1681 [14]       <!-- any subject -->
1682 [15]       <AnySubject/>
1683 [16]     </Subject>
1684 [17]   </Subjects>
1685 [18]   <Resources>
1686 [19]     <Resource>
1687 [20]       <!-- any resource in the target namespace -->
1688 [21]       <ResourceMatch
1689       MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
```

```

1690 [22] <AttributeValue
1691      DataType="http://www.w3.org/2001/XMLSchema#string">
1692 [23]      http://www.medico.com/records.xsd
1693 [24] </AttributeValue>
1694 [25] <ResourceAttributeDesignator AttributeId=
1695 [26]      "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1696      DataType="http://www.w3.org/2001/XMLSchema#string"/>
1697 [27] </ResourceMatch>
1698 [28] </Resource>
1699 [29] </Resources>
1700 [30] <Actions>
1701 [31] <Action>
1702 [32] <!-- any action -->
1703 [33] <AnyAction/>
1704 [34] </Action>
1705 [35] </Actions>
1706 [36] </Target>
1707 [37] <!-- include policy from the example 3 by reference -->
1708 [38] <PolicyIdReference>
1709 [39]      urn:oasis:names:tc:xacml:1.0:examples:policyid:3
1710 [40] </PolicyIdReference>
1711 [41] <!-- policy 2 combines rules from the examples 1, 2,
1712 [42]      and 4 is included by value. -->
1713 [43] <Policy
1714 [44]      PolicyId="urn:oasis:names:tc:xacml:examples:policyid:2"
1715 [45]      RuleCombiningAlgId=
1716 [46]      "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
1717      overrides">
1718 [47] <Description>
1719 [48]      Policy for any medical record in the
1720 [49]      http://www.medico.com/schemas/record.xsd namespace
1721 [50] </Description>
1722 [51] <Target> ... </Target>
1723 [52] <Rule
1724 [53]      RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"
1725 [54]      Effect="Permit"> ... </Rule>
1726 [55] <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
1727 [56]      Effect="Permit"> ... </Rule>
1728 [57] <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:4"
1729 [58]      Effect="Deny"> ... </Rule>
1730 [59] <Obligations> ... </Obligations>
1731 [60] </Policy>
1732 [61] </PolicySet>
1733

```

1734 [02]-[07] PolicySet declaration. Standard XML namespace declarations are included as well as
1735 PolicySetId, and **policy combining algorithm** identifier.

1736 [05]-[06] PolicySetId is used for identifying this **policy set** and for possible inclusion of this
1737 **policy set** into another **policy set**.

1738 [07] **Policy combining algorithm** identifier. Policies in the **policy set** are combined according to
1739 the specified **policy combining algorithm** identifier when the **authorization decision** is
1740 computed.

1741 [08]-[10] Free form description of the **policy set**.

1742 [11]-[36] PolicySet Target element defines a set of **decision requests** that are applicable to
1743 this PolicySet.

1744 [38]-[40] PolicyIdReference includes **policy** by id.

1745 [43]-[60] **Policy 2** is explicitly included in this **policy set**.

1746 5. Policy syntax (normative, with the exception of 1747 the schema fragments)

1748 5.1. Element <PolicySet>

1749 The <PolicySet> element is a top-level element in the XACML policy schema. <PolicySet> is
1750 an aggregation of other *policy sets* and *policies*. *Policy sets* MAY be included in an enclosing
1751 <PolicySet> element either directly using the <PolicySet> element or indirectly using the
1752 <PolicySetIdReference> element. *Policies* MAY be included in an enclosing <PolicySet>
1753 element either directly using the <Policy> element or indirectly using the <PolicyIdReference>
1754 element.

1755 If a <PolicySet> element contains references to other *policy sets* or *policies* in the form of
1756 URLs, then these references MAY be resolvable.

1757 *Policies* included in the <PolicySet> element MUST be combined by the algorithm specified by
1758 the PolicyCombiningAlgId attribute.

1759 The <Target> element defines the applicability of the <PolicySet> to a set of *decision*
1760 *requests*. If the <Target> element within <PolicySet> matches the *request context*, then the
1761 <PolicySet> element MAY be used by the *PDP* in making its *authorization decision*.

1762 The <Obligations> element contains a set of *obligations* that MUST be fulfilled by the *PEP* in
1763 conjunction with the *authorization decision*. If the *PEP* does not understand any of the
1764 *obligations*, then it MUST act as if the *PDP* had returned a "Deny" *authorization decision* value.

```
1765 <xs:element name="PolicySet" type="xacml:PolicySetType"/>  
1766 <xs:complexType name="PolicySetType">  
1767   <xs:sequence>  
1768     <xs:element ref="xacml:Description" minOccurs="0"/>  
1769     <xs:element ref="xacml:PolicySetDefaults" minOccurs="0"/>  
1770     <xs:element ref="xacml:Target"/>  
1771     <xs:choice minOccurs="0" maxOccurs="unbounded">  
1772       <xs:element ref="xacml:PolicySet"/>  
1773       <xs:element ref="xacml:Policy"/>  
1774       <xs:element ref="xacml:PolicySetIdReference"/>  
1775       <xs:element ref="xacml:PolicyIdReference"/>  
1776     </xs:choice>  
1777     <xs:element ref="xacml:Obligations" minOccurs="0"/>  
1778   </xs:sequence>  
1779   <xs:attribute name="PolicySetId" type="xs:anyURI" use="required"/>  
1780   <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI"  
1781   use="required"/>  
1782 </xs:complexType>
```

1783 The <PolicySet> element is of **PolicySetType** complex type.

1784 The <PolicySet> element contains the following attributes and elements:

1785 PolicySetId [Required]

1786 *Policy set* identifier. It is the responsibility of the *PAP* to ensure that no two *policies*
1787 visible to the *PDP* have the same identifier. This MAY be achieved by following a
1788 predefined URN or URI scheme. If the *policy set* identifier is in the form of a URL, then it
1789 MAY be resolvable.

1790

1791 PolicyCombiningAlgId [Required]

1792 The identifier of the *policy-combining algorithm* by which the <PolicySet>
 1793 components MUST be combined. Standard *policy-combining algorithms* are listed in
 1794 Appendix C. Standard *policy-combining algorithm* identifiers are listed in Section B.10.

1795 <Description> [Optional]

1796 A free-form description of the <PolicySet>.

1797 <PolicySetDefaults> [Optional]

1798 A set of default values applicable to the <PolicySet>. The scope of the
 1799 <PolicySetDefaults> element SHALL be the enclosing *policy set*.

1800 <Target> [Required]

1801 The <Target> element defines the applicability of a <PolicySet> to a set of *decision*
 1802 *requests*.

1803 The <Target> element MAY be declared by the creator of the <PolicySet> or it MAY be
 1804 computed from the <Target> elements of the referenced <Policy> elements, either as
 1805 an intersection or as a union.

1806 <PolicySet> [Any Number]

1807 A *policy set* component that is included in this *policy set*.

1808 <Policy> [Any Number]

1809 A *policy* component that is included in this *policy set*.

1810 <PolicySetIdReference> [Any Number]

1811 A reference to a <PolicySet> component that MUST be included in this *policy set*. If
 1812 <PolicySetIdReference> is a URL, then it MAY be resolvable.

1813 <PolicyIdReference> [Any Number]

1814 A reference to a <Policy> component that MUST be included in this *policy set*. If the
 1815 <PolicyIdReference> is a URL, then it MAY be resolvable.

1816 <Obligations> [Optional]

1817 Contains the set of <Obligation> elements. See Section 7.11 for a description of how
 1818 the set of *obligations* to be returned by the *PDP* shall be determined.

1819 5.2. Element <Description>

1820 The <Description> element is used for a free-form description of the <PolicySet> element,
 1821 <Policy> element and <Rule> element. The <Description> element is of **xs:string** simple
 1822 type.

1823

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

1824 5.3. Element <PolicySetDefaults>

1825 The <PolicySetDefaults> element SHALL specify default values that apply to the
 1826 <PolicySet> element.

```

1827 <xs:element name="PolicySetDefaults" type="xacml:DefaultsType"/>
1828 <xs:complexType name="DefaultsType">
1829   <xs:sequence>
1830     <xs:choice>
1831       <xs:element ref="xacml:XPathVersion" minOccurs="0"/>
1832     </xs:choice>
1833   </xs:sequence>
1834 </xs:complexType>

```

1835 <PolicySetDefaults> element is of **DefaultsType** complex type.

1836 The <PolicySetDefaults> element contains the following elements:

1837 <XPathVersion> [Optional]

1838 Default XPath version.

1839 5.4. Element <XPathVersion>

1840 The <XPathVersion> element SHALL specify the version of the XPath specification to be used by
1841 <AttributeSelector> elements.

```

1842 <xs:element name="XPathVersion" type="xs:anyURI"/>

```

1843 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**
1844 or **policy** contains <AttributeSelector> elements.

1846 5.5. Element <Target>

1847 The <Target> element identifies the set of **decision requests** that the parent element is intended
1848 to evaluate. The <Target> element SHALL appear as a child of <PolicySet>, <Policy> and
1849 <Rule> elements. It contains definitions for **subjects**, **resources** and **actions**.

1850 The <Target> element SHALL contain a **conjunctive sequence** of <Subjects>, <Resources>
1851 and <Actions> elements. For the parent of the <Target> element to be applicable to the
1852 **decision request**, there MUST be at least one positive match between each section of the
1853 <Target> element and the corresponding section of the <xacml-context:Request> element.

```

1854 <xs:element name="Target" type="xacml:TargetType"/>
1855 <xs:complexType name="TargetType">
1856   <xs:sequence>
1857     <xs:element ref="xacml:Subjects"/>
1858     <xs:element ref="xacml:Resources"/>
1859     <xs:element ref="xacml:Actions"/>
1860   </xs:sequence>
1861 </xs:complexType>

```

1862 The <Target> element is of **TargetType** complex type.

1863 The <Target> element contains the following elements:

1864 <Subjects> [Required]

1865 Matching specification for the **subject attributes** in the **context**.

1866 <Resources> [Required]

1867 Matching specification for the **resource attributes** in the **context**.

1868

1869 <Actions> [Required]

1870 Matching specification for the *action attributes* in the *context*.

1871 5.6. Element <Subjects>

1872 The <Subjects> element SHALL contain a *disjunctive sequence* of <Subject> elements.

```
1873 <xs:element name="Subjects" type="xacml:SubjectsType"/>
1874 <xs:complexType name="SubjectsType">
1875   <xs:choice>
1876     <xs:element ref="xacml:Subject" maxOccurs="unbounded"/>
1877     <xs:element ref="xacml:AnySubject"/>
1878   </xs:choice>
1879 </xs:complexType>
```

1880 The <Subjects> element is of **SubjectsType** complex type.

1881 The <Subjects> element contains the following elements:

1882 <Subject> [One To Many, Required Choice]

1883 See Section 5.7.

1884 <AnySubject> [Required Choice]

1885 See Section 5.8.

1886 5.7. Element <Subject>

1887 The <Subject> element SHALL contain a *conjunctive sequence* of <SubjectMatch>
1888 elements.

```
1889 <xs:element name="Subject" type="xacml:SubjectType"/>
1890 <xs:complexType name="SubjectType">
1891   <xs:sequence>
1892     <xs:element ref="xacml:SubjectMatch" maxOccurs="unbounded"/>
1893   </xs:sequence>
1894 </xs:complexType>
```

1895 The <Subject> element is of **SubjectType** complex type.

1896 The <Subject> element contains the following elements:

1897 <SubjectMatch> [One to Many]

1898 A *conjunctive sequence* of individual matches of the *subject attributes* in the *context*
1899 and the embedded *attribute* values.

1900 5.8. Element <AnySubject>

1901 The <AnySubject> element SHALL match any *subject attribute* in the *context*.

```
1902 <xs:element name="AnySubject"/>
```

1903 5.9. Element <SubjectMatch>

1904 The <SubjectMatch> element SHALL identify a set of *subject*-related entities by matching
1905 *attribute* values in a <xacml-context:Subject> element of the *context* with the embedded
1906 *attribute* value.

```

1907 <xs:element name="SubjectMatch" type="xacml:SubjectMatchType" />
1908 <xs:complexType name="SubjectMatchType">
1909   <xs:sequence>
1910     <xs:element ref="xacml:AttributeValue" />
1911     <xs:choice>
1912       <xs:element ref="xacml:SubjectAttributeDesignator" />
1913       <xs:element ref="xacml:AttributeSelector" />
1914     </xs:choice>
1915   </xs:sequence>
1916   <xs:attribute name="MatchId" type="xs:anyURI" use="required" />
1917 </xs:complexType>

```

1918 The <SubjectMatch> element is of **SubjectMatchType** complex type.

1919 The <SubjectMatch> element contains the following attributes and elements:

1920 MatchId [Required]

1921 Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI** with
1922 legal values documented in Section A.12.

1923 <AttributeValue> [Required]

1924 Embedded **attribute** value.

1925 <SubjectAttributeDesignator> [Required choice]

1926 Identifies one or more **attribute** values in a <Subject> element of the **context**.

1927 <AttributeSelector> [Required choice]

1928 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
1929 expression SHOULD resolve to an **attribute** in a <Subject> element of the **context**.

1930 **5.10. Element <Resources>**

1931 The <Resources> element SHALL contain a **disjunctive sequence** of <Resource> elements.

```

1932 <xs:element name="Resources" type="xacml:ResourcesType" />
1933 <xs:complexType name="ResourcesType">
1934   <xs:choice>
1935     <xs:element ref="xacml:Resource" maxOccurs="unbounded" />
1936     <xs:element ref="xacml:AnyResource" />
1937   </xs:choice>
1938 </xs:complexType>

```

1939 The <Resources> element is of **ResourcesType** complex type.

1940 The <Resources> element contains the following elements:

1941 <Resource> [One To Many, Required Choice]

1942 See Section 5.11.

1943 <AnyResource> [Required Choice]

1944 See Section 5.12.

1945 **5.11. Element <Resource>**

1946 The <Resource> element SHALL contain a **conjunctive sequence** of <ResourceMatch>
1947 elements.

```
1948 <xs:element name="Resource" type="xacml:ResourceType" />
1949 <xs:complexType name="ResourceType">
1950 <xs:sequence>
1951 <xs:element ref="xacml:ResourceMatch" maxOccurs="unbounded" />
1952 </xs:sequence>
1953 </xs:complexType>
```

1954 The <Resource> element is of **ResourceType** complex type.

1955 The <Resource> element contains the following elements:

1956 <ResourceMatch> [One to Many]

1957 A **conjunctive sequence** of individual matches of the **resource attributes** in the **context**
1958 and the embedded **attribute** values.

1959 5.12. Element <AnyResource>

1960 The <AnyResource> element SHALL match any **resource attribute** in the **context**.

```
1961 <xs:element name="AnyResource" />
```

1962 5.13. Element <ResourceMatch>

1963 The <ResourceMatch> element SHALL identify a set of **resource**-related entities by matching
1964 **attribute** values in the <xacml-context:Resource> element of the **context** with the embedded
1965 **attribute** value.

```
1966 <xs:element name="ResourceMatch" type="xacml:ResourceMatchType" />
1967 <xs:complexType name="ResourceMatchType">
1968 <xs:sequence>
1969 <xs:element ref="xacml:AttributeValue" />
1970 <xs:choice>
1971 <xs:element ref="xacml:ResourceAttributeDesignator" />
1972 <xs:element ref="xacml:AttributeSelector" />
1973 </xs:choice>
1974 </xs:sequence>
1975 <xs:attribute name="MatchId" type="xs:anyMatch" use="required" />
1976 </xs:complexType>
```

1977 The <ResourceMatch> element is of **ResourceMatchType** complex type.

1978 The <ResourceMatch> element contains the following attributes and elements:

1979 MatchId [Required]

1980 Specifies a matching function. Values of this attribute MUST be of type **xs:anyURI**, with
1981 legal values documented in Section A.12.

1982 <AttributeValue> [Required]

1983 Embedded **attribute** value.

1984 <ResourceAttributeDesignator> [Required Choice]

1985 Identifies one or more **attribute** values in the <Resource> element of the **context**.

1986 <AttributeSelector> [Required Choice]

1987 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
1988 expression SHOULD resolve to an **attribute** in the <Resource> element of the **context**.

1989

5.14. Element <Actions>

1990

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

1991

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

1992

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

1993

```
<xs:choice>
```

1994

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

1995

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

1996

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

1997

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

1998

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

1999

The <Actions> element contains the following elements:

2000

<Action> [One To Many, Required Choice]

2001

See Section 5.15.

2002

<AnyAction> [Required Choice]

2003

See Section 5.16.

2004

5.15. Element <Action>

2005

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

2006

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

2007

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

2008

```
<xs:sequence>
```

2009

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

2010

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

2011

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

2012

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

2013

The <Action> element contains the following elements:

2014

<ActionMatch> [One to Many]

2015

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

2016

2017

5.16. Element <AnyAction>

2018

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

2019

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

2020

2021

5.17. Element <ActionMatch>

2022

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

2023

2024

2025

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

2026

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

2027

```
<xs:sequence>
```

2028

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

```

2029     <xs:choice>
2030         <xs:element ref="xacml:ActionAttributeDesignator" />
2031         <xs:element ref="xacml:AttributeSelector" />
2032     </xs:choice>
2033 </xs:sequence>
2034 <xs:attribute name="MatchId" type="xs:anyURI" use="required" />
2035 </xs:complexType>

```

2036 The <ActionMatch> element is of **ActionMatchType** complex type.

2037 The <ActionMatch> element contains the following attributes and elements:

2038 MatchId [Required]

2039 Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI**, with
 2040 legal values documented in Section A.12.

2041 <AttributeValue> [Required]

2042 Embedded **attribute** value.

2043 <ActionAttributeDesignator> [Required Choice]

2044 Identifies one or more **attribute** values in the <Action> element of the **context**.

2045 <AttributeSelector> [Required Choice]

2046 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
 2047 expression SHOULD resolve to an **attribute** in the <Action> element of the **context**.

2048 **5.18. Element <PolicySetIdReference>**

2049 The <PolicySetIdReference> element SHALL be used to reference a <PolicySet> element
 2050 by id. If <PolicySetIdReference> is a URL, then it MAY be resolvable to the <PolicySet>.
 2051 The mechanism for resolving a **policy set** reference to the corresponding **policy set** is outside the
 2052 scope of this specification.

```

2053 <xs:element name="PolicySetIdReference" type="xs:anyURI" />

```

2054 Element <PolicySetIdReference> is of **xs:anyURI** simple type.

2055 **5.19. Element <PolicyIdReference>**

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

```

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

```

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

2062 **5.20. Element <Policy>**

2063 The <Policy> element is the smallest entity that SHALL be presented to the **PDP** for evaluation.

2064 The main components of this element are the <Target>, <Rule> and <Obligations> elements
 2065 and the RuleCombiningAlgId attribute.

2066 The <Target> element SHALL define the applicability of the <Policy> to a set of **decision**
2067 **requests**.

2068 **Rules** included in the <Policy> element MUST be combined by the algorithm specified by the
2069 RuleCombiningAlgId attribute.

2070 The <Obligations> element SHALL contain a set of **obligations** that MUST be fulfilled by the
2071 **PDP** in conjunction with the **authorization decision**.

```
2072 <xs:element name="Policy" type="xacml:PolicyType"/>
2073 <xs:complexType name="PolicyType">
2074   <xs:sequence>
2075     <xs:element ref="xacml:Description" minOccurs="0"/>
2076     <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>
2077     <xs:element ref="xacml:Target"/>
2078     <xs:element ref="xacml:Rule" minOccurs="0" maxOccurs="unbounded"/>
2079     <xs:element ref="xacml:Obligations" minOccurs="0"/>
2080   </xs:sequence>
2081   <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
2082   <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>
2083 </xs:complexType>
```

2084 The <Policy> element is of **PolicyType** complex type.

2085 The <Policy> element contains the following attributes and elements:

2086 PolicyId [Required]

2087 **Policy** identifier. It is the responsibility of the **PAP** to ensure that no two **policies** visible to
2088 the **PDP** have the same identifier. This MAY be achieved by following a predefined URN or
2089 URI scheme. If the **policy** identifier is in the form of a URL, then it MAY be resolvable.

2090 RuleCombiningAlgId [Required]

2091 The identifier of the rule-combining algorithm by which the <Policy> components MUST
2092 be combined. Standard rule-combining algorithms are listed in Appendix C. Standard rule-
2093 combining algorithm identifiers are listed in Section B.10.

2094 <Description> [Optional]

2095 A free-form description of the **policy**. See Section 5.2 Element <Description>.

2096 <PolicyDefaults> [Optional]

2097 Defines a set of default values applicable to the **policy**. The scope of the
2098 <PolicyDefaults> element SHALL be the enclosing policy.

2099 <Target> [Required]

2100 The <Target> element SHALL define the applicability of a <Policy> to a set of **decision**
2101 **requests**.

2102 The <Target> element MAY be declared by the creator of the <Policy> element, or it
2103 MAY be computed from the <Target> elements of the referenced <Rule> elements either
2104 as an intersection or as a union.

2105 <Rule> [Any Number]

2106 A sequence of authorizations that MUST be combined according to the
2107 RuleCombiningAlgId attribute. **Rules** whose <Target> elements match the **decision**
2108 **request** MUST be considered. **Rules** whose <Target> elements do not match the
2109 **decision request** SHALL be ignored.

2110 <Obligations> [Optional]

2111 A **conjunctive sequence** of **obligations** that MUST be fulfilled by the **PEP** in conjunction
2112 with the **authorization decision**. See Section 7.11 for a description of how the set of
2113 **obligations** to be returned by the **PDP** SHALL be determined.

2114 5.21. Element <PolicyDefaults>

2115 The <PolicyDefaults> element SHALL specify default values that apply to the <Policy>
2116 element.

```
2117 <xs:element name="PolicyDefaults" type="xacml:DefaultsType" />  
2118 <xs:complexType name="DefaultsType">  
2119 <xs:sequence>  
2120 <xs:choice>  
2121 <xs:element ref="xacml:XPathVersion" minOccurs="0" />  
2122 </xs:choice>  
2123 </xs:sequence>  
2124 </xs:complexType>
```

2125 <PolicyDefaults> element is of **DefaultsType** complex type.

2126 The <PolicyDefaults> element contains the following elements:

2127 <XPathVersion> [Optional]

2128 Default XPath version.

2129 5.22. Element <Rule>

2130 The <Rule> element SHALL define the individual **rules** in the **policy**. The main components of
2131 this element are the <Target> and <Condition> elements and the Effect attribute.

```
2132 <xs:element name="Rule" type="xacml:RuleType" />  
2133 <xs:complexType name="RuleType">  
2134 <xs:sequence>  
2135 <xs:element ref="xacml:Description" minOccurs="0" />  
2136 <xs:element ref="xacml:Target" minOccurs="0" />  
2137 <xs:element ref="xacml:Condition" minOccurs="0" />  
2138 </xs:sequence>  
2139 <xs:attribute name="RuleId" type="xs:anyURI" use="required" />  
2140 <xs:attribute name="Effect" type="xacml:EffectType" use="required" />  
2141 </xs:complexType>
```

2142 The <Rule> element is of **RuleType** complex type.

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

2144 RuleId [Required]

2145 A URN identifying this **rule**.

2146 Effect [Required]

2147 **Rule effect.** Values of this attribute are either "Permit" or "Deny".

2148 <Description> [Optional]

2149 A free-form description of the **rule**.

2150

2151 <Target> [Optional]

2152 Identifies the set of **decision requests** that the <Rule> element is intended to evaluate. If
 2153 this element is omitted, then the **target** for the <Rule> SHALL be defined by the
 2154 <Target> element of the enclosing <Policy> element. See Section 5.5 for details.

2155 <Condition> [Optional]

2156 A **predicate** that MUST be satisfied for the **rule** to be assigned its Effect value. A
 2157 **condition** is a boolean function over a combination of **subject, resource, action** and
 2158 **environment attributes** or other functions.

2159 5.23. Simple type EffectType

2160 The **EffectType** simple type defines the values allowed for the Effect attribute of the <Rule>
 2161 element and for the FulfillOn attribute of the <Obligation> element.

```
2162 <xs:simpleType name="EffectType">
2163   <xs:restriction base="xs:string">
2164     <xs:enumeration value="Permit"/>
2165     <xs:enumeration value="Deny"/>
2166   </xs:restriction>
2167 </xs:simpleType>
```

2168 5.24. Element <Condition>

2169 The <Condition> element is a boolean function over **subject, resource, action** and
 2170 **environment attributes** or functions of **attributes**. If the <Condition> element evaluates to
 2171 "True", then the enclosing <Rule> element is assigned its Effect value.

```
2172 <xs:element name="Condition" type="xacml:ApplyType"/>
```

2173 The <Condition> element is of **ApplyType** complex type.

2174 5.25. Element <Apply>

2175 The <Apply> element denotes application of a function to its arguments, thus encoding a function
 2176 call. The <Apply> element can be applied to any combination of <Apply> ,
 2177 <AttributeValue> , <SubjectAttributeDesignator> ,
 2178 <ResourceAttributeDesignator> , <ActionAttributeDesignator> ,
 2179 <EnvironmentAttributeDesignator> and <AttributeSelector> arguments.

```
2180 <xs:element name="Apply" type="xacml:ApplyType"/>
2181 <xs:complexType name="ApplyType">
2182   <xs:choice minOccurs="0" maxOccurs="unbounded">
2183     <xs:element ref="xacml:Function"/>
2184     <xs:element ref="xacml:Apply"/>
2185     <xs:element ref="xacml:AttributeValue"/>
2186     <xs:element ref="xacml:SubjectAttributeDesignator"/>
2187     <xs:element ref="xacml:ResourceAttributeDesignator"/>
2188     <xs:element ref="xacml:ActionAttributeDesignator"/>
2189     <xs:element ref="xacml:EnvironmentAttributeDesignator"/>
2190     <xs:element ref="xacml:AttributeSelector"/>
2191   </xs:choice>
2192   <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>
2193 </xs:complexType>
```

2194 The <Apply> element is of **ApplyType** complex type.

2195 The <Apply> element contains the following attributes and elements:

- 2196 FunctionId [Required]
- 2197 The URN of a function. XACML-defined functions are described in Appendix A.
- 2198 <Function> [Optional]
- 2199 The name of a function that is applied to the elements of a **bag**. See Section A14.11.
- 2200 <Apply> [Optional]
- 2201 A nested function-call argument.
- 2202 <AttributeValue> [Optional]
- 2203 A literal value argument.
- 2204 <SubjectAttributeDesignator> [Optional]
- 2205 A **subject attribute** argument.
- 2206 <ResourceAttributeDesignator> [Optional]
- 2207 A **resource attribute** argument.
- 2208 <ActionAttributeDesignator> [Optional]
- 2209 An **action attribute** argument.
- 2210 <EnvironmentAttributeDesignator> [Optional]
- 2211 An **environment attribute** argument.
- 2212 <AttributeSelector> [Optional]
- 2213 An **attribute** selector argument.

2214 **5.26. Element <Function>**

- 2215 The `Function` element SHALL be used to name a function that is applied by the higher-order **bag**
- 2216 functions to every element of a **bag**. The higher-order **bag** functions are described in Section
- 2217 A14.11.

```
2218       <xs:element name="Function" type="xacml:FunctionType"/>
2219       <xs:complexType name="FunctionType">
2220        <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>
2221       </xs:complexType>
```

- 2222 The `Function` element is of **FunctionType** complex type.

- 2223 The `Function` element contains the following attributes:

- 2224 FunctionId [Required]

- 2225 The identifier for the function that is applied to the elements of a **bag** by the higher-order **bag**
- 2226 functions.

2227 **5.27. Complex type AttributeDesignatorType**

- 2228 The **AttributeDesignatorType** complex type is the type for elements and extensions that identify
- 2229 **attributes**. An element of this type contains properties by which it MAY be matched to **attributes**
- 2230 in the request **context**.

2231 In addition, elements of this type MAY control behaviour in the event that no matching **attribute** is
2232 present in the **context**.

2233 Elements of this type SHALL NOT alter the match semantics of named **attributes**, but MAY narrow
2234 the search space.

```
2235 <xs:complexType name="AttributeDesignatorType">  
2236 <xs:attribute name="AttributeId" type="xs:anyURI" use="required" />  
2237 <xs:attribute name="DataType" type="xs:anyURI" use="required" />  
2238 <xs:attribute name="Issuer" type="xs:anyURI" use="optional" />  
2239 <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"  
2240 default="false" />  
2241 </xs:complexType>
```

2242 A named **attribute** SHALL match an **attribute** if the values of their respective AttributeId,
2243 DataType and Issuer attributes match. The **attribute** designator's AttributeId MUST match,
2244 by URI equality, the AttributeId of the **attribute**. The **attribute** designator's DataType MUST
2245 match, by URI equality, the DataType of the same **attribute**.

2246 If the Issuer attribute is present in the **attribute** designator, then it MUST match, by URI equality,
2247 the Issuer of the same **attribute**. If the Issuer is not present in the **attribute** designator, then
2248 the matching of the **attribute** to the named **attribute** SHALL be governed by AttributeId and
2249 DataType attributes alone.

2250 The <AttributeDesignatorType> contains the following attributes:

2251 AttributeId [Required]

2252 This attribute SHALL specify the AttributeId with which to match the **attribute**.

2253 DataType [Required]

2254 This attribute SHALL specify the data-type with which to match the **attribute**.

2255 Issuer [Optional]

2256 This attribute, if supplied, SHALL specify the Issuer with which to match the **attribute**.

2257 MustBePresent [Optional]

2258 This attribute governs whether the element returns "Indeterminate" in the case where the
2259 the named **attribute** is absent. If the *named attribute* is absent and MustBePresent is
2260 "True", then this element SHALL result in "Indeterminate". The default value SHALL be
2261 "False".

2262 **5.28. Element <SubjectAttributeDesignator>**

2263 The <SubjectAttributeDesignator> element is of the **SubjectAttributeDesignatorType**.
2264 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType**
2265 complex type. It is the base type for elements and extensions that refer to *named categorized*
2266 **subject attributes**. A *named categorized subject attribute* is defined as follows:

2267 A **subject** is represented by a <Subject> element in the <xacml-context:Request> element.
2268 Each <Subject> element SHALL contain the XML attribute SubjectCategory. This attribute is
2269 called the *subject category attribute*.

2270 A *categorized subject* is a **subject** that is identified by a particular *subject category attribute*.

2271 A **subject attribute** is an **attribute** of a particular **subject**, i.e. contained within a <Subject>
2272 element.

2273 A named **subject attribute** is a named **attribute** for a **subject**.

2274 A named categorized **subject attribute** is a named **subject attribute** for a particular **categorized**
2275 **subject**.

2276 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType** with a
2277 **SubjectCategory** attribute. The **SubjectAttributeDesignatorType** extends the match
2278 semantics of the **AttributeDesignatorType** such that it narrows the **attribute** search space to the
2279 specific **categorized subject** such that the value of this element's **SubjectCategory** attribute
2280 matches, by string-equality, the value of the <Request> element's **subject category attribute**.

2281 If there are multiple **subjects** with the same **SubjectCategory** xml attribute, then they SHALL be
2282 treated as if they were one **categorized subject**.

2283 Elements and extensions of the **SubjectAttributeDesignatorType** complex type determine the
2284 presence of select **attribute values** associated with **named categorized subject attributes**.
2285 Elements and extensions of the **SubjectAttributeDesignatorType** SHALL NOT alter the match
2286 semantics of **named categorized subject attributes**, but MAY narrow the search space.

```

2287 <xs:complexType name="SubjectAttributeDesignatorType">
2288   <xs:complexContent>
2289     <xs:extension base="xacml:AttributeDesignatorType">
2290       <xs:attribute name="SubjectCategory"
2291         type="xs:anyURI"
2292         use="optional"
2293         default="
2294           urn:oasis:tc:xacml:1.0:subject-category:access-subject" />
2295     </xs:extension>
2296   </xs:complexContent>
2297 </xs:complexType>

```

2298 The <SubjectAttributeDesignatorType> complex type contains the following attribute in
2299 addition to the attributes of the **AttributeDesignatorType** complex type:

2300 **SubjectCategory** [Optional]

2301 This attribute SHALL specify the **categorized subject** from which to match **named subject**
2302 **attributes**. If **SubjectCategory** is not present, then its default value of
2303 "urn:oasis:tc:xacml:1.0:subject-category:access-subject" SHALL be used.

2304 5.29. Element <ResourceAttributeDesignator>

2305 The <ResourceAttributeDesignator> element retrieves a **bag** of values for a **named**
2306 **resource attribute**. A **resource attribute** is an **attribute** contained within the <Resource>
2307 element of the <xacml-context:Request> element. A **named resource attribute** is a **named**
2308 **attribute** that matches a **resource attribute**. A **named resource attribute** SHALL be considered
2309 **present** if there is at least one **resource attribute** that matches the criteria set out below. A
2310 **resource attribute** value is an **attribute** value that is contained within a **resource attribute**.

2311 The <ResourceAttributeDesignator> element SHALL return a **bag** containing all the
2312 **resource attribute** values that are matched by the **named resource attribute**. The
2313 **MustBePresent** attribute governs whether this element returns an empty **bag** or "Indeterminate"
2314 in the case that the **named resource attribute** is absent. If the **named resource attribute** is not
2315 present and the **MustBePresent** attribute is "False" (its default value), then this element SHALL
2316 evaluate to an empty **bag**. If the **named resource attribute** is not present and the
2317 **MustBePresent** attribute is "True", then this element SHALL evaluate to "Indeterminate".
2318 Regardless of the **MustBePresent** attribute, if it cannot be determined whether the **named**
2319 **resource attribute** is present or not in the **request context**, or the value of the **named resource**
2320 **attribute** is unavailable, then the expression SHALL evaluate to "Indeterminate".

2321 A *named resource attribute* SHALL match a **resource attribute** as per the match semantics
2322 specified in the **AttributeDesignatorType** complex type [Section 5.27]

2323 The <ResourceAttributeDesignator> MAY appear in the <ResourceMatch> element and
2324 MAY be passed to the <Apply> element as an argument.

```
2325 <xs:element name="ResourceAttributeDesignator"  
2326 type="xacml:AttributeDesignatorType" />
```

2327 The <ResourceAttributeDesignator> element is of the **AttributeDesignatorType**
2328 complex type.

2329 5.30. Element <ActionAttributeDesignator>

2330 The <ActionAttributeDesignator> element retrieves a **bag** of values for a *named action*
2331 **attribute**. An **action attribute** is an **attribute** contained within the <Action> element of the
2332 <xacml-context:Request> element. A *named action attribute* has specific criteria (described
2333 below) with which to match an **action attribute**. A *named action attribute* SHALL be considered
2334 *present*, if there is at least one **action attribute** that matches the criteria. An **action attribute value**
2335 is an **attribute value** that is contained within an **action attribute**.

2336 The <ActionAttributeDesignator> element SHALL return a **bag** of all the **action attribute**
2337 values that are matched by the *named action attribute*. The `MustBePresent` attribute governs
2338 whether this element returns an empty **bag** or “Indeterminate” in the case that the *named action*
2339 **attribute** is absent. If the *named action attribute* is not present and the `MustBePresent` attribute
2340 is “False” (its default value), then this element SHALL evaluate to an empty **bag**. If the *named*
2341 **action attribute** is not present and the `MustBePresent` attribute is “True”, then this element
2342 SHALL evaluate to “Indeterminate”. Regardless of the `MustBePresent` attribute, if it cannot be
2343 determined whether the *named action attribute* is present or not present in the request **context**, or
2344 the value of the *named action attribute* is unavailable, then the expression SHALL evaluate to
2345 “Indeterminate”.

2346 A *named action attribute* SHALL match an **action attribute** as per the match semantics specified
2347 in the **AttributeDesignatorType** complex type [Section 5.27].

2348 The <ActionAttributeDesignator> MAY appear in the <ActionMatch> element and MAY
2349 be passed to the <Apply> element as an argument.

```
2350 <xs:element name="ActionAttributeDesignator"  
2351 type="xacml:AttributeDesignatorType" />
```

2352 The <ActionAttributeDesignator> element is of the **AttributeDesignatorType** complex
2353 type.

2354 5.31. Element <EnvironmentAttributeDesignator>

2355 The <EnvironmentAttributeDesignator> element retrieves a **bag** of values for a *named*
2356 **environment attribute**. An **environment attribute** is an **attribute** contained within the
2357 <Environment> element of the <xacml-context:Request> element. A *named environment*
2358 **attribute** has specific criteria (described below) with which to match an **environment attribute**. A
2359 *named environment attribute* SHALL be considered *present*, if there is at least one **environment**
2360 **attribute** that matches the criteria. An **environment attribute value** is an **attribute value** that is
2361 contained within an **environment attribute**.

2362 The <EnvironmentAttributeDesignator> element SHALL evaluate to a **bag** of all the
2363 **environment attribute** values that are matched by the *named environment attribute*. The
2364 `MustBePresent` attribute governs whether this element returns an empty **bag** or “Indeterminate”
2365 in the case that the *named environment attribute* is absent. If the *named environment attribute*

2366 is not present and the `MustBePresent` attribute is "False" (its default value), then this element
2367 SHALL evaluate to an empty **bag**. If the *named environment attribute* is not present and the
2368 `MustBePresent` attribute is "True", then this element SHALL evaluate to "Indeterminate".
2369 Regardless of the `MustBePresent` attribute, if it cannot be determined whether the *named*
2370 **environment attribute** is present or not present in the request **context**, or the value of the *named*
2371 **environment attribute** is unavailable, then the expression SHALL evaluate to "Indeterminate".

2372 A *named environment attribute* SHALL match an **environment attribute** as per the match
2373 semantics specified in the **AttributeDesignatorType** complex type [Section 5.27].

2374 The `<EnvironmentAttributeDesignator>` MAY be passed to the `<Apply>` element as an
2375 argument.

```
2376 <xs:element name="EnvironmentAttributeDesignator"  
2377           type="xacml:AttributeDesignatorType" />
```

2378 The `<EnvironmentAttributeDesignator>` element is of the **AttributeDesignatorType**
2379 complex type.

2380 5.32. Element `<AttributeSelector>`

2381 The `AttributeSelector` element's `RequestContextPath` XML attribute SHALL contain a
2382 legal XPath expression whose context node is the `<xacml-context:Request>` element. The
2383 `AttributeSelector` element SHALL evaluate to a **bag** of values whose data-type is specified by
2384 the element's `DataType` attribute. If the `DataType` specified in the `AttributeSelector` is a
2385 primitive data type defined in [XF] or [XS], then the value returned by the XPath expression SHALL
2386 be converted to the `DataType` specified in the `AttributeSelector` using the constructor
2387 function below [XF Section 4] that corresponds to the `DataType`. If an error results from using the
2388 constructor function, then the value of the `AttributeSelector` SHALL be "Indeterminate".

```
2389     xs:string()  
2390     xs:boolean()  
2391     xs:integer()  
2392     xs:double()  
2393     xs:dateTime()  
2394     xs:date()  
2395     xs:time()  
2396     xs:hexBinary()  
2397     xs:base64Binary()  
2398     xs:anyURI()  
2399     xf:yearMonthDuration()  
2400     xf:dayTimeDuration()  
2401
```

2402
2403 If the `DataType` specified in the `AttributeSelector` is not one of the preceding primitive
2404 `DataTypes`, then the `AttributeSelector` SHALL return a bag of instances of the specified
2405 `DataType`. If there are errors encountered in converting the values returned by the XPath
2406 expression to the specified `DataType`, then the result of the `AttributeSelector` SHALL be
2407 "Indeterminate".

2408
2409 If the policy writer intends to select the string value of an element's contents rather than the node
2410 representing the element itself, then the XPath expression MUST terminate in `"/text()`". The
2411 resulting sequence of string-data SHALL be converted to a **bag** of values of the type that is implied
2412 by the type system.

2413 Support for the `<AttributeSelector>` element is OPTIONAL.

2414

```

2415 <xs:element name="AttributeSelector" type="xacml:AttributeSelectorType"/>
2416 <xs:complexType name="AttributeSelectorType">
2417   <xs:attribute name="RequestContextPath" type="xs:string" use="required"/>
2418   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
2419   <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"
2420   default="false"
2421 </xs:complexType>

```

2422 The <AttributeSelector> element is of **AttributeSelectorType** complex type.

2423 The <AttributeSelector> element has the following attributes:

2424 RequestContextPath [Required]

2425 An XPath expression whose context node is the <xacml-context:Request> element.
 2426 There SHALL be no restriction on the XPath syntax.

2427 DataType [Required]

2428 The bag of values returned by the AttributeSelector SHALL be of this data type.

2429 MustBePresent [Optional]

2430 Whether or not the designated *attribute* must be present in the *context*.

2431 **5.33. Element <AttributeValue>**

2432 The <AttributeValue> element SHALL contain a literal *attribute* value.

```

2433 <xs:element name="AttributeValue" type="xacml:AttributeValueType"/>
2434 <xs:complexType name="AttributeValueType" mixed="true">
2435   <xs:sequence>
2436     <xs:any namespace="##any" processContents="lax" minOccurs="0"
2437     maxOccurs="unbounded"/>
2438   </xs:sequence>
2439   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
2440   <xs:anyAttribute namespace="##any" processContents="lax"/>
2441 </xs:complexType>

```

2442 The <AttributeValue> element is of **AttributeValueType** complex type.

2443 The <AttributeValue> element has the following attributes:

2444 DataType [Required]

2445 The data-type of the *attribute* value.

2446 **5.34. Element <Obligations>**

2447 The <Obligations> element SHALL contain a set of <Obligation> elements.

2448 Support for the <Obligations> element is OPTIONAL.

```

2449 <xs:element name="Obligations" type="xacml:ObligationsType"/>
2450 <xs:complexType name="ObligationsType">
2451   <xs:sequence>
2452     <xs:element ref="xacml:Obligation" maxOccurs="unbounded"/>
2453   </xs:sequence>
2454 </xs:complexType>

```

2455 The <Obligations> element is of **ObligationsType** complexType.

2456 The <Obligations> element contains the following element:

2457 <Obligation> [One to Many]

2458 A sequence of **obligations**

2459 5.35. Element <Obligation>

2460 The <Obligation> element SHALL contain an identifier for the **obligation** and a set of **attributes**
2461 that form arguments of the action defined by the **obligation**. The FulfillOn attribute SHALL
2462 indicate the **effect** for which this **obligation** applies.

```
2463 <xs:element name="Obligation" type="xacml:ObligationType"/>  
2464 <xs:complexType name="ObligationType">  
2465 <xs:sequence>  
2466 <xs:element ref="xacml:AttributeAssignment" maxOccurs="unbounded"/>  
2467 </xs:sequence>  
2468 <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>  
2469 <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required"/>  
2470 </xs:complexType>
```

2471 The <Obligation> element is of **ObligationType** complexType. See Section 7.11 for a
2472 description of how the set of **obligations** to be returned by the PDP is determined.

2473 The <Obligation> element contains the following elements and attributes:

2474 ObligationId [Required]

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

2477 FulfillOn [Required]

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

2479 <AttributeAssignment> [One To Many]

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

2482 5.36. Element <AttributeAssignment>

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

```
2487 <xs:element name="AttributeAssignment"  
2488 type="xacml:AttributeAssignmentType"/>  
2489 <xs:complexType name="AttributeAssignmentType" mixed="true">  
2490 <xs:complexContent>  
2491 <xs:extension base="xacml:AttributeValueType">  
2492 <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2493 </xs:extension>  
2494 </xs:complexContent>  
2495 </xs:complexType>
```

2496 The <AttributeAssignment> element is of **AttributeAssignmentType** complex type.

2497 The <AttributeAssignment> element contains the following attributes:

2498 AttributeId [Required]

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

2502 6.1. Element <Request>

2503 The <Request> element is a top-level element in the XACML *context* schema. The <Request>
2504 element is an abstraction layer used by the *policy* language. Any proprietary system using the
2505 XACML specification MUST transform its *decision request* into the form of an XACML *context*
2506 <Request>.

2507 The <Request> element contains <Subject>, <Resource>, <Action> and <Environment>
2508 elements. There may be multiple <Subject> elements. Each child element contains a sequence
2509 of <xacml-context:Attribute> elements associated with the *subject*, *resource*, *action* and
2510 *environment* respectively.

```
2511 <xs:element name="Request" type="xacml-context:RequestType"/>
2512 <xs:complexType name="RequestType">
2513   <xs:sequence>
2514     <xs:element ref="xacml-context:Subject" maxOccurs="unbounded"/>
2515     <xs:element ref="xacml-context:Resource"/>
2516     <xs:element ref="xacml-context:Action"/>
2517     <xs:element ref="xacml-context:Environment" minOccurs="0"/>
2518   </xs:sequence>
2519 </xs:complexType>
```

2520 The <Request> element is of **RequestType** complex type.

2521 The <Request> element contains the following elements:

2522 <Subject> [One to Many]

2523 Specifies information about a *subject* of the request *context* by listing a sequence of
2524 <Attribute> elements associated with the *subject*. One or more <Subject> elements
2525 are allowed. A *subject* is an entity associated with the *access* request. One *subject*
2526 might represent the human user that initiated the application from which the request was
2527 issued. Another *subject* might represent the application's executable code that created the
2528 request. Another *subject* might represent the machine on which the application was
2529 executing. Another *subject* might represent the entity that is to be the recipient of the
2530 *resource*. Attributes of each of these entities MUST be enclosed in a separate
2531 <Subject> element.

2532 <Resource> [Required]

2533 Specifies information about the resource for which access is being requested by listing a
2534 sequence of <Attribute> elements associated with the resource. It MAY include a
2535 <ResourceContent> element.

2536 <Action> [Required]

2537 Specifies the requested *action* to be performed on the *resource* by listing a set of
2538 <Attribute> elements associated with the *action*.

2539 <Environment> [Optional]

2540 Contains a set of <Attribute> elements of the **environment**. These <Attribute>
2541 elements MAY form a part of **policy** evaluation.

2542 6.2. Element <Subject>

2543 The <Subject> element specifies a **subject** by listing a sequence of <Attribute> elements
2544 associated with the **subject**.

```
2545 <xs:element name="Subject" type="xacml-context:SubjectType" />  
2546 <xs:complexType name="SubjectType">  
2547 <xs:sequence>  
2548 <xs:element ref="xacml-context:Attribute" minOccurs="0"  
2549 maxOccurs="unbounded" />  
2550 </xs:sequence>  
2551 <xs:attribute name="SubjectCategory" type="xs:anyURI" use="optional"  
2552 default="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" />  
2553 </xs:complexType>
```

2554 The <Subject> element is of **SubjectType** complex type.

2555 The <Subject> element contains the following elements:

2556 SubjectCategory [Optional]

2557 This attribute indicates the role that the parent <Subject> played in the formation of the
2558 access request. If this attribute is not present in a given <Subject> element, then the
2559 default value of "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" SHALL be
2560 used, indicating that the parent <Subject> element represents the entity ultimately
2561 responsible for initiating the **access** request.

2562 If more than one <Subject> element contains a "urn:oasis:names:tc:xacml:1.0:subject-
2563 category" attribute with the same value, then the PDP SHALL treat the contents of those
2564 elements as if they were contained in the same <Subject> element.

2565 <Attribute> [Any Number]

2566 A sequence of attributes that apply to the subject.

2567 Typically, a <Subject> element will contain an <Attribute> with an AttributeId of
2568 "urn:oasis:names:tc:xacml:1.0:subject:subject-id", containing the identity of the **subject**.

2569 A <Subject> element MAY contain additional <Attribute> elements.

2570 6.3. Element <Resource>

2571 The <Resource> element specifies information about the **resource** to which **access** is requested,
2572 by listing a sequence of <Attribute> elements associated with the **resource**. It MAY include the
2573 **resource** content.

```
2574 <xs:element name="Resource" type="xacml-context:ResourceType" />  
2575 <xs:complexType name="ResourceType">  
2576 <xs:sequence>  
2577 <xs:element ref="xacml-context:ResourceContent" minOccurs="0" />  
2578 <xs:element ref="xacml-context:Attribute" minOccurs="0"  
2579 maxOccurs="unbounded" />  
2580 </xs:sequence>  
2581 </xs:complexType>
```

2582 The <Resource> element is of **ResourceType** complex type.

2583 The <Resource> element contains the following elements:

2584 <ResourceContent> [Optional]

2585 The **resource** content.

2586 <Attribute> [Any Number]

2587 A sequence of **resource attributes**. The <Resource> element MUST contain one and
2588 only one <Attribute> with an AttributeId of

2589 "urn:oasis:names:tc:xacml:1.0:resource:resource-id". This **attribute**
2590 specifies the identity of the **resource** to which **access** is requested.

2591 A <Resource> element MAY contain additional <Attribute> elements.

2592 **6.4. Element <ResourceContent>**

2593 The <ResourceContent> element is a notional placeholder for the **resource** content. If an
2594 XACML **policy** references the contents of the **resource**, then the <ResourceContent> element
2595 SHALL be used as the reference point.

```
2596 <xs:complexType name="ResourceContentType" mixed="true">  
2597 <xs:sequence>  
2598 <xs:any namespace="##any" processContents="lax" minOccurs="0"  
2599 maxOccurs="unbounded" />  
2600 </xs:sequence>  
2601 <xs:anyAttribute namespace="##any" processContents="lax" />  
2602 </xs:complexType>
```

2603 The <ResourceContent> element is of **ResourceContentType** complex type.

2604 The <ResourceContent> element allows arbitrary elements and attributes.

2605 **6.5. Element <Action>**

2606 The <Action> element specifies the requested **action** on the **resource**, by listing a set of
2607 <Attribute> elements associated with the **action**.

```
2608 <xs:element name="Action" type="xacml-context:ActionType" />  
2609 <xs:complexType name="ActionType">  
2610 <xs:sequence>  
2611 <xs:element ref="xacml-context:Attribute" minOccurs="0"  
2612 maxOccurs="unbounded" />  
2613 </xs:sequence>  
2614 </xs:complexType>
```

2615 The <Action> element is of **ActionType** complex type.

2616 The <Action> element contains the following elements:

2617 <Attribute> [Any Number]

2618 List of **attributes** of the **action** to be performed on the **resource**.

2619 **6.6. Element <Environment>**

2620 The <Environment> element contains a set of **attributes** of the **environment**. These **attributes**
2621 MAY form part of the **policy** evaluation.

2622

```

2623     <xs:element name="Environment" type="xacml-context:EnvironmentType" />
2624     <xs:complexType name="EnvironmentType">
2625         <xs:sequence>
2626             <xs:element ref="xacml-context:Attribute" minOccurs="0"
2627             maxOccurs="unbounded" />
2628         </xs:sequence>
2629     </xs:complexType>

```

2630 The <Environment> element is of **EnvironmentType** complex type.

2631 The <Environment> element contains the following elements:

2632 <Attribute> [Any Number]

2633 A list of **environment attributes**. Environment **attributes** are **attributes** that are not
2634 associated with either the **resource**, the **action** or any of the **subjects** of the **access**
2635 request.

2636 6.7. Element <Attribute>

2637 The <Attribute> element is the central abstraction of the request **context**. It contains an
2638 **attribute** value and **attribute** meta-data. The **attribute** meta-data comprises the **attribute**
2639 identifier, the **attribute** issuer and the **attribute** issue instant. **Attribute** designators and **attribute**
2640 selectors in the **policy** MAY refer to **attributes** by means of this meta-data.

```

2641     <xs:element name="Attribute" type="xacml-context:AttributeType" />
2642     <xs:complexType name="AttributeType">
2643         <xs:sequence>
2644             <xs:element ref="xacml-context:AttributeValue" minOccurs="0" />
2645         </xs:sequence>
2646         <xs:attribute name="AttributeId" type="xs:anyURI" use="required" />
2647         <xs:attribute name="DataType" type="xs:anyURI" use="required" />
2648         <xs:attribute name="Issuer" type="xs:string" use="optional" />
2649         <xs:attribute name="IssueInstant" type="xs:dateTime" use="optional" />
2650     </xs:complexType>

```

2651 The <Attribute> element is of **AttributeType** complex type.

2652 The <Attribute> element contains the following attributes and elements:

2653 AttributeId [Required]

2654 **Attribute** identifier. A number of identifiers are reserved by XACML to denote commonly
2655 used **attributes**.

2656 DataType [Required]

2657 The data-type of the contents of the <AttributeValue> element. This SHALL be either
2658 a primitive type defined by the XACML 1.0 specification or a type defined in a namespace
2659 declared in the <xacml-context> element.

2660 Issuer [Optional]

2661 **Attribute** issuer. This attribute value MAY be an x500Name that binds to a public key, or it
2662 may be some other identifier exchanged out-of-band by issuing and relying parties.

2663 IssueInstant [Optional]

2664 The date and time at which the **attribute** was issued.

2665

2666 <AttributeValue> [Optional]

2667 At most one *attribute* value.

2668 **6.8. Element <AttributeValue>**

2669 The <AttributeValue> element contains the value of an *attribute*.

```
2670 <xs:element name="AttributeValue" type="xacml-context:AttributeValueType" />
2671 <xs:complexType name="AttributeValueType" mixed="true">
2672 <xs:sequence>
2673 <xs:any namespace="##any" processContents="lax" minOccurs="0"
2674 maxOccurs="unbounded" />
2675 </xs:sequence>
2676 <xs:anyAttribute namespace="##any" processContents="lax" />
2677 </xs:complexType>
```

2678 The <AttributeValue> element is of **AttributeValueType** type.

2679 The data-type of the <AttributeValue> MAY be specified by using the `DataType` attribute of
2680 the parent <Attribute> element.

2681 **6.9. Element <Response>**

2682 The <Response> element is a top-level element in the XACML *context* schema. The
2683 <Response> element is an abstraction layer used by the *policy* language. Any proprietary
2684 system using the XACML specification MUST transform an XACML *context* <Response> into the
2685 form of its *authorization decision*.

2686 The <Response> element encapsulates the *authorization decision* produced by the *PDP*. It
2687 includes a sequence of one or more results, with one <Result> element per requested *resource*.
2688 Multiple results MAY be returned when the value of the “urn:oasis:xacml:1.0:resource:scope”
2689 resource *attribute* in the request *context* is “Descendants” or “Children”. Support for multiple
2690 results is OPTIONAL.

```
2691 <xs:element name="Response" type="xacml-context:ResponseType" />
2692 <xs:complexType name="ResponseType">
2693 <xs:sequence>
2694 <xs:element ref="xacml-context:Result" maxOccurs="unbounded" />
2695 </xs:sequence>
2696 </xs:complexType>
```

2697 The <Response> element is of **ResponseType** complex type.

2698 The <Response> element contains the following elements:

2699 <Result> [One to Many]

2700 An authorization decision result.

2701 **6.10. Element <Result>**

2702 The <Result> element represents an *authorization decision* result for the *resource* specified by
2703 the `ResourceId` *attribute*. It MAY include a set of *obligations* that MUST be fulfilled by the *PEP*.
2704 If the *PEP* does not understand an *obligation*, then it MUST act as if the *PDP* had denied *access*
2705 to the requested *resource*.

2706

```
2707 <xs:element name="Result" type="xacml-context:ResultType" />
```

```

2708 <xs:complexType name="ResultType">
2709   <xs:sequence>
2710     <xs:element ref="xacml-context:Decision"/>
2711     <xs:element ref="xacml-context:Status"/>
2712     <xs:element ref="xacml:Obligations" minOccurs="0"/>
2713   </xs:sequence>
2714   <xs:attribute name="ResourceId" type="xs:string" use="optional"/>
2715 </xs:complexType>

```

2716 The <Result> element is of **ResultType** complex type.

2717 The <Result> element contains the following attributes and elements:

2718 ResourceId [Optional]

2719 The identifier of the requested **resource**. If this attribute is omitted, then the **resource**
2720 identity is specified by the "urn:oasis:names:tc:xacml:1.0:resource:resource-
2721 id" **resource attribute** in the corresponding <Request> element.

2722 <Decision> [Required]

2723 The **authorization decision**: "Permit", "Deny", "Indeterminate" or "NotApplicable".

2724 <Status> [Required]

2725 Indicates whether errors occurred during evaluation of the **decision request**, and
2726 optionally, information about those errors.

2727 <xacml:Obligations> [Optional]

2728 A list of **obligations** that MUST be fulfilled by the **PEP**. If the **PEP** does not understand an
2729 **obligation**, then it MUST act as if the **PDP** had denied **access** to the requested **resource**.
2730 See Section 7.11 for a description of how the set of **obligations** to be returned by the PDP
2731 is determined.

2732 6.11. Element <Decision>

2733 The <Decision> element contains the result of **policy** evaluation.

```

2734 <xs:element name="Decision" type="xacml-context:DecisionType"/>
2735 <xs:simpleType name="DecisionType">
2736   <xs:restriction base="xs:string">
2737     <xs:enumeration value="Permit"/>
2738     <xs:enumeration value="Deny"/>
2739     <xs:enumeration value="Indeterminate"/>
2740     <xs:enumeration value="NotApplicable"/>
2741   </xs:restriction>
2742 </xs:simpleType>

```

2743 The <Decision> element is of **DecisionType** simple type.

2744 The values of the <Decision> element have the following meanings:

2745 "Permit": the requested **access** is permitted.

2746 "Deny": the requested **access** is denied.

2747 "Indeterminate": the **PDP** is unable to evaluate the requested **access**. Reasons for such
2748 inability include: missing **attributes**, network errors while retrieving **policies**, division by
2749 zero during **policy** evaluation, syntax errors in the **decision request** or in the **policy**, etc..

2750 "NotApplicable": the **PDP** does not have any **policy** that applies to this **decision request**.

2751

6.12. Element <Status>

2752 The <Status> element represents the status of the *authorization decision* result.

```
2753 <xs:element name="Status" type="xacml-context:StatusType"/>
2754 <xs:complexType name="StatusType">
2755   <xs:sequence>
2756     <xs:element ref="xacml-context:StatusCode"/>
2757     <xs:element ref="xacml-context:StatusMessage" minOccurs="0"/>
2758     <xs:element ref="xacml-context:StatusDetail" minOccurs="0"/>
2759   </xs:sequence>
2760 </xs:complexType>
```

2761 The <Status> element is of **StatusType** complex type.

2762 The <Status> element contains the following elements:

2763 <StatusCode> [Required]

2764 Status code.

2765 <StatusMessage> [Optional]

2766 A status message describing the status code.

2767 <StatusDetail> [Optional]

2768 Additional status information.

2769 6.13. Element <StatusCode>

2770 The <StatusCode> element contains a major status code value and an optional sequence of
2771 minor status codes.

```
2772 <xs:element name="StatusCode" type="xacml-context:StatusCodeType"/>
2773 <xs:complexType name="StatusCodeType">
2774   <xs:sequence>
2775     <xs:element ref="xacml-context:StatusCode" minOccurs="0"/>
2776   </xs:sequence>
2777   <xs:attribute name="Value" type="xs:anyURI" use="required"/>
2778 </xs:complexType>
```

2779 The <StatusCode> element is of **StatusCodeType** complex type.

2780 The <StatusCode> element contains the following attributes and elements:

2781 Value [Required]

2782 See Section B.9 for a list of values.

2783 <StatusCode> [Any Number]

2784 Minor status code. This status code qualifies its parent status code.

2785 6.14. Element <StatusMessage>

2786 The <StatusMessage> element is a free-form description of the status code.

```
2787 <xs:element name="StatusMessage" type="xs:string"/>
```

2788 The <StatusMessage> element is of **xs:string** type.

2789

6.15. Element <StatusDetail>

2790 The <StatusDetail> element qualifies the <Status> element with additional information.

```
2791 <xs:element name="StatusDetail" type="xacml-context:StatusDetailType"/>
2792 <xs:complexType name="StatusDetailType">
2793   <xs:sequence>
2794     <xs:any namespace="##any" processContents="lax" minOccurs="0"
2795     maxOccurs="unbounded"/>
2796   </xs:sequence>
2797 </xs:complexType>
```

2798 The <StatusDetail> element is of **StatusDetailType** complex type.

2799 The <StatusDetail> element allows arbitrary XML content.

2800 Inclusion of a <StatusDetail> element is optional. However, if a **PDP** returns one of the
2801 following XACML-defined <StatusCode> values and includes a <StatusDetail> element, then
2802 the following rules apply.

2803 urn:oasis:names:tc:xacml:1.0:status:ok

2804 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “ok” status value.

2805 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

2806 A **PDP** MAY choose not to return any <StatusDetail> information or MAY choose to return a
2807 <StatusDetail> element containing one or more <xacml-context:Attribute> elements. If
2808 the **PDP** includes <AttributeValue> elements in the <Attribute> element, then this indicates
2809 the acceptable values for that **attribute**. If no <AttributeValue> elements are included, then
2810 this indicates the names of **attributes** that the **PDP** failed to resolve during its evaluation. The list
2811 of **attributes** may be partial or complete. There is no guarantee by the **PDP** that supplying the
2812 missing values or **attributes** will be sufficient to satisfy the **policy**.

2813 urn:oasis:names:tc:xacml:1.0:status:syntax-error

2814 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “syntax-error” status
2815 value. A syntax error may represent either a problem with the **policy** being used or with the
2816 request **context**. The **PDP** MAY return a <StatusMessage> describing the problem.

2817 urn:oasis:names:tc:xacml:1.0:status:processing-error

2818 A **PDP** MUST NOT return <StatusDetail> element in conjunction with the “processing-error”
2819 status value. This status code indicates an internal problem in the **PDP**. For security reasons, the
2820 **PDP** MAY choose to return no further information to the **PEP**. In the case of a divide-by-zero error
2821 or other computational error, the **PDP** MAY return a <StatusMessage> describing the nature of
2822 the error.

2823 7. Functional requirements (normative)

2824 This section specifies certain functional requirements that are not directly associated with the
2825 production or consumption of a particular XACML element.

2826 7.1. Policy enforcement point

2827 This section describes the requirements for the **PEP**.

2828 An application functions in the role of the **PEP** if it guards access to a set of **resources** and asks
2829 the **PDP** for an **authorization decision**. The **PEP** MUST abide by the **authorization decision** in
2830 the following way:

2831 A **PEP** SHALL allow access to the **resource** only if a valid XACML response of "Permit" is returned
2832 by the **PDP**. The **PEP** SHALL deny access to the **resource** in all other cases. An XACML
2833 response of "Permit" SHALL be considered valid only if the **PEP** understands all of the **obligations**
2834 contained in the response.

2835 7.2. Base policy

2836 A **PDP** SHALL represent one **policy** or **policy set**, called its *base policy*. This base **policy** MAY be
2837 a <Policy> element containing a <Target> element that matches every possible **decision**
2838 **request**, or (for instance) it MAY be a <Policy> element containing a <Target> element that
2839 matches only a specific **subject**. In such cases, the base policy SHALL form the root-node of a
2840 tree of policies connected by <PolicyIdReference> and <PolicySetIdReference>
2841 elements to all the **rules** that may be applicable to any **decision request** that the **PDP** is capable
2842 of evaluating.

2843 In the case of a **PDP** that retrieves **policies** according to the **decision request** that it is processing,
2844 the base policy SHALL contain a <Policy> element containing a <Target> element that matches
2845 every possible **decision request** and a PolicyCombiningAlgId attribute with the value "Only-
2846 one-applicable". In other words, the **PDP** SHALL return an error if it retrieves policies that do not
2847 form a single tree.

2848 7.3. Target evaluation

2849 The **target** value SHALL be "Match" if the **subject**, **resource** and **action** specified in the **target** all
2850 match values in the request **context**. The **target** value SHALL be "No-match" if one or more of the
2851 **subject**, **resource** and **action** specified in the **target** do not match values in the request **context**.
2852 The value of a <SubjectMatch>, <ResourceMatch> or <ActionMatch> element, in which a
2853 referenced **attribute** value cannot be obtained, depends on the value of the MustBePresent
2854 attribute of the <AttributeDesignator> or <AttributeSelector> element. If the
2855 MustBePresent attribute is "True", then the result of the <SubjectMatch>, <ResourceMatch>
2856 or <ActionMatch> element SHALL be "Indeterminate" in this case. If the MustBePresent
2857 attribute is "False" or missing, then the result of the <SubjectMatch>, <ResourceMatch> or
2858 <ActionMatch> element SHALL be "No-match".

2859 7.4. Condition evaluation

2860 The **condition** value SHALL be "True" if the <Condition> element is absent, or if it evaluates to
2861 "True" for the **attribute** values supplied in the request **context**. Its value is "False" if the
2862 <Condition> element evaluates to "False" for the **attribute** values supplied in the request
2863 **context**. If any **attribute** value referenced in the **condition** cannot be obtained, then the **condition**
2864 SHALL evaluate to "Indeterminate".

2865 7.5. Rule evaluation

2866 A **rule** has a value that can be calculated by evaluating its contents. **Rule** evaluation involves
2867 separate evaluation of the **rule's target** and **condition**. The **rule** truth table is shown in Table 1.

2868

2869

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

2870

Table 1 - Rule truth table

2871 If the **target** value is "No-match" or "Indeterminate" then the **rule** value SHALL be "NotApplicable"
 2872 or "Indeterminate", respectively, regardless of the value of the **condition**. For these cases,
 2873 therefore, the **condition** need not be evaluated in order to determine the **rule** value.

2874 If the **target** value is "Match" and the **condition** value is "True", then the **effect** specified in the **rule**
 2875 SHALL determine the **rule** value.

2876 7.6. Policy evaluation

2877 The value of a **policy** SHALL be determined only by its contents, considered in relation to the
 2878 contents of the **request context**. A **policy's** value SHALL be determined by evaluation of the
 2879 **policy's target** and **rules**, according to the specified **rule-combining algorithm**.

2880 The **policy's target** SHALL be evaluated to determine the applicability of the **policy**. If the **target**
 2881 evaluates to "Match", then the value of the **policy** SHALL be determined by evaluation of the
 2882 **policy's rules**, according to the specified **rule-combining algorithm**. If the **target** evaluates to
 2883 "No-Match", then the value of the **policy** SHALL be "NotApplicable". If the **target** evaluates to
 2884 "Indeterminate", then the value of the **policy** SHALL be "Indeterminate".

2885 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"

2886

Table 2 - Policy truth table

2887 A Rules value of "At-least-one-applicable" SHALL be used if the <Rule> element is absent, or if
 2888 one or more of the **rules** contained in the **policy** is applicable to the **decision request** (i.e., returns
 2889 a value of "Effect"; see Section 7.5). A value of "None-applicable" SHALL be used if no **rule**
 2890 contained in the **policy** is applicable to the request and if no **rule** contained in the **policy** returns a

2891 value of "Indeterminate". If no **rule** contained in the **policy** is applicable to the request but one or
 2892 more **rule** returns a value of "Indeterminate", then **rules** SHALL evaluate to "Indeterminate".

2893 If the **target** value is "No-match" or "Indeterminate" then the **policy** value SHALL be
 2894 "NotApplicable" or "Indeterminate", respectively, regardless of the value of the **rules**. For these
 2895 cases, therefore, the **rules** need not be evaluated in order to determine the **policy** value.

2896 If the **target** value is "Match" and the **rules** value is "At-least-one-applicable" or "Indeterminate",
 2897 then the **rule-combining algorithm** specified in the **policy** SHALL determine the **policy** value.

2898 7.7. Policy Set evaluation

2899 The value of a **policy set** SHALL be determined by its contents, considered in relation to the
 2900 contents of the **request context**. A **policy set's** value SHALL be determined by evaluation of the
 2901 **policy set's target, policies** and **policy sets**, according to the specified **policy-combining**
 2902 **algorithm**.

2903 The **policy set's target** SHALL be evaluated to determine the applicability of the **policy set**. If the
 2904 **target** evaluates to "Match" then the value of the **policy set** SHALL be determined by evaluation of
 2905 the **policy set's policies** and **policy sets**, according to the specified **policy-combining algorithm**.
 2906 If the **target** evaluates to "Not-Match", then the value of the **policy set** shall be "NotApplicable". If
 2907 the **target** evaluates to "Indeterminate", then the value of the **policy set** SHALL be "Indeterminate".

2908 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"

2909 **Table 3 – Policy set truth table**

2910 A **policies** value of "At-least-one-applicable" SHALL be used if there are no contained or
 2911 referenced **policies** or **policy sets**, or if one or more of the **policies** or **policy sets** contained in or
 2912 referenced by the **policy set** is applicable to the **decision request** (i.e., returns a value determined
 2913 by its **rule-combining algorithm**; see Section 7.6). A value of "None-applicable" SHALL be used if
 2914 no **policy** or **policy set** contained in or referenced by the **policy set** is applicable to the request
 2915 and if no **policy** or **policy set** contained in or referenced by the **policy set** returns a value of
 2916 "Indeterminate". If no **policy** or **policy set** contained in or referenced by the **policy set** is
 2917 applicable to the request but one or more **policy** or **policy set** returns a value of "Indeterminate",
 2918 then **policies** SHALL evaluate to "Indeterminate".

2919 If the **target** value is "No-match" or "Indeterminate" then the **policy set** value SHALL be
 2920 "NotApplicable" or "Indeterminate", respectively, regardless of the value of the **policies**. For these
 2921 cases, therefore, the **policies** need not be evaluated in order to determine the **policy set** value.

2922 If the **target** value is "Match" and the **policies** value is "At-least-one-applicable" or "Indeterminate",
2923 then the **policy-combining algorithm** specified in the **policy set** SHALL determine the **policy set**
2924 value.

2925 7.8. Hierarchical resources

2926 It is often the case that a **resource** is organized as a hierarchy (e.g. file system, XML document).
2927 Some access requesters may request **access** to an entire subtree of a **resource** specified by a
2928 node. XACML allows the **PEP** (or **context handler**) to specify whether the **decision request** is
2929 just for a single **resource** or for a subtree below the specified **resource**. The latter is equivalent to
2930 repeating a single request for each node in the entire subtree. When a request **context** contains a
2931 resource attribute of type

2932 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2933 with a value of "Immediate", or if it does not contain that **attribute**, then the **decision request**
2934 SHALL be interpreted to apply to just the single **resource** specified by the
2935 "urn:oasis:names:tc:xacml:1.0:resource:resource-id" **attribute**.

2936 When the

2937 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2938 **attribute** has the value "Children", the **decision request** SHALL be interpreted to apply to the
2939 specified **resource** and its immediate children **resources**.

2940 When the

2941 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2942 **attribute** has the value "Descendants", the **decision request** SHALL be interpreted to apply to
2943 both the specified **resource** and all its descendant **resources**.

2944 In the case of "Children" and "Descendants", the **authorization decision** MAY include multiple
2945 results for the multiple sub-nodes in the **resource** sub-tree.

2946 An XACML **authorization response** MAY contain multiple <Result> elements.

2947 Note that the method by which the **PDP** discovers whether the **resource** is hierarchically organized
2948 or not is outside the scope of XACML.

2949 In the case where a child or descendant **resource** cannot be accessed, the <Result> element
2950 associated with the parent element SHALL contain a <StatusCode> Value of
2951 "urn:oasis:names:tc:xacml:1.0:status:processing-error".

2952 7.9. Attributes

2953 **Attributes** are specified in the request **context**, regardless of whether or not they appeared in the
2954 original **decision request**, and are referred to in the **policy** by **subject**, **resource**, **action** and
2955 **environment attribute** designators and **attribute** selectors. A *named attribute* is the term used for
2956 the criteria that the specific **subject**, **resource**, **action** and **environment attribute** designators and
2957 selectors use to refer to **attributes** in the **subject**, **resource**, **action** and **environment** elements of
2958 the request **context**, respectively.

2959

7.9.1. Attribute Matching

2960 A *named attribute* has specific criteria with which to match *attributes* in the *context*. An *attribute*
2961 specifies *AttributeId*, *DataType* and *Issuer* attributes, and each *named attribute* also
2962 specifies *AttributeId*, *DataType* and optional *Issuer* attributes. A *named attribute* SHALL
2963 match an *attribute* if the values of their respective *AttributeId*, *DataType* and optional *Issuer*
2964 attributes match within their particular element, e.g. *subject*, *resource*, *action* or *environment*, of
2965 the *context*. The *AttributeId* of the named *attribute* MUST match, by URI equality, the
2966 *AttributeId* of the context *attribute*. The *DataType* of the named *attribute* MUST match, by
2967 URI equality, the *DataType* of the same context *attribute*. If *Issuer* is supplied in the named
2968 *attribute*, then it MUST match, by URI equality, the *Issuer* of the same context *attribute*. If
2969 *Issuer* is not supplied in the *named attribute*, then the matching of the context *attribute* to the
2970 *named attribute* SHALL be governed by *AttributeId* and *DataType* alone, regardless of the
2971 presence, absence, or actual value of *Issuer*. In the case of an *attribute* selector, the matching
2972 of the *attribute* to the *named attribute* SHALL be governed by the XPath expression and
2973 *DataType*.

2974

7.9.2. Attribute Retrieval

2975 The *PDP* SHALL request the values of *attributes* in the request *context* from the *context handler*.
2976 The *PDP* SHALL reference the *attributes* as if they were in a physical request *context* document,
2977 but the *context handler* is responsible for obtaining and supplying the requested values. The
2978 *context handler* SHALL return the values of *attributes* that match the *attribute* designator or
2979 *attribute* selector and form them into a *bag* of values with the specified data-type. If no *attributes*
2980 from the request *context* match, then the *attribute* SHALL be considered missing. If the *attribute*
2981 is missing, then *MustBePresent* governs whether the *attribute* designator or *attribute* selector
2982 returns an empty *bag* or an "Indeterminate" result. If *MustBePresent* is "False" (default value),
2983 then a missing *attribute* SHALL result in an empty *bag*. If *MustBePresent* is "True", then a
2984 missing *attribute* SHALL result in "Indeterminate". This "Indeterminate" result SHALL be handled
2985 in accordance with the specification of the encompassing expressions, *rules*, *policies* and *policy*
2986 *sets*. If the result is "Indeterminate", then the *AttributeId*, *DataType* and *Issuer* of the
2987 *attribute* MAY be listed in the *authorization decision* as described in Section 7.10. However, a
2988 *PDP* MAY choose not to return such information for security reasons.

2989

7.9.3. Environment Attributes

2990 *Environment attributes* are listed in Section B.8. If a value for one of these *attributes* is supplied
2991 in the *decision request*, then the *context handler* SHALL use that value. Otherwise, the *context*
2992 *handler* SHALL supply a value. For the date and time *attributes*, the supplied value SHALL have
2993 the semantics of "date and time that apply to the *decision request*".

2994

7.10. Authorization decision

2995 Given a valid XACML *policy* or *policy set*, a compliant XACML *PDP* MUST evaluate the *policy* as
2996 specified in Sections 5, 0 and 4.2. The *PDP* MUST return a response *context*, with one
2997 <Decision> element of value "Permit", "Deny", "Indeterminate" or "NotApplicable".

2998 If the *PDP* cannot make a decision, then an "Indeterminate" <Decision> element contents SHALL
2999 be returned. The *PDP* MAY return a <Decision> element contents of "Indeterminate" with a
3000 status code of:

3001

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

3002 signifying that more information is needed. In this case, the <Status> element MAY list the
3003 names and data-types of any **attributes** of the **subjects** and the **resource** that are needed by the
3004 **PDP** to refine its decision. A **PEP** MAY resubmit a refined request **context** in response to a
3005 <Decision> element contents of "Indeterminate" with a status code of

3006 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3007 by adding **attribute** values for the **attribute** names that were listed in the previous response. When
3008 the **PDP** returns a <Decision> element contents of "Indeterminate", with a status code of

3009 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3010 it MUST NOT list the names and data-types of any **attribute** of the **subject** or the **resource** for
3011 which values were supplied in the original request. Note, this requirement forces the **PDP** to
3012 eventually return an **authorization decision** of "Permit", "Deny" or "Indeterminate" with some other
3013 status code, in response to successively-refined requests.

3014 7.11. Obligations

3015 A **policy** or **policy set** may contain one or more **obligations**. When such a **policy** or **policy set** is
3016 evaluated, an **obligation** SHALL be passed up to the next level of evaluation (the enclosing or
3017 referencing **policy set** or **authorization decision**) only if the **effect** of the **policy** or **policy set**
3018 being evaluated matches the value of the `xacml:FulfillOn` attribute of the **obligation**.

3019
3020 As a consequence of this procedure, no **obligations** SHALL be returned to the **PEP** if the **policies**
3021 or **policy sets** from which they are drawn are not evaluated, or if their evaluated result is
3022 "Indeterminate" or "NotApplicable", or if the **decision** resulting from evaluating the **policy** or **policy**
3023 **set** does not match the **decision** resulting from evaluating an enclosing **policy set**.

3024
3025 If the **PDP's** evaluation is viewed as a tree of **policy sets** and **policies**, each of which returns
3026 "Permit" or "Deny", then the set of **obligations** returned by the **PDP** to the **PEP** will include only the
3027 **obligations** associated with those paths where the **effect** at each level of evaluation is the same as
3028 the **effect** being returned by the **PDP**.

3029 A **PEP** that receives a valid XACML response of "Permit" with **obligations** SHALL be responsible
3030 for fulfilling *all* of those **obligations**. A **PEP** that receives an XACML response of "Deny" with
3031 **obligations** SHALL be responsible for fulfilling all of the **obligations** that it *understands*.

3032 7.12. Unsupported functionality

3033 If the **PDP** attempts to evaluate a **policy set** or **policy** that contains an optional element type or
3034 feature that the **PDP** does not support, then the **PDP** SHALL return a <Decision> value of
3035 "Indeterminate". If a <StatusCode> element is also returned, then its value SHALL be
3036 "urn:oasis:names:tc:xacml:1.0:status:syntax-error" in the case of an unsupported element type, and
3037 "urn:oasis:names:tc:xacml:1.0:status:processing-error" in the case of an unsupported feature.

3038 7.13. Syntax and type errors

3039 If a **policy** that contains invalid syntax is evaluated by the XACML **PDP** at the time a **decision**
3040 **request** is received, then the result of that **policy** SHALL be "Indeterminate" with a StatusCode
3041 value of "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

3042 If a **policy** that contains invalid static data-types is evaluated by the XACML **PDP** at the time a
3043 **decision request** is received, then the result of that **policy** SHALL be "Indeterminate" with a
3044 StatusCode value of "urn:oasis:names:tc:xacml:1.0:status:processing-error".

3045

8. XACML extensibility points (non-normative)

3046
3047

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

3048

8.1. Extensible XML attribute types

3049
3050

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

3051

AttributeId,

3052

AttributeValue,

3053

DataType,

3054

FunctionId,

3055

MatchId,

3056

ObligationId,

3057

PolicyCombiningAlgId,

3058

RuleCombiningAlgId,

3059

StatusCode,

3060

SubjectCategory.

3061

See Section 5 for definitions of these attribute types.

3062

8.2. Structured attributes

3063
3064
3065

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

3066
3067
3068
3069
3070
3071
3072

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

3073
3074
3075

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

3076

9. Security and privacy considerations (non-normative)

3077

3078 This section identifies possible security and privacy compromise scenarios that should be
3079 considered when implementing an XACML-based system. The section is informative only. It is left
3080 to the implementer to decide whether these compromise scenarios are practical in their
3081 environment and to select appropriate safeguards.

3082

9.1. Threat model

3083 We assume here that the adversary has access to the communication channel between the
3084 XACML actors and is able to interpret, insert, delete and modify messages or parts of messages.

3085 Additionally, an actor may use information from a former transaction maliciously in subsequent
3086 transactions. It is further assumed that *rules* and *policies* are only as reliable as the actors that
3087 create and use them. Thus it is incumbent on each actor to establish appropriate trust in the other
3088 actors upon which it relies. Mechanisms for trust establishment are outside the scope of this
3089 specification.

3090 The messages that are transmitted between the actors in the XACML model are susceptible to
3091 attack by malicious third parties. Other points of vulnerability include the *PEP*, the *PDP* and the
3092 *PAP*. While some of these entities are not strictly within the scope of this specification, their
3093 compromise could lead to the compromise of *access control* enforced by the *PEP*.

3094 It should be noted that there are other components of a distributed system that may be
3095 compromised, such as an operating system and the domain-name system (DNS) that are outside
3096 the scope of this discussion of threat models. Compromise in these components may also lead to a
3097 policy violation.

3098 The following sections detail specific compromise scenarios that may be relevant to an XACML
3099 system.

3100

9.1.1. Unauthorized disclosure

3101 XACML does not specify any inherent mechanisms for confidentiality of the messages exchanged
3102 between actors. Therefore, an adversary could observe the messages in transit. Under certain
3103 security policies, disclosure of this information is a violation. Disclosure of *attributes* or the types
3104 of *decision requests* that a *subject* submits may be a breach of privacy policy. In the commercial
3105 sector, the consequences of unauthorized disclosure of personal data may range from
3106 embarrassment to the custodian to imprisonment and large fines in the case of medical or financial
3107 data.

3108 Unauthorized disclosure is addressed by confidentiality mechanisms.

3109

9.1.2. Message replay

3110 A message replay attack is one in which the adversary records and replays legitimate messages
3111 between XACML actors. This attack may lead to denial of service, the use of out-of-date
3112 information or impersonation.

3113 Prevention of replay attacks requires the use of message freshness mechanisms.

3114 Note that encryption of the message does not mitigate a replay attack since the message is just
3115 replayed and does not have to be understood by the adversary.

3116

9.1.3. Message insertion

3117 A message insertion attack is one in which the adversary inserts messages in the sequence of
3118 messages between XACML actors.

3119 The solution to a message insertion attack is to use mutual authentication and a message
3120 sequence integrity mechanism between the actors. It should be noted that just using SSL mutual
3121 authentication is not sufficient. This only proves that the other party is the one identified by the
3122 subject of the X.509 certificate. In order to be effective, it is necessary to confirm that the certificate
3123 subject is authorized to send the message.

3124

9.1.4. Message deletion

3125 A message deletion attack is one in which the adversary deletes messages in the sequence of
3126 messages between XACML actors. Message deletion may lead to denial of service. However, a
3127 properly designed XACML system should not render an incorrect authorization decision as a result
3128 of a message deletion attack.

3129 The solution to a message deletion attack is to use a message integrity mechanism between the
3130 actors.

3131

9.1.5. Message modification

3132 If an adversary can intercept a message and change its contents, then they may be able to alter an
3133 **authorization decision**. Message integrity mechanisms can prevent a successful message
3134 modification attack.

3135

9.1.6. NotApplicable results

3136 A result of "NotApplicable" means that the **PDP** did not have a policy whose target matched the
3137 information in the **decision request**. In general, we highly recommend using a "default-deny"
3138 policy, so that when a **PDP** would have returned "NotApplicable", a result of "Deny" is returned
3139 instead.

3140 In some security models, however, such as is common in many Web Servers, a result of
3141 "NotApplicable" is treated as equivalent to "Permit". There are particular security considerations
3142 that must be taken into account for this to be safe. These are explained in the following
3143 paragraphs.

3144 If "NotApplicable" is to be treated as "Permit", it is vital that the matching algorithms used by the
3145 policy to match elements in the decision request are closely aligned with the data syntax used by
3146 the applications that will be submitting the decision request. A failure to match will be treated as
3147 "Permit", so an unintended failure to match may allow unintended access.

3148 A common example of this is a Web Server. Commercial http responders allow a variety of
3149 syntaxes to be treated equivalently. The "%" can be used to represent characters by hex value.
3150 The URL path "/./" provides multiple ways of specifying the same value. Multiple character sets
3151 may be permitted and, in some cases, the same printed character can be represented by different
3152 binary values. Unless the matching algorithm used by the policy is sophisticated enough to catch
3153 these variations, unintended access may be permitted.

3154 It is safe to treat "NotApplicable" as "Permit" only in a closed environment where all applications
3155 that formulate a decision request can be guaranteed to use the exact syntax expected by the
3156 policies used by the **PDP**. In a more open environment, where decision requests may be received
3157 from applications that may use any legal syntax, it is strongly recommended that "NotApplicable"
3158 NOT be treated as "Permit" unless matching rules have been very carefully designed to match all
3159 possible applicable inputs, regardless of syntax or type variations.

3160

9.1.7. Negative rules

3161 A negative **rule** is one that is based on a **predicate** not being "True". If not used with care,
3162 negative **rules** can lead to policy violation, therefore some authorities recommend that they not be
3163 used. However, negative **rules** can be extremely efficient in certain cases, so XACML has chosen
3164 to include them. Nevertheless, it is recommended that they be used with care and avoided if
3165 possible.

3166 A common use for negative **rules** is to deny **access** to an individual or subgroup when their
3167 membership in a larger group would otherwise permit them access. For example, we might want to
3168 write a **rule** that allows all Vice Presidents to see the unpublished financial data, except for Joe,
3169 who is only a Ceremonial Vice President and can be indiscreet in his communications. If we have
3170 complete control of the administration of **subject attributes**, a superior approach would be to
3171 define "Vice President" and "Ceremonial Vice President" as distinct groups and then define **rules**
3172 accordingly. However, in some environments this approach may not be feasible. (It is worth noting
3173 in passing that, generally speaking, referring to individuals in **rules** does not scale well. Generally,
3174 shared **attributes** are preferred.)

3175 If not used with care, negative **rules** can lead to policy violation in two common cases. They are:
3176 when **attributes** are suppressed and when the base group changes. An example of suppressed
3177 **attributes** would be if we have a policy that **access** should be permitted, *unless* the **subject** is a
3178 credit risk. If it is possible that the **attribute** of being a credit risk may be unknown to the **PDP** for
3179 some reason, then unauthorized **access** may be permitted. In some environments, the **subject**
3180 may be able to suppress the publication of **attributes** by the application of privacy controls, or the
3181 server or repository that contains the information may be unavailable for accidental or intentional
3182 reasons.

3183 An example of a changing base group would be if there is a policy that everyone in the engineering
3184 department may change software source code, except for secretaries. Suppose now that the
3185 department was to merge with another engineering department and the intent is to maintain the
3186 same policy. However, the new department also includes individuals identified as administrative
3187 assistants, who ought to be treated in the same way as secretaries. Unless the policy is altered,
3188 they will unintentionally be permitted to change software source code. Problems of this type are
3189 easy to avoid when one individual administers all **policies**, but when administration is distributed,
3190 as XACML allows, this type of situation must be explicitly guarded against.

3191 9.2. Safeguards

3192 9.2.1. Authentication

3193 Authentication provides the means for one party in a transaction to determine the identity of the
3194 other party in the transaction. Authentication may be in one direction, or it may be bilateral.

3195 Given the sensitive nature of **access control** systems, it is important for a **PEP** to authenticate the
3196 identity of the **PDP** to which it sends **decision requests**. Otherwise, there is a risk that an
3197 adversary could provide false or invalid **authorization decisions**, leading to a policy violation.

3198 It is equally important for a **PDP** to authenticate the identity of the **PEP** and assess the level of trust
3199 to determine what, if any, sensitive data should be passed. One should keep in mind that even
3200 simple "Permit" or "Deny" responses could be exploited if an adversary were allowed to make
3201 unlimited requests to a **PDP**.

3202 Many different techniques may be used to provide authentication, such as co-located code, a
3203 private network, a VPN or digital signatures. Authentication may also be performed as part of the
3204 communication protocol used to exchange the **contexts**. In this case, authentication may be
3205 performed at the message level or at the session level.

3206

9.2.2. Policy administration

3207 If the contents of *policies* are exposed outside of the *access control* system, potential *subjects*
3208 may use this information to determine how to gain unauthorized *access*.

3209 To prevent this threat, the repository used for the storage of *policies* may itself require *access*
3210 *control*. In addition, the <Status> element should be used to return values of missing *attributes*
3211 only when exposure of the identities of those *attributes* will not compromise security.

3212

9.2.3. Confidentiality

3213 Confidentiality mechanisms ensure that the contents of a message can be read only by the desired
3214 recipients and not by anyone else who encounters the message while it is in transit. There are two
3215 areas in which confidentiality should be considered: one is confidentiality during transmission; the
3216 other is confidentiality within a <Policy> element.

3217

9.2.3.1. Communication confidentiality

3218 In some environments it is deemed good practice to treat all data within an *access control* system
3219 as confidential. In other environments, *policies* may be made freely available for distribution,
3220 inspection and audit. The idea behind keeping *policy* information secret is to make it more difficult
3221 for an adversary to know what steps might be sufficient to obtain unauthorized *access*. Regardless
3222 of the approach chosen, the security of the *access control* system should not depend on the
3223 secrecy of the *policy*.

3224 Any security concerns or requirements related to transmitting or exchanging XACML <Policy>
3225 elements are outside the scope of the XACML standard. While it is often important to ensure that
3226 the integrity and confidentiality of <Policy> elements is maintained when they are exchanged
3227 between two parties, it is left to the implementers to determine the appropriate mechanisms for their
3228 environment.

3229 Communications confidentiality can be provided by a confidentiality mechanism, such as SSL.
3230 Using a point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points
3231 is compromised.

3232

9.2.3.2. Statement level confidentiality

3233 In some cases, an implementation may want to encrypt only parts of an XACML <Policy>
3234 element.

3235 The XML Encryption Syntax and Processing Candidate Recommendation from W3C can be used
3236 to encrypt all or parts of an XML document. This specification is recommended for use with
3237 XACML.

3238 It should go without saying that if a repository is used to facilitate the communication of cleartext
3239 (i.e., unencrypted) *policy* between the *PAP* and *PDP*, then a secure repository should be used to
3240 store this sensitive data.

3241

9.2.4. Policy integrity

3242 The XACML *policy*, used by the *PDP* to evaluate the request *context*, is the heart of the system.
3243 Therefore, maintaining its integrity is essential. There are two aspects to maintaining the integrity of
3244 the *policy*. One is to ensure that <Policy> elements have not been altered since they were
3245 originally created by the *PAP*. The other is to ensure that <Policy> elements have not been
3246 inserted or deleted from the set of *policies*.

3247 In many cases, both aspects can be achieved by ensuring the integrity of the actors and
3248 implementing session-level mechanisms to secure the communication between actors. The
3249 selection of the appropriate mechanisms is left to the implementers. However, when **policy** is
3250 distributed between organizations to be acted on at a later time, or when the **policy** travels with the
3251 protected resource, it would be useful to sign the **policy**. In these cases, the XML Signature
3252 Syntax and Processing standard from W3C is recommended to be used with XACML.

3253 Digital signatures should only be used to ensure the integrity of the statements. Digital signatures
3254 should not be used as a method of selecting or evaluating **policy**. That is, the **PDP** should not
3255 request a **policy** based on who signed it or whether or not it has been signed (as such a basis for
3256 selection would, itself, be a matter of policy). However, the **PDP** must verify that the key used to
3257 sign the **policy** is one controlled by the purported issuer of the **policy**. The means to do this are
3258 dependent on the specific signature technology chosen and are outside the scope of this document.

3259 **9.2.5. Policy identifiers**

3260 Since **policies** can be referenced by their identifiers, it is the responsibility of the **PAP** to ensure
3261 that these are unique. Confusion between identifiers could lead to misidentification of the
3262 **applicable policy**. This specification is silent on whether a **PAP** must generate a new identifier
3263 when a **policy** is modified or may use the same identifier in the modified **policy**. This is a matter of
3264 administrative practice. However, care must be taken in either case. If the identifier is reused,
3265 there is a danger that other **policies** or **policy sets** that reference it may be adversely affected.
3266 Conversely, if a new identifier is used, these other **policies** may continue to use the prior **policy**,
3267 unless it is deleted. In either case the results may not be what the **policy** administrator intends.

3268 **9.2.6. Trust model**

3269 Discussions of authentication, integrity and confidentiality mechanisms necessarily assume an
3270 underlying trust model: how can one actor come to believe that a given key is uniquely associated
3271 with a specific, identified actor so that the key can be used to encrypt data for that actor or verify
3272 signatures (or other integrity structures) from that actor? Many different types of trust model exist,
3273 including strict hierarchies, distributed authorities, the Web, the bridge and so on.

3274 It is worth considering the relationships between the various actors of the **access control** system in
3275 terms of the interdependencies that do and do not exist.

- 3276 • None of the entities of the authorization system are dependent on the **PEP**. They may
3277 collect data from it, for example authentication, but are responsible for verifying it.
- 3278 • The correct operation of the system depends on the ability of the **PEP** to actually enforce
3279 **policy** decisions.
- 3280 • The **PEP** depends on the **PDP** to correctly evaluate **policies**. This in turn implies that the
3281 **PDP** is supplied with the correct inputs. Other than that, the **PDP** does not depend on the
3282 **PEP**.
- 3283 • The **PDP** depends on the **PAP** to supply appropriate policies. The **PAP** is not dependent
3284 on other components.

3285 **9.2.7. Privacy**

3286 It is important to be aware that any transactions that occur with respect to **access control** may
3287 reveal private information about the actors. For example, if an XACML **policy** states that certain
3288 data may only be read by **subjects** with "Gold Card Member" status, then any transaction in which
3289 a **subject** is permitted **access** to that data leaks information to an adversary about the **subject's**
3290 status. Privacy considerations may therefore lead to encryption and/or to **access control policies**

3291 surrounding the enforcement of XACML **policy** instances themselves: confidentiality-protected
3292 channels for the request/response protocol messages, protection of **subject attributes** in storage
3293 and in transit, and so on.

3294 Selection and use of privacy mechanisms appropriate to a given environment are outside the scope
3295 of XACML. The decision regarding whether, how and when to deploy such mechanisms is left to
3296 the implementers associated with the environment.

3297 **10. Conformance (normative)**

3298 **10.1. Introduction**

3299 The XACML specification addresses the following aspect of conformance:

3300 The XACML specification defines a number of functions, etc. that have somewhat specialist
3301 application, therefore they are not required to be implemented in an implementation that claims to
3302 conform with the OASIS standard.

3303 **10.2. Conformance tables**

3304 This section lists those portions of the specification that **MUST** be included in an implementation of
3305 a **PDP** that claims to conform with XACML v1.0. A set of test cases has been created to assist in
3306 this process. These test cases are hosted by Sun Microsystems and can be located from the
3307 XACML Web page. The site hosting the test cases contains a full description of the test cases and
3308 how to execute them.

3309 Note: "M" means mandatory-to-implement. "O" means optional.

3310 **10.2.1. Schema elements**

3311 The implementation **MUST** support those schema elements that are marked "M".

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

xacml:AnySubject	M
xacml:Apply	M
xacml:AttributeAssignment	O
xacml:AttributeSelector	O
xacml:AttributeValue	M
xacml:Condition	M
xacml:Description	M
xacml:EnvironmentAttributeDesignator	M
xacml:Function	M
xacml:Obligation	O
xacml:Obligations	O
xacml:Policy	M
xacml:PolicyDefaults	O
xacml:PolicyIdReference	M
xacml:PolicySet	M
xacml:PolicySetDefaults	O
xacml:PolicySetIdReference	M
xacml:Resource	M
xacml:ResourceAttributeDesignator	M
xacml:ResourceMatch	M
xacml:Resources	M
xacml:Rule	M
xacml:Subject	M
xacml:SubjectMatch	M
xacml:Subjects	M
xacml:Target	M
xacml:XPathVersion	O

3312 **10.2.2. Identifier Prefixes**

3313 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

3314 **10.2.3. Algorithms**

3315 The implementation MUST include the rule- and policy-combining algorithms associated with the
3316 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

3317 **10.2.4. Status Codes**

3318 Implementation support for the urn:oasis:names:tc:xacml:1.0:context:status element is optional, but
 3319 if the element is supported, then the following status codes must be supported and must be used in
 3320 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

3321 **10.2.5. Attributes**

3322 The implementation MUST support the attributes associated with the following attribute identifiers
 3323 as specified by XACML. If values for these **attributes** are not present in the **decision request**,
 3324 then their values MUST be supplied by the **PDP**. So, unlike most other **attributes**, their semantics
 3325 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

3326 **10.2.6. Identifiers**

3327 The implementation MUST use the attributes associated with the following identifiers in the way
 3328 XACML has defined. This requirement pertains primarily to implementations of a **PAP** or **PEP** that
 3329 use XACML, since the semantics of the attributes are transparent to the **PDP**.

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

urn:oasis:names:tc:xacml:1.0:action:implied-action	M
--	---

3330 **10.2.7. Data-types**

3331 The implementation MUST support the data-types associated with the following identifiers marked
3332 "M".

Data-type	M/O
http://www.w3.org/2001/XMLSchema#string	M
http://www.w3.org/2001/XMLSchema#boolean	M
http://www.w3.org/2001/XMLSchema#integer	M
http://www.w3.org/2001/XMLSchema#double	M
http://www.w3.org/2001/XMLSchema#time	M
http://www.w3.org/2001/XMLSchema#date	M
http://www.w3.org/2001/XMLSchema#dateTime	M
http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration	M
http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration	M
http://www.w3.org/2001/XMLSchema#anyURI	M
http://www.w3.org/2001/XMLSchema#hexBinary	M
http://www.w3.org/2001/XMLSchema#base64Binary	M
urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name	M
urn:oasis:names:tc:xacml:1.0:data-type:x500Name	M

3333 **10.2.8. Functions**

3334 The implementation MUST properly process those functions associated with the identifiers marked
3335 with an "M".

Function	M/O
urn:oasis:names:tc:xacml:1.0:function:string-equal	M
urn:oasis:names:tc:xacml:1.0:function:boolean-equal	M
urn:oasis:names:tc:xacml:1.0:function:integer-equal	M
urn:oasis:names:tc:xacml:1.0:function:double-equal	M
urn:oasis:names:tc:xacml:1.0:function:date-equal	M
urn:oasis:names:tc:xacml:1.0:function:time-equal	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-equal	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-equal	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-equal	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-equal	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-equal	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal	M
urn:oasis:names:tc:xacml:1.0:function:integer-add	M
urn:oasis:names:tc:xacml:1.0:function:double-add	M
urn:oasis:names:tc:xacml:1.0:function:integer-subtract	M
urn:oasis:names:tc:xacml:1.0:function:double-subtract	M
urn:oasis:names:tc:xacml:1.0:function:integer-multiply	M
urn:oasis:names:tc:xacml:1.0:function:double-multiply	M
urn:oasis:names:tc:xacml:1.0:function:integer-divide	M
urn:oasis:names:tc:xacml:1.0:function:double-divide	M
urn:oasis:names:tc:xacml:1.0:function:integer-mod	M
urn:oasis:names:tc:xacml:1.0:function:integer-abs	M
urn:oasis:names:tc:xacml:1.0:function:double-abs	M
urn:oasis:names:tc:xacml:1.0:function:round	M

urn:oasis:names:tc:xacml:1.0:function:floor	M
urn:oasis:names:tc:xacml:1.0:function:string-normalize-space	M
urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case	M
urn:oasis:names:tc:xacml:1.0:function:double-to-integer	M
urn:oasis:names:tc:xacml:1.0:function:integer-to-double	M
urn:oasis:names:tc:xacml:1.0:function:or	M
urn:oasis:names:tc:xacml:1.0:function:and	M
urn:oasis:names:tc:xacml:1.0:function:n-of	M
urn:oasis:names:tc:xacml:1.0:function:not	M
urn:oasis:names:tc:xacml:1.0:function: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

3336

11. References

3337

- 3338 [DS] D. Eastlake et al., *XML-Signature Syntax and Processing*,
3339 <http://www.w3.org/TR/xmldsig-core/>, World Wide Web Consortium.
- 3340 [Haskell] Haskell, a purely functional language. Available at
3341 <http://www.haskell.org/>
- 3342 [Hinton94] Hinton, H, M, Lee,, E, S, The Compatibility of Policies, Proceedings 2nd
3343 ACM Conference on Computer and Communications Security, Nov 1994,
3344 Fairfax, Virginia, USA.
- 3345 [IEEE754] IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-
3346 7653-8, IEEE Product No. SH10116-TBR
- 3347 [Kudo00] Kudo M and Hada S, XML document security based on provisional
3348 authorization, Proceedings of the Seventh ACM Conference on Computer
3349 and Communications Security, Nov 2000, Athens, Greece, pp 87-96.
- 3350 [LDAP-1] RFC2256, A summary of the X500(96) User Schema for use with LDAPv3,
3351 Section 5, M Wahl, December 1997 <http://www.ietf.org/rfc/rfc2798.txt>
- 3352 [LDAP-2] RFC2798, Definition of the inetOrgPerson, M. Smith, April 2000
3353 <http://www.ietf.org/rfc/rfc2798.txt>
- 3354 [MathML] Mathematical Markup Language (MathML), Version 2.0, W3C
3355 Recommendation, 21 February 2001. Available at:
3356 <http://www.w3.org/TR/MathML2/>
- 3357 [Perritt93] Perritt, H. Knowbots, Permissions Headers and Contract Law, Conference
3358 on Technological Strategies for Protecting Intellectual Property in the
3359 Networked Multimedia Environment, April 1993. Available at:
3360 <http://www.ifla.org/documents/infopol/copyright/perh2.txt>
- 3361 [RBAC] Role-Based Access Controls, David Ferraiolo and Richard Kuhn, 15th
3362 National Computer Security Conference, 1992. Available at:
3363 <http://csrc.nist.gov/rbac>
- 3364 [RegEx] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001,
3365 Appendix D. Available at: <http://www.w3.org/TR/xmlschema-0/>
- 3366 [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
3367 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997
- 3368 [SAML] Security Assertion Markup Language available from [http://www.oasis-](http://www.oasis-open.org/committees/security/#documents)
3369 [open.org/committees/security/#documents](http://www.oasis-open.org/committees/security/#documents)

3370	[Sloman94]	Sloman, M. Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Volume 2, part 4. Plenum Press. 1994.
3371		
3372		
3373	[XF]	XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft 16 August 2002. Available at: http://www.w3.org/TR/2002/WD-xquery-operators-20020816
3374		
3375		
3376	[XS]	XML Schema, parts 1 and 2. Available at: http://www.w3.org/TR/xmlschema-1/ and http://www.w3.org/TR/xmlschema-2/
3377		
3378		
3379	[XPath]	XML Path Language (XPath), Version 1.0, W3C Recommendation 16 November 1999. Available at: http://www.w3.org/TR/xpath
3380		
3381	[XSLT]	XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999. Available at: http://www.w3.org/TR/xslt
3382		
3383		

3384 **Appendix A. Standard data-types, functions and**
3385 **their semantics (normative)**

3386 **A.1. Introduction**

3387 This section contains a specification of the data-types and functions used in XACML to create
3388 **predicates** for a **rule's condition** and **target** matches.

3389 This specification combines the various standards set forth by IEEE and ANSI for string
3390 representation of numeric values, as well as the evaluation of arithmetic functions.

3391 This section describes the primitive data-types, **bags** and construction of expressions using
3392 XACML constructs. Finally, each standard function is named and its operational semantics are
3393 described.

3394 **A.2. Primitive types**

3395 Although XML instances represent all data-types as strings, an XACML **PDP** must reason about
3396 types of data that, while they have string representations, are not just strings. Types such as
3397 boolean, integer and double **MUST** be converted from their XML string representations to values
3398 that can be compared with values in their domain of discourse, such as numbers. The following
3399 primitive data-types are specified for use with XACML and have explicit data representations:

- 3400 • <http://www.w3.org/2001/XMLSchema#string>
- 3401 • <http://www.w3.org/2001/XMLSchema#boolean>
- 3402 • <http://www.w3.org/2001/XMLSchema#integer>
- 3403 • <http://www.w3.org/2001/XMLSchema#double>
- 3404 • <http://www.w3.org/2001/XMLSchema#time>
- 3405 • <http://www.w3.org/2001/XMLSchema#date>
- 3406 • <http://www.w3.org/2001/XMLSchema#dateTime>
- 3407 • <http://www.w3.org/2001/XMLSchema#anyURI>
- 3408 • <http://www.w3.org/2001/XMLSchema#hexBinary>
- 3409 • <http://www.w3.org/2001/XMLSchema#base64Binary>
- 3410 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration>
- 3411 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration>
- 3412 • <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
- 3413 • <urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name>

3414 A.3. Structured types

3415 An XACML `<AttributeValue>` element MAY contain an instance of a structured XML data-type,
3416 for example `<ds:KeyInfo>`. XACML 1.0 supports several ways for comparing such
3417 `<AttributeValue>` elements.

3418 1. In some cases, such an `<AttributeValue>` element MAY be compared using one of the
3419 XACML string functions, such as “regexp-string-match”, described below. This requires
3420 that the structured data `<AttributeValue>` be given the
3421 `DataType="http://www.w3.org/2001/XMLSchema#string"`. For example, a structured data-
3422 type that is actually a `ds:KeyInfo/KeyName` would appear in the Context as:

```
3423 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3424 <ds:KeyName>jhibbert-key</ds:KeyName>  
3425 </AttributeValue>
```

3426 In general, this method will not be adequate unless the structured data-type is quite simple.

3427 2. An `<AttributeSelector>` element MAY be used to select the value of a leaf sub-
3428 element of the structured data-type by means of an XPath expression. That value MAY
3429 then be compared using one of the supported XACML functions appropriate for its primitive
3430 data-type. This method requires support by the **PDP** for the optional XPath expressions
3431 feature.

3432 3. An `<AttributeSelector>` element MAY be used to select the value of any node in the
3433 structured data-type by means of an XPath expression. This node MAY then be compared
3434 using one of the XPath-based functions described in Section A14.13. This method requires
3435 support by the **PDP** for the optional XPath expressions and XPath functions features.

3436 A.4. Representations

3437 An XACML **PDP** SHALL be capable of converting string representations into various primitive data-
3438 types. For integers and doubles, XACML SHALL use the conversions described in [IEEE754].

3439 This document combines the various standards set forth by IEEE and ANSI for string
3440 representation of numeric values.

3441 XACML defines two additional data-types; these are “urn:oasis:names:tc:xacml:1.0:data-
3442 type:x500Name” and “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name”. These types
3443 represent identifiers for **subjects** and appear in several standard applications, such as TLS/SSL
3444 and electronic mail.

3445 The “urn:oasis:names:tc:xacml:1.0:data-type:x500Name” primitive type represents an X.500
3446 Distinguished Name. The string representation of an X.500 distinguished name is specified in IETF
3447 RFC 2253 “Lightweight Directory Access Protocol (v3): UTF-8 String Representation of
3448 Distinguished Names”.¹

3449 The “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name” primitive type represents electronic mail
3450 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.

3451 An RFC822 name consists of a *local-part* followed by "@" followed by a *domain-part*. The *local-*
3452 *part* is case-sensitive, while the *domain-part* (which is usually a DNS host name) is not case-
3453 sensitive.²

3454 A.5. Bags

3455 XACML defines implicit collections of its primitive types. XACML refers to a collection of values that
3456 are of a single primitive type as a **bag**. **Bags** of primitive types are needed because selections of
3457 nodes from an XML **resource** or XACML request **context** may return more than one value.

3458 The <AttributeSelector> element uses an XPath expression to specify the selection of data
3459 from an XML **resource**. The result of an XPath expression is termed a *node-set*, which contains all
3460 the leaf nodes from the XML **resource** that match the predicate in the XPath expression. Based on
3461 the various indexing functions provided in the XPath specification, it SHALL be implied that a
3462 resultant node-set is the collection of the matching nodes. XACML also defines the
3463 <AttributeDesignator> **element** to have the same matching methodology for attributes in the
3464 XACML request **context**.

3465 The values in a **bag** are not ordered, and some of the values may be duplicates. There SHALL be
3466 no notion of a **bag** containing **bags**, or a **bag** containing values of differing types. I.e. a **bag** in
3467 XACML SHALL contain only values that are of the same primitive type.

3468 A.6. Expressions

3469 XACML specifies expressions in terms of the following elements, of which the <Apply> and
3470 <Condition> elements recursively compose greater expressions. Valid expressions shall be type
3471 correct, which means that the types of each of the elements contained within <Apply> and
3472 <Condition> elements shall agree with the respective argument types of the function that is
3473 named by the FunctionId attribute. The resultant type of the <Apply> or <Condition>
3474 element shall be the resultant type of the function, which may be narrowed to a primitive data-type,
3475 or a bag of a primitive data-type, by type-unification. XACML defines an evaluation result of
3476 "Indeterminate", which is said to be the result of an invalid expression, or an operational error
3477 occurring during the evaluation of the expression.

3478 XACML defines the following elements to be legal XACML expressions:

- 3479 • <AttributeValue>
- 3480 • <SubjectAttributeDesignator>
- 3481 • <SubjectAttributeSelector>
- 3482 • <ResourceAttributeDesignator>
- 3483 • <ActionAttributeDesignator>
- 3484 • <EnvironmentAttributeDesignator>

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

- 3485 • <AttributeSelector>
- 3486 • <Apply>
- 3487 • <Condition>
- 3488 • <Function>

3489 A.7. Element <AttributeValue>

3490 The <AttributeValue> element SHALL represent an explicit value of a primitive type. For
 3491 example:

```

3492 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-equal">
3493   <AttributeValue
3494     DataType="http://www.w3.org/2001/XMLSchema#integer">123</AttributeValue>
3495   <AttributeValue
3496     DataType="http://www.w3.org/2001/XMLSchema#integer">123</AttributeValue>
3497 </Apply>
  
```

3498 A.8. Elements <AttributeDesignator> and 3499 <AttributeSelector>

3500 The <AttributeDesignator> and <AttributeSelector> elements SHALL evaluate to a **bag**
 3501 of a specific primitive type. The type SHALL be inferred from the function in which it appears. Each
 3502 element SHALL contain a URI or XPath expression, respectively, to identify the required **attribute**
 3503 values. If an operational error were to occur while finding the values, the value of the element
 3504 SHALL be set to "Indeterminate". If the required **attribute** cannot be located, then the value of the
 3505 element SHALL be set to an empty **bag** of the inferred primitive type.

3506 A.9. Element <Apply>

3507 XACML function calls are represented by the <Apply> element. The function to be applied is
 3508 named in the `FunctionId` attribute of this element. The value of the <Apply> element SHALL be
 3509 set to either a primitive data-type or a **bag** of a primitive type, whose data-type SHALL be inferred
 3510 from the `FunctionId`. The arguments of a function SHALL be the values of the XACML
 3511 expressions that are contained as ordered elements in an <Apply> element. The legal number of
 3512 arguments within an <Apply> element SHALL depend upon the `functionId`.

3513 A.10. Element <Condition>

3514 The <Condition> element MAY appear in the <Rule> element as the premise for emitting the
 3515 corresponding **effect** of the **rule**. The <Condition> element has the same structure as the
 3516 <Apply> element, with the restriction that its result SHALL be of data-type
 3517 "http://www.w3.org/2001/XMLSchema#boolean". The evaluation of the <Condition> element
 3518 SHALL follow the same evaluation semantics as those of the <Apply> element.

3519 A.11.Element <Function>

3520 The <Function> element names a standard XACML function or an extension function in its
3521 FunctionId attribute. The <Function> element MAY be used as an argument in functions that
3522 take a function as an argument.

3523 A.12.Matching elements

3524 Matching elements appear in the <Target> element of *rules*, *policies* and *policy sets*. They are
3525 the following:

3526 <SubjectMatch>

3527 <ResourceMatch>

3528 <ActionMatch>

3529 These elements represent boolean expressions over attributes of the subject, resource, and action,
3530 respectively. A matching element contains a MatchId attribute that specifies the function to be
3531 used in performing the match evaluation, an **attribute value**, and an <AttributeDesignator>
3532 or <AttributeSelector> element that specifies the **attribute** in the **context** that is to be
3533 matched against the specified value.

3534 The MatchId attribute SHALL specify a function that compares two arguments, returning a result
3535 type of "http://www.w3.org/2001/XMLSchema#boolean". The **attribute** value specified in the
3536 matching element SHALL be supplied to the MatchId function as its first argument. An element of
3537 the **bag** returned by the <AttributeDesignator> or <AttributeSelector> element SHALL
3538 be supplied to the MatchId function as its second argument. The data-type of the **attribute** value
3539 SHALL match the data-type of the first argument expected by the MatchId function. The data-type
3540 of the <AttributeDesignator> or <AttributeSelector> element SHALL match the data-
3541 type of the second argument expected by the MatchId function.

3542 The XACML standard functions that meet the requirements for use as a MatchId attribute value
3543 are:

3544 urn:oasis:names:tc:xacml:1.0:function:-type-equal

3545 urn:oasis:names:tc:xacml:1.0:function:-type-greater-than

3546 urn:oasis:names:tc:xacml:1.0:function:-type-greater-than-or-equal

3547 urn:oasis:names:tc:xacml:1.0:function:-type-less-than

3548 urn:oasis:names:tc:xacml:1.0:function:-type-less-than-or-equal

3549 urn:oasis:names:tc:xacml:1.0:function:-type-match

3550 In addition, functions that are strictly within an extension to XACML MAY appear as a value for the
3551 MatchId attribute, and those functions MAY use data-types that are also extensions, so long as
3552 the extension function returns a boolean result and takes an **attribute** value as its first argument
3553 and an <AttributeDesignator> or <AttributeSelector> as its second argument. The
3554 function used as the value for the MatchId attribute SHOULD be easily indexable. Use of non-
3555 indexable or complex functions may prevent efficient evaluation of **decision requests**.

3556 The evaluation semantics for a matching element is as follows. If an operational error were to
3557 occur while evaluating the <AttributeDesignator> or <AttributeSelector> element, then

3558 the result of the entire expression SHALL be "Indeterminate". If the <AttributeDesignator> or
3559 <AttributeSelector> element were to evaluate to an empty **bag**, then the result of the
3560 expression SHALL be "False". Otherwise, the MatchId function SHALL be applied between the
3561 explicit **attribute** value and each element of the **bag** returned from the <AttributeDesignator>
3562 or <AttributeSelector> element. If at least one of those function applications were to evaluate
3563 to "True", then the result of the entire expression SHALL be "True". Otherwise, if at least one of the
3564 function applications results in "Indeterminate", then the result SHALL be "Indeterminate". Finally,
3565 only if all function applications evaluate to "False", the result of the entire expression SHALL be
3566 "False".

3567 It is possible to express the semantics of a **target** matching element in a **condition**. For instance,
3568 the **target** match expression that compares a "subject-name" starting with the name "John" can be
3569 expressed as follows:

```
3570 <SubjectMatch  
3571   MatchId="urn:oasis:names:tc:xacml:1.0:function:regexp-string-match">  
3572   <SubjectAttributeDesignator  
3573     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3574     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3575   <AttributeValue  
3576     DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>  
3577 </SubjectMatch>
```

3578 Alternatively, the same match semantics can be expressed as an <Apply> element in a **condition**
3579 by using the "urn:oasis:names:tc:xacml:1.0:function:any-of" function, as follows:

```
3580 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">  
3581   <Function  
3582     FunctionId="urn:oasis:names:tc:xacml:1.0:function:regexp-string-match"/>  
3583   <AttributeValue  
3584     DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>  
3585   <SubjectAttributeDesignator  
3586     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3587     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3588 </Apply>
```

3589

3590 This expression of the semantics is NOT normative.

3591 A.13.Arithmetic evaluation

3592 IEEE 754 [IEEE 754] specifies how to evaluate arithmetic functions in a context, which specifies
3593 defaults for precision, rounding, etc. XACML SHALL use this specification for the evaluation of all
3594 integer and double functions relying on the *Extended Default Context*, enhanced with double
3595 precision:

3596 flags - all set to 0

3597 trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the "division-by-zero" trap
3598 enabler, which SHALL be set to 1

3599 precision - is set to the designated double precision

3600 rounding - is set to round-half-even (IEEE 854 §4.1)

3601 A.14.XACML standard functions

3602 XACML specifies the following functions that are prefixed with the
3603 "urn:oasis:names:tc:xacml:1.0:function:" relative name space identifier.

3604 A14.1 Equality predicates

3605 The following functions are the *equality* functions for the various primitive types. Each function for a
3606 particular data-type follows a specified standard convention for that data-type. If an argument of
3607 one of these functions were to evaluate to "Indeterminate", then the function SHALL be set to
3608 "Indeterminate".

- 3609 • string-equal

3610 This function SHALL take two arguments of "http://www.w3.org/2001/XMLSchema#string"
3611 and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". The function
3612 SHALL return "True" if and only if the value of both of its arguments are of equal length and
3613 each string is determined to be equal byte-by-byte according to the function "integer-equal".

- 3614 • boolean-equal

3615 This function SHALL take two arguments of
3616 "http://www.w3.org/2001/XMLSchema#boolean" and SHALL return "True" if and only if both
3617 values are equal.

- 3618 • integer-equal

3619 This function SHALL take two arguments of data-type
3620 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return an
3621 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on
3622 integers according to IEEE 754 [IEEE 754].

- 3623 • double-equal

3624 This function SHALL take two arguments of data-type
3625 "http://www.w3.org/2001/XMLSchema#double" and SHALL return an
3626 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on
3627 doubles according to IEEE 754 [IEEE 754].

- 3628 • date-equal

3629 This function SHALL take two arguments of data-type
3630 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
3631 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
3632 according to the "op:date-equal" function [XF Section 8.3.11].

- 3633 • time-equal

3634 This function SHALL take two arguments of data-type
3635 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
3636 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according
3637 to the "op:time-equal" function [XF Section 8.3.14].

- 3638 • dateTime-equal

3639 This function SHALL take two arguments of data-type
3640 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an

3641 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
3642 according to the "op:dateTime-equal" function [XF Section 8.3.8].

3643 • dayTimeDuration-equal

3644 This function SHALL take two arguments of data-type "http://www.w3.org/TR/2002/WD-
3645 xquery-operators-20020816#dayTimeDuration" and SHALL return an
3646 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
3647 according to the "op:dayTimeDuration-equal" function [XF Section 8.3.5]. Note that the
3648 lexical representation of each argument MUST be converted to a value expressed in
3649 fractional seconds [XF Section 8.2.2].

3650 • yearMonthDuration-equal

3651 This function SHALL take two arguments of data-type "http://www.w3.org/TR/2002/WD-
3652 xquery-operators-20020816#yearMonthDuration" and SHALL return an
3653 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
3654 according to the "op:yearMonthDuration-equal" function [XF Section 8.3.2]. Note that the
3655 lexical representation of each argument MUST be converted to a value expressed in
3656 integer months [XF Section 8.2.1].

3657 • anyURI-equal

3658 This function SHALL take two arguments of data-type
3659 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return an
3660 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
3661 according to the "op:anyURI-equal" function [XF Section 10.2.1].

3662 • x500Name-equal

3663 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-
3664 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It
3665 shall return "True" if and only if each Relative Distinguished Name (RDN) in the two
3666 arguments matches. Two RDNs shall be said to match if and only if the result of the
3667 following operations is "True"³.

- 3668 1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory
3669 Access Protocol (v3): UTF-8 String Representation of Distinguished Names".
- 3670 2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute
3671 ValuePairs in that RDN in ascending order when compared as octet strings
3672 (described in ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").
- 3673 3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key
3674 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section
3675 4.1.2.4 "Issuer".

3676 • rfc822Name-equal

3677 This function SHALL take two arguments of data-type "urn:oasis:names:tc:xacml:1.0:data-
3678 type:rfc822Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean".
3679 This function SHALL determine whether two "urn:oasis:names:tc:xacml:1.0:data-
3680 type:rfc822Name" arguments are equal. An RFC822 name consists of a *local-part* followed
3681 by "@" followed by a *domain-part*. The *local-part* is case-sensitive, while the *domain-part*
3682 (which is usually a DNS host name) is not case-sensitive. Perform the following
3683 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.

- 3684 1. Normalize the *domain*-part of each argument to lower case
- 3685 2. Compare the expressions by applying the function
- 3686 “urn:oasis:names:tc:xacml:1.0:function:string-equal” to the normalized arguments.
- 3687 • hexBinary-equal
- 3688 This function SHALL take two arguments of data-type
- 3689 “http://www.w3.org/2001/XMLSchema#hexBinary” and SHALL return an
- 3690 “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL return "True" if the
- 3691 octet sequences represented by the value of both arguments have equal length and are
- 3692 equal in a conjunctive, point-wise, comparison using the
- 3693 “urn:oasis:names:tc:xacml:1.0:function:integer-equal”. The conversion from the string
- 3694 representation to an octet sequence SHALL be as specified in [XS Section 8.2.15]
- 3695 • base64Binary-equal
- 3696 This function SHALL take two arguments of data-type
- 3697 “http://www.w3.org/2001/XMLSchema#base64Binary” and SHALL return an
- 3698 “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL return "True" if the
- 3699 octet sequences represented by the value of both arguments have equal length and are
- 3700 equal in a conjunctive, point-wise, comparison using the
- 3701 “urn:oasis:names:tc:xacml:1.0:function:integer-equal”. The conversion from the string
- 3702 representation to an octet sequence SHALL be as specified in [XS Section 8.2.16]

3703 **A14.2 Arithmetic functions**

3704 All of the following functions SHALL take two arguments of the specified *data-type*, integer or

3705 double, and SHALL return an element of integer or double data-type, respectively. However, the

3706 “add” functions MAY take more than two arguments. Each function evaluation SHALL proceed as

3707 specified by their logical counterparts in IEEE 754 [IEEE 754]. In an expression that contains any

3708 of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to

3709 "Indeterminate". In the case of the divide functions, if the divisor is zero, then the function SHALL

3710 evaluate to “Indeterminate”.

- 3711 • integer-add
- 3712 This function MAY have two or more arguments.
- 3713 • double-add
- 3714 This function MAY have two or more arguments.
- 3715 • integer-subtract
- 3716 • double-subtract
- 3717 • integer-multiply
- 3718 • double-multiply
- 3719 • integer-divide
- 3720 • double-divide
- 3721 • integer-mod

3722 The following functions SHALL take a single argument of the specified *data-type*. The round and

3723 floor functions SHALL take a single argument of data-type

3724 “http://www.w3.org/2001/XMLSchema#double” and return data-type

3725 "http://www.w3.org/2001/XMLSchema#double". In an expression that contains any of these
3726 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3727 "Indeterminate".

3728 • integer-abs

3729 • double-abs

3730 • round

3731 • floor

3732 **A14.3 String conversion functions**

3733 The following functions convert between values of the XACML
3734 "http://www.w3.org/2001/XMLSchema#string" primitive types. In an expression that contains any of
3735 these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3736 "Indeterminate".

3737 • string-normalize-space

3738 This function SHALL take one argument of data-type
3739 "http://www.w3.org/2001/XMLSchema#string" and SHALL normalize the value by stripping
3740 off all leading and trailing whitespace characters.

3741 • string-normalize-to-lower-case

3742 This function SHALL take one argument of "http://www.w3.org/2001/XMLSchema#string"
3743 and SHALL normalize the value by converting each upper case character to its lower case
3744 equivalent.

3745 **A14.4 Numeric data-type conversion functions**

3746 The following functions convert between the XACML
3747 "http://www.w3.org/2001/XMLSchema#integer" and "http://www.w3.org/2001/XMLSchema#double"
3748 primitive types. In any expression in which the functions defined below are applied, if any argument
3749 while being evaluated results in "Indeterminate", the expression SHALL return "Indeterminate".

3750 • double-to-integer

3751 This function SHALL take one argument of data-type
3752 "http://www.w3.org/2001/XMLSchema#double" and SHALL truncate its numeric value to a
3753 whole number and return an element of data-type
3754 "http://www.w3.org/2001/XMLSchema#integer".

3755 • integer-to-double

3756 This function SHALL take one argument of data-type
3757 "http://www.w3.org/2001/XMLSchema#integer" and SHALL promote its value to an element
3758 of data-type "http://www.w3.org/2001/XMLSchema#double" of the same numeric value.

3759 **A14.5 Logical functions**

3760 This section contains the specification for logical functions that operate on arguments of the
3761 "http://www.w3.org/2001/XMLSchema#boolean" data-type.

3762

- 3763 • or

3764 This function SHALL return "False" if it has no arguments and SHALL return "True" if one of
 3765 its arguments evaluates to "True". The order of evaluation SHALL be from first argument to
 3766 last. The evaluation SHALL stop with a result of "True" if any argument evaluates to "True",
 3767 leaving the rest of the arguments unevaluated. In an expression that contains any of these
 3768 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
 3769 "Indeterminate".
- 3770 • and

3771 This function SHALL return "True" if it has no arguments and SHALL return "False" if one of
 3772 its arguments evaluates to "False". The order of evaluation SHALL be from first argument
 3773 to last. The evaluation SHALL stop with a result of "False" if any argument evaluates to
 3774 "False", leaving the rest of the arguments unevaluated. In an expression that contains any
 3775 of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate
 3776 to "Indeterminate".
- 3777 • n-of

3778 The first argument to this function SHALL be of data-type
 3779 "http://www.w3.org/2001/XMLSchema#integer", specifying the number of the remaining
 3780 arguments that MUST evaluate to "True" for the expression to be considered "True". If the
 3781 first argument is 0, the result SHALL be "True". If the number of arguments after the first
 3782 one is less than the value of the first argument, then the expression SHALL result in
 3783 "Indeterminate". The order of evaluation SHALL be: first evaluate the integer value, then
 3784 evaluate each subsequent argument. The evaluation SHALL stop and return "True" if the
 3785 specified number of arguments evaluate to "True". The evaluation of arguments SHALL
 3786 stop if it is determined that evaluating the remaining arguments will not satisfy the
 3787 requirement. In an expression that contains any of these functions, if any argument is
 3788 "Indeterminate", then the expression SHALL evaluate to "Indeterminate".
- 3789 • not

3790 This function SHALL take one logical argument. If the argument evaluates to "True", then
 3791 the result of the expression SHALL be "False". If the argument evaluates to "False", then
 3792 the result of the expression SHALL be "True". In an expression that contains any of these
 3793 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
 3794 "Indeterminate".

3795 **A14.6 Arithmetic comparison functions**

3796 These functions form a minimal set for comparing two numbers, yielding a boolean result. They
 3797 SHALL comply with the rules governed by IEEE 754 [IEEE 754]. In an expression that contains
 3798 any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
 3799 "Indeterminate".

- 3800 • integer-greater-than
- 3801 • integer-greater-than-or-equal
- 3802 • integer-less-than
- 3803 • integer-less-than-or-equal
- 3804 • double-greater-than
- 3805 • double-greater-than-or-equal

- 3806 • double-less-than
- 3807 • double-less-than-or-equal

3808 **A14.7 Date and time arithmetic functions**

3809 These functions perform arithmetic operations with the date and time. In an expression that
 3810 contains any of these functions, if any argument is "Indeterminate", then the expression SHALL
 3811 evaluate to "Indeterminate".

- 3812 • dateTime-add-dayTimeDuration

3813 This function SHALL take two arguments, the first is of data-type
 3814 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is of data-type
 3815 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration". It SHALL
 3816 return a result of "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL
 3817 return the value by adding the second argument to the first argument according to the
 3818 specification of adding durations to date and time [XS Appendix E].

- 3819 • dateTime-add-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/2002/WD-xquery-operators-20020816#yearMonthDuration". It
 3823 SHALL return a result of "http://www.w3.org/2001/XMLSchema#dateTime". This function
 3824 SHALL return the value by adding the second argument to the first argument according to
 3825 the specification of adding durations to date and time [XS Appendix E].

- 3826 • dateTime-subtract-dayTimeDuration

3827 This function SHALL take two arguments, the first is a
 3828 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a
 3829 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration". It SHALL
 3830 return a result of "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument
 3831 is a positive duration, then this function SHALL return the value by adding the
 3832 corresponding negative duration, as per the specification [XS Appendix E]. If the second
 3833 argument is a negative duration, then the result SHALL be as if the function
 3834 "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration" had been applied
 3835 to the corresponding positive duration.

- 3836 • dateTime-subtract-yearMonthDuration

3837 This function SHALL take two arguments, the first is a
 3838 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a
 3839 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It
 3840 SHALL return a result of "http://www.w3.org/2001/XMLSchema#dateTime". If the second
 3841 argument is a positive duration, then this function SHALL return the value by adding the
 3842 corresponding negative duration, as per the specification [XS Appendix E]. If the second
 3843 argument is a negative duration, then the result SHALL be as if the function
 3844 "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration" had been
 3845 applied to the corresponding positive duration.

- 3846 • date-add-yearMonthDuration

3847 This function SHALL take two arguments, the first is a
 3848 "http://www.w3.org/2001/XMLSchema#date" and the second is a
 3849 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It
 3850 return a result of "http://www.w3.org/2001/XMLSchema#date". This function SHALL return

3851 the value by adding the second argument to the first argument according to the
3852 specification of adding durations to date [XS Appendix E].

3853 • date-subtract-yearMonthDuration

3854 This function SHALL take two arguments, the first is a
3855 “http://www.w3.org/2001/XMLSchema#date” and the second is a
3856 “http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration”. It
3857 SHALL return a result of “http://www.w3.org/2001/XMLSchema#date”. If the second
3858 argument is a positive duration, then this function SHALL return the value by adding the
3859 corresponding negative duration, as per the specification [XS Appendix E]. If the second
3860 argument is a negative duration, then the result SHALL be as if the function
3861 “urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration” had been applied to
3862 the corresponding positive duration.

3863 **A14.8 Non-numeric comparison functions**

3864 These functions perform comparison operations on two arguments of non-numerical types. In an
3865 expression that contains any of these functions, if any argument is "Indeterminate", then the
3866 expression SHALL evaluate to "Indeterminate".

3867 • string-greater-than

3868 This function SHALL take two arguments of data-type
3869 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
3870 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if and only if the
3871 arguments are compared byte by byte and, after an initial prefix of corresponding bytes
3872 from both arguments that are considered equal by
3873 “urn:oasis:names:tc:xacml:1.0:function:integer-equal”, the next byte by byte comparison is
3874 such that the byte from the first argument is greater than the byte from the second
3875 argument by the use of the function “urn:oasis:names:tc:xacml:1.0:function:integer-equal”.

3876 • string-greater-than-or-equal

3877 This function SHALL take two arguments of data-type
3878 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
3879 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated
3880 with the logical function “urn:oasis:names:tc:xacml:1.0:function:or” with two arguments
3881 containing the functions “urn:oasis:names:tc:xacml:1.0:function:string-greater-than” and
3882 “urn:oasis:names:tc:xacml:1.0:function:string-equal” containing the original arguments

3883 • string-less-than

3884 This function SHALL take two arguments of data-type
3885 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
3886 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if and only if the
3887 arguments are compared byte by byte and, after an initial prefix of corresponding bytes
3888 from both arguments are considered equal by
3889 “urn:oasis:names:tc:xacml:1.0:function:integer-equal”, the next byte by byte comparison is
3890 such that the byte from the first argument is less than the byte from the second argument
3891 by the use of the function “urn:oasis:names:tc:xacml:1.0:function:integer-less-than”.

3892 • string-less-than-or-equal

3893 This function SHALL take two arguments of data-type
3894 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
3895 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated
3896 with the function “urn:oasis:names:tc:xacml:1.0:function:or” with two arguments containing

- 3897 the functions “urn:oasis:names:tc:xacml:1.0:function:string-less-than” and
3898 “urn:oasis:names:tc:xacml:1.0:function:string-equal” containing the original arguments.
- 3899 • time-greater-than
- 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 greater than the second argument according to the order relation specified for
3904 “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].
- 3905 • time-greater-than-or-equal
- 3906 This function SHALL take two arguments of data-type
3907 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an
3908 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
3909 argument is greater than or equal to the second argument according to the order relation
3910 specified for “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].
- 3911 • time-less-than
- 3912 This function SHALL take two arguments of data-type
3913 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an
3914 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
3915 argument is less than the second argument according to the order relation specified for
3916 “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].
- 3917 • time-less-than-or-equal
- 3918 This function SHALL take two arguments of data-type
3919 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an
3920 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
3921 argument is less than or equal to the second argument according to the order relation
3922 specified for “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].
- 3923 • dateTime-greater-than
- 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 greater than the second argument according to the order relation specified for
3928 “http://www.w3.org/2001/XMLSchema#dateTime” [XS Section 3.2.7].
- 3929 • dateTime-greater-than-or-equal
- 3930 This function SHALL take two arguments of data-type
3931 “http://www.w3.org/2001/XMLSchema#dateTime” and SHALL return an
3932 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
3933 argument is greater than or equal to the second argument according to the order relation
3934 specified for “http://www.w3.org/2001/XMLSchema#dateTime” [XS Section 3.2.7].
- 3935 • dateTime-less-than
- 3936 This function SHALL take two arguments of data-type
3937 “http://www.w3.org/2001/XMLSchema#dateTime” and SHALL return an
3938 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
3939 argument is less than the second argument according to the order relation specified for
3940 “http://www.w3.org/2001/XMLSchema#dateTime” [XS Section 3.2.7].
- 3941

- 3942 • `dateTime-less-than-or-equal`
- 3943 This function SHALL take two arguments of data-type
3944 “`http://www.w3.org/2001/XMLSchema#dateTime`” and SHALL return an
3945 “`http://www.w3.org/2001/XMLSchema#boolean`”. It SHALL return "True" if the first
3946 argument is less than or equal to the second argument according to the order relation
3947 specified for “`http://www.w3.org/2001/XMLSchema#dateTime`” [XS Section 3.2.7].
- 3948 • `date-greater-than`
- 3949 This function SHALL take two arguments of data-type
3950 “`http://www.w3.org/2001/XMLSchema#date`” and SHALL return an
3951 “`http://www.w3.org/2001/XMLSchema#boolean`”. It SHALL return "True" if the first
3952 argument is greater than the second argument according to the order relation specified for
3953 “`http://www.w3.org/2001/XMLSchema#date`” [XS Section 3.2.9].
- 3954 • `date-greater-than-or-equal`
- 3955 This function SHALL take two arguments of data-type
3956 “`http://www.w3.org/2001/XMLSchema#date`” and SHALL return an
3957 “`http://www.w3.org/2001/XMLSchema#boolean`”. It SHALL return "True" if the first
3958 argument is greater than or equal to the second argument according to the order relation
3959 specified for “`http://www.w3.org/2001/XMLSchema#date`” [XS Section 3.2.9].
- 3960 • `date-less-than`
- 3961 This function SHALL take two arguments of data-type
3962 “`http://www.w3.org/2001/XMLSchema#date`” and SHALL return an
3963 “`http://www.w3.org/2001/XMLSchema#boolean`”. It SHALL return "True" if the first
3964 argument is less than the second argument according to the order relation specified for
3965 “`http://www.w3.org/2001/XMLSchema#date`” [XS Section 3.2.9].
- 3966 • `date-less-than-or-equal`
- 3967 This function SHALL take two arguments of data-type
3968 “`http://www.w3.org/2001/XMLSchema#date`” and SHALL return an
3969 “`http://www.w3.org/2001/XMLSchema#boolean`”. It SHALL return "True" if the first
3970 argument is less than or equal to the second argument according to the order relation
3971 specified for “`http://www.w3.org/2001/XMLSchema#date`” [XS Section 3.2.9].

3972 **A14.9 Bag functions**

3973 These functions operate on a **bag** of *type* values, where *data-type* is one of the primitive types. In
3974 an expression that contains any of these functions, if any argument is "Indeterminate", then the
3975 expression SHALL evaluate to "Indeterminate". Some additional conditions defined for each
3976 function below SHALL cause the expression to evaluate to "Indeterminate".

- 3977 • `type-one-and-only`
- 3978 This function SHALL take an argument of a **bag** of *type* values and SHALL return a value
3979 of *data-type*. It SHALL return the only value in the **bag**. If the **bag** does not have one and
3980 only one value, then the expression SHALL evaluate to "Indeterminate".
- 3981 • `type-bag-size`
- 3982 This function SHALL take a **bag** of *type* values as an argument and SHALL return an
3983 “`http://www.w3.org/2001/XMLSchema#integer`” indicating the number of values in the **bag**.
- 3984

3985

- 3986 • *type-is-in*

3987 This function SHALL take an argument of data-type *type* as the first argument and a **bag** of
3988 *type* values as the second argument. The expression SHALL evaluate to "True" if the first
3989 argument matches by the "urn:oasis:names:tc:xacml:1.0:function:type-equal" to any value
3990 in the **bag**.

- 3991 • *type-bag*

3992 This function SHALL take any number of arguments of a single data-type and return a **bag**
3993 of *type* values containing the values of the arguments. An application of this function to
3994 zero arguments SHALL produce an empty **bag** of the specified data-type.

3995 **A14.10 Set functions**

3996 These functions operate on **bags** mimicking sets by eliminating duplicate elements from a **bag**. In
3997 an expression that contains any of these functions, if any argument is "Indeterminate", then the
3998 expression SHALL evaluate to "Indeterminate".

- 3999 • *type-intersection*

4000 This function SHALL take two arguments that are both a **bag** of *type* values. The
4001 expression SHALL return a **bag** of *type* values such that it contains only elements that are
4002 common between the two **bags**, which is determined by
4003 "urn:oasis:names:tc:xacml:1.0:function:type-equal". No duplicates as determined by
4004 "urn:oasis:names:tc:xacml:1.0:function:type-equal" SHALL exist in the result.

- 4005 • *type-at-least-one-member-of*

4006 This function SHALL take two arguments that are both a **bag** of *type* values. The
4007 expression SHALL evaluate to "True" if at least one element of the first argument is
4008 contained in the second argument as determined by
4009 "urn:oasis:names:tc:xacml:1.0:function:type-is-in".

- 4010 • *type-union*

4011 This function SHALL take two arguments that are both a **bag** of *type* values. The
4012 expression SHALL return a **bag** of *type* such that it contains all elements of both **bags**. No
4013 duplicates as determined by "urn:oasis:names:tc:xacml:1.0:function:type-equal" SHALL
4014 exist in the result.

- 4015 • *type-subset*

4016 This function SHALL take two arguments that are both a **bag** of *type* values. It SHALL
4017 return "True" if the first argument is a subset of the second argument. Each argument is
4018 considered to have its duplicates removed as determined by
4019 "urn:oasis:names:tc:xacml:1.0:function:type-equal" before subset calculation.

- 4020 • *type-set-equals*

4021 This function SHALL take two arguments that are both a **bag** of *type* values and SHALL
4022 return the result of applying "urn:oasis:names:tc:xacml:1.0:function:and" to the application
4023 of "urn:oasis:names:tc:xacml:1.0:function:type-subset" to the first and second arguments
4024 and the application of "urn:oasis:names:tc:xacml:1.0:function:type-subset" to the second
4025 and first arguments.

4026 **A14.11 Higher-order bag functions**

4027 This section describes functions in XACML that perform operations on **bags** such that functions
4028 may be applied to the **bags** in general.

4029 In this section, a general-purpose functional language called Haskell [**Haskell**] is used to formally
4030 specify the semantics of these functions. Although the English description is adequate, a formal
4031 specification of the semantics is helpful.

4032 For a quick summary, in the following Haskell notation, a function definition takes the form of
4033 clauses that are applied to patterns of structures, namely lists. The symbol “[]” denotes the empty
4034 list, whereas the expression “(x:xs)” matches against an argument of a non-empty list of which “x”
4035 represents the first element of the list, and “xs” is the rest of the list, which may be an empty list. We
4036 use the Haskell notion of a list, which is an ordered collection of elements, to model the XACML
4037 **bags** of values.

4038 A simple Haskell definition of a familiar function “urn:oasis:names:tc:xacml:1.0:function:and” that
4039 takes a list of booleans is defined as follows:

4040 and:: [Bool] -> Bool

4041 and [] = "True"

4042 and (x:xs) = x && (and xs)

4043 The first definition line denoted by a “::” formally describes the data-type of the function, which takes
4044 a list of booleans, denoted by “[Bool]”, and returns a boolean, denoted by “Bool”. The second
4045 definition line is a clause that states that the function “and” applied to the empty list is “True”. The
4046 second definition line is a clause that states that for a non-empty list, such that the first element is
4047 “x”, which is a value of data-type Bool, the function “and” applied to x SHALL be combined with,
4048 using the logical conjunction function, which is denoted by the infix symbol “&&”, the result of
4049 recursively applying the function “and” to the rest of the list. Of course, an application of the “and”
4050 function is “True” if and only if the list to which it is applied is empty or every element of the list is
4051 “True”. For example, the evaluation of the following Haskell expressions,

4052 (and []), (and ["True"]), (and ["True","True"]), (and ["True","True","False"])

4053 evaluate to "True", "True", "True", and "False", respectively.

4054 In an expression that contains any of these functions, if any argument is "Indeterminate", then the
4055 expression SHALL evaluate to "Indeterminate".

- 4056 • any-of

4057 This function applies a boolean function between a specific primitive value and a **bag** of
4058 values, and SHALL return "True" if and only if the predicate is "True" for at least one
4059 element of the **bag**.

4060 This function SHALL take three arguments. The first argument SHALL be a <Function>
4061 element that names a boolean function that takes two arguments of primitive types. The
4062 second argument SHALL be a value of a primitive data-type. The third argument SHALL
4063 be a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function
4064 named in the <Function> element is applied to the second argument and each element
4065 of the third argument (the **bag**) and the results are combined with
4066 “urn:oasis:names:tc:xacml:1.0:function:or”.

4067 In Haskell, the semantics of this operation are as follows:

```
4068 any_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
4069 any_of f a [] = "False"
4070 any_of f a (x:xs) = (f a x) || (any_of f a xs)
```

4071 In the above notation, “f” is the function name to be applied, “a” is the primitive value, and
4072 “(x:xs)” represents the first element of the list as “x” and the rest of the list as “xs”.

4073 For example, the following expression SHALL return "True":

```
4074 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
4075 <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
4076 <AttributeValue
4077 DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4078 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4079 <AttributeValue
4080 DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4081 <AttributeValue
4082 DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4083 <AttributeValue
4084 DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4085 <AttributeValue
4086 DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4087 </Apply>
4088 </Apply>
```

4089 This expression is "True" because the first argument is equal to at least one of the
4090 elements of the **bag**.

4091 • all-of

4092 This function applies a boolean function between a specific primitive value and a **bag** of
4093 values, and returns "True" if and only if the predicate is "True" for every element of the **bag**.

4094 This function SHALL take three arguments. The first argument SHALL be a <Function>
4095 element that names a boolean function that takes two arguments of primitive types. The
4096 second argument SHALL be a value of a primitive data-type. The third argument SHALL
4097 be a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function
4098 named in the <Function> element were applied to the second argument and each
4099 element of the third argument (the **bag**) and the results were combined using
4100 "urn:oasis:names:tc:xacml:1.0:function:and".

4101 In Haskell, the semantics of this operation are as follows:

```
4102 all_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
4103 all_of f a [] = "False"
4104 all_of f a (x:xs) = (f a x) && (all_of f a xs)
```

4105 In the above notation, “f” is the function name to be applied, “a” is the primitive value, and
4106 “(x:xs)” represents the first element of the list as “x” and the rest of the list as “xs”.

4107 For example, the following expression SHALL evaluate to "True":

```

4108 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">
4109   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4110   greater"/>
4111   <AttributeValue
4112   DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4113   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4114     <AttributeValue
4115     DataType="http://www.w3.org/2001/XMLSchema#integer">9</AttributeValue>
4116     <AttributeValue
4117     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4118     <AttributeValue
4119     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4120     <AttributeValue
4121     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4122   </Apply>
4123 </Apply>

```

4124 This expression is "True" because the first argument is greater than *all* of the elements of
4125 the **bag**.

4126 • any-of-any

4127 This function applies a boolean function between each element of a **bag** of values and
4128 each element of another **bag** of values, and returns "True" if and only if the predicate is
4129 "True" for at least one comparison.

4130 This function SHALL take three arguments. The first argument SHALL be a <Function>
4131 element that names a boolean function that takes two arguments of primitive types. The
4132 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be
4133 a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function
4134 named in the <Function> element were applied between *every* element in the second
4135 argument and *every* element of the third argument (the **bag**) and the results were
4136 combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics are that the
4137 result of the expression SHALL be "True" if and only if the applied predicate is "True" for
4138 *any* comparison of elements from the two **bags**.

4139 In Haskell, taking advantage of the "any_of" function defined above, the semantics of the
4140 "any_of_any" function are as follows:

```

4141 any_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4142 any_of_any f [] ys = "False"
4143 any_of_any f (x:xs) ys = (any_of f x ys) || (any_of_any f xs ys)

```

4144 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4145 element of the list as "x" and the rest of the list as "xs".

4146 For example, the following expression SHALL evaluate to "True":

```

4147 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">
4148   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
4149   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4150     <AttributeValue
4151       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4152     <AttributeValue
4153       DataType="http://www.w3.org/2001/XMLSchema#string">Mary</AttributeValue>
4154   </Apply>
4155   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4156     <AttributeValue
4157       DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4158     <AttributeValue
4159       DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4160     <AttributeValue
4161       DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4162     <AttributeValue
4163       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4164   </Apply>
4165 </Apply>

```

4166 This expression is "True" because at least one of the elements of the first **bag**, namely
4167 "Ringo", is equal to at least one of the string values of the second **bag**.

4168 • all-of-any

4169 This function applies a boolean function between the elements of two **bags**. The
4170 expression is "True" if and only if the predicate is "True" between each and all of the
4171 elements of the first **bag** collectively against at least one element of the second **bag**.

4172 This function SHALL take three arguments. The first argument SHALL be a <Function>
4173 element that names a boolean function that takes two arguments of primitive types. The
4174 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be
4175 a **bag** of a primitive data-type. The expression SHALL be evaluated as if function named in
4176 the <Function> element were applied between every element in the second argument
4177 and every element of the third argument (the **bag**) using
4178 "urn:oasis:names:tc:xacml:1.0:function:and". The semantics are that the result of the
4179 expression SHALL be "True" if and only if the applied predicate is "True" for each element
4180 of the first **bag** and any element of the second **bag**.

4181 In Haskell, taking advantage of the "any_of" function defined in Haskell above, the
4182 semantics of the "all_of_any" function are as follows:

```

4183               all_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4184               all_of_any f []       ys = "False"
4185               all_of_any f (x:xs) ys = (any_of f x ys) && (all_of_any f xs ys)

```

4186 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4187 element of the list as "x" and the rest of the list as "xs".

4188 For example, the following expression SHALL evaluate to "True":


```

4189 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-any">
4190   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4191   greater"/>
4192   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4193     <AttributeValue
4194     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4195     <AttributeValue
4196     DataType="http://www.w3.org/2001/XMLSchema#integer">20</AttributeValue>
4197   </Apply>
4198   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4199     <AttributeValue
4200     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4201     <AttributeValue
4202     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4203     <AttributeValue
4204     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4205     <AttributeValue
4206     DataType="http://www.w3.org/2001/XMLSchema#integer">21</AttributeValue>
4207   </Apply>
4208 </Apply>

```

4209 This expression is "True" because all of the elements of the first **bag**, each "10" and "20",
4210 are greater than at least one of the integer values "1", "3", "5", "21" of the second **bag**.

4211 • any-of-all

4212 This function applies a boolean function between the elements of two **bags**. The
4213 expression SHALL be "True" if and only if the predicate is "True" between at least one of
4214 the elements of the first **bag** collectively against all the elements of the second **bag**.

4215 This function SHALL take three arguments. The first argument SHALL be a <Function>
4216 element that names a boolean function that takes two arguments of primitive types. The
4217 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be
4218 a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function
4219 named in the <Function> element were applied between *every* element in the second
4220 argument and *every* element of the third argument (the **bag**) and the results were
4221 combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics are that the
4222 result of the expression SHALL be "True" if and only if the applied predicate is "True" for
4223 *any* element of the first **bag** compared to *all* the elements of the second **bag**.

4224 In Haskell, taking advantage of the "all_of" function defined in Haskell above, the semantics
4225 of the "any_of_all" function are as follows:

```

4226 any_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4227 any_of_all f [] ys = "False"
4228 any_of_all f (x:xs) ys = (all_of f x ys) || ( any_of_all f xs ys)

```

4229 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4230 element of the list as "x" and the rest of the list as "xs".

4231 For example, the following expression SHALL evaluate to "True":

```

4232 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-all">
4233   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4234   greater"/>
4235   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4236     <AttributeValue
4237     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4238     <AttributeValue
4239     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4240   </Apply>
4241   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4242     <AttributeValue
4243     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4244     <AttributeValue
4245     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4246     <AttributeValue
4247     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4248     <AttributeValue
4249     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4250   </Apply>
4251 </Apply>

```

4252 This expression is "True" because at least one element of the first **bag**, namely "5", is
4253 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4254 • all-of-all

4255 This function applies a boolean function between the elements of two **bags**. The
4256 expression SHALL be "True" if and only if the predicate is "True" between each and all of
4257 the elements of the first **bag** collectively against all the elements of the second **bag**.

4258 This function SHALL take three arguments. The first argument SHALL be a <Function>
4259 element that names a boolean function that takes two arguments of primitive types. The
4260 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be
4261 a **bag** of a primitive data-type. The expression is evaluated as if the function named in the
4262 <Function> element were applied between *every* element in the second argument and
4263 *every* element of the third argument (the **bag**) and the results were combined using
4264 "urn:oasis:names:tc:xacml:1.0:function:and". The semantics are that the result of the
4265 expression is "True" if and only if the applied predicate is "True" for *all* elements of the first
4266 **bag** compared to *all* the elements of the second **bag**.

4267 In Haskell, taking advantage of the "all_of" function defined in Haskell above, the semantics
4268 of the "all_of_all" function is as follows:

```

4269 all_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4270 all_of_all f [] ys = "False"
4271 all_of_all f (x:xs) ys = (all_of f x ys) && (all_of_all f xs ys)

```

4272 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4273 element of the list as "x" and the rest of the list as "xs".

4274 For example, the following expression SHALL evaluate to "True":

```

4275 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-all">
4276   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4277   greater"/>
4278   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4279     <AttributeValue
4280     DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>
4281     <AttributeValue
4282     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4283   </Apply>
4284   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4285     <AttributeValue
4286     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4287     <AttributeValue
4288     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4289     <AttributeValue
4290     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4291     <AttributeValue
4292     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4293   </Apply>
4294 </Apply>

```

4295 This expression is "True" because all elements of the first **bag**, "5" and "6", are each
4296 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4297 • map

4298 This function converts a **bag** of values to another **bag** of values.

4299 This function SHALL take two arguments. The first function SHALL be a <Function>
4300 element naming a function that takes a single argument of a primitive data-type and returns
4301 a value of a primitive data-type. The second argument SHALL be a **bag** of a primitive data-
4302 type. The expression SHALL be evaluated as if the function named in the <Function>
4303 element were applied to each element in the **bag** resulting in a **bag** of the converted value.
4304 The result SHALL be a **bag** of the primitive data-type that is the same data-type that is
4305 returned by the function named in the <Function> element.

4306 In Haskell, this function is defined as follows:

```

4307     map:: (a -> b) -> [a] -> [b]
4308     map f []     = []
4309     map f (x:xs) = (f x) : (map f xs)

```

4310 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4311 element of the list as "x" and the rest of the list as "xs".

4312 For example, the following expression,

```

4313 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:map">
4314   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
4315   normalize-to-lower-case">
4316   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4317     <AttributeValue
4318     DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>
4319     <AttributeValue
4320     DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>
4321   </Apply>
4322 </Apply>

```

4323 evaluates to a **bag** containing "hello" and "world!".

4324

A14.12 Special match functions

4325 These functions operate on various types and evaluate to
4326 "http://www.w3.org/2001/XMLSchema#boolean" based on the specified standard matching
4327 algorithm. In an expression that contains any of these functions, if any argument is "Indeterminate",
4328 then the expression SHALL evaluate to "Indeterminate".

4329 • regex-string-match

4330 This function decides a regular expression match. It SHALL take two arguments of
4331 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4332 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4333 expression and the second argument SHALL be a general string. The function
4334 specification SHALL be that of the "xf:matches" function with the arguments reversed [XF
4335 Section 6.3.15].

4336 • x500Name-match

4337 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-
4338 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It
4339 shall return "True" if and only if the first argument matches some terminal sequence of
4340 RDNs from the second argument when compared using x500Name-equal.

4341 • rfc822Name-match

4342 This function SHALL take two arguments, the first is of data-type
4343 "http://www.w3.org/2001/XMLSchema#string" and the second is of data-type
4344 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an
4345 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if
4346 the first argument matches the second argument according to the following specification.

4347 An RFC822 name consists of a local-part followed by "@" followed by domain-part. The
4348 local-part is case-sensitive, while the domain-part (which is usually a DNS name) is not
4349 case-sensitive.⁴

4350 The second argument contains a complete rfc822Name. The first argument is a complete
4351 or partial rfc822Name used to select appropriate values in the second argument as follows.

4352 In order to match a particular mailbox in the second argument, the first argument must
4353 specify the complete mail address to be matched. For example, if the first argument is
4354 "Anderson@sun.com", this matches a value in the second argument of
4355 "Anderson@sun.com" and "Anderson@SUN.COM", but not "Anne.Anderson@sun.com",
4356 "anderson@sun.com" or "Anderson@east.sun.com".

4357 In order to match any mail address at a particular domain in the second argument, the first
4358 argument must specify only a domain name (usually a DNS name). For example, if the first
4359 argument is "sun.com", this matches a value in the first argument of "Anderson@sun.com"
4360 or "Baxter@SUN.COM", but not "Anderson@east.sun.com".

4361 In order to match any mail address in a particular domain in the second argument, the first
4362 argument must specify the desired domain-part with a leading ".". For example, if the first
4363 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.

4364 "Anderson@east.sun.com" and "anne.anderson@ISRG.EAST.SUN.COM" but not
4365 "Anderson@sun.com".

4366 **A14.13 XPath-based functions**

4367 This section specifies functions that take XPath expressions for arguments. An XPath expression
4368 evaluates to a *node-set*, which is a set of XML nodes that match the expression. A node or node-
4369 set is not in the formal data-type system of XACML. All comparison or other operations on node-
4370 sets are performed in the isolation of the particular function specified. The XPath expressions in
4371 these functions are restricted to the XACML request **context**. The following functions are defined:

- 4372 • xpath-node-count

4373 This function SHALL take an "http://www.w3.org/2001/XMLSchema#string" as an
4374 argument, which SHALL be interpreted as an XPath expression, and evaluates to an
4375 "http://www.w3.org/2001/XMLSchema#integer". The value returned from the function
4376 SHALL be the count of the nodes within the node-set that matches the given XPath
4377 expression.

- 4378 • xpath-node-equal

4379 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,
4380 which SHALL be interpreted as XPath expressions, and SHALL return an
4381 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if any
4382 XML node from the node-set matched by the first argument equals according to the
4383 "op:node-equal" function [XF Section 13.1.6] any XML node from the node-set matched by
4384 the second argument.

- 4385 • xpath-node-match

4386 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,
4387 which SHALL be interpreted as XPath expressions and SHALL return an
4388 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL first extend the first
4389 argument to match an XML document in a hierarchical fashion. If *a* is an XPath expression
4390 and it is specified as the first argument, it SHALL be interpreted to mean match the set of
4391 nodes specified by the enhanced XPath expression "*a* | *a*/* | *a*/*@*". In other words, the
4392 expression *a* SHALL match all elements and attributes below the element specified by *a*.
4393 This function SHALL evaluate to "True" if any XML node that matches the enhanced XPath
4394 expression is equal according to "op:node-equal" [XF Section 13.1.6] to any XML node
4395 from the node-set matched by the second argument.

4396 **A14.14 Extension functions and primitive types**

4397 Functions and primitive types are specified by string identifiers allowing for the introduction of
4398 functions in addition to those specified by XACML. This approach allows one to extend the XACML
4399 module with special functions and special primitive data-types.

4400 In order to preserve some integrity to the XACML evaluation strategy, the result of all function
4401 applications SHALL depend only on the values of its arguments. Global and hidden parameters
4402 SHALL NOT affect the evaluation of an expression. Functions SHALL NOT have side effects, as
4403 evaluation order cannot be guaranteed in a standard way.

4404 Appendix B. XACML identifiers (normative)

4405 This section defines standard identifiers for commonly used entities. All XACML-defined identifiers
4406 have the common base:

4407 `urn:oasis:names:tc:xacml:1.0`

4408 B.1. XACML namespaces

4409 There are currently two defined XACML namespaces.

4410 Policies are defined using this identifier.

4411 `urn:oasis:names:tc:xacml:1.0:policy`

4412 Request and response **contexts** are defined using this identifier.

4413 `urn:oasis:names:tc:xacml:1.0:context`

4414 B.2. Access subject categories

4415 This identifier indicates the system entity that initiated the **access** request. That is, the initial entity
4416 in a request chain. If **subject** category is not specified, this is the default value.

4417 `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

4418 This identifier indicates the system entity that will receive the results of the request. Used when it is
4419 distinct from the access-subject.

4420 `urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject`

4421 This identifier indicates a system entity through which the **access** request was passed. There may
4422 be more than one. No means is provided to specify the order in which they passed the message.

4423 `urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject`

4424 This identifier indicates a system entity associated with a local or remote codebase that generated
4425 the request. Corresponding **subject attributes** might include the URL from which it was loaded
4426 and/or the identity of the code-signer. There may be more than one. No means is provided to
4427 specify the order they processed the request.

4428 `urn:oasis:names:tc:xacml:1.0:subject-category:codebase`

4429 This identifier indicates a system entity associated with the computer that initiated the **access**
4430 request. An example would be an IPsec identity.

4431 `urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine`

4432 B.3. XACML functions

4433 This identifier is the base for all the identifiers in the table of functions. See Section A.1.

4434 `urn:oasis:names:tc:xacml:1.0:function`

4435 B.4. Data-types

4436 The following identifiers indicate useful data-types.

4437 X.500 distinguished name

4438 urn:oasis:names:tc:xacml:1.0:data-type:x500Name

4439 An x500Name contains an ITU-T Rec. X.520 Distinguished Name. The valid syntax for such a
 4440 name is described in IETF RFC 2253 "Lightweight Directory Access Protocol (v3): UTF-8 String
 4441 Representation of Distinguished Names".

4442 RFC822 Name

4443 urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name

4444 An rfc822Name contains an "e-mail name". The valid syntax for such a name is described in IETF
 4445 RFC 2821, Section 4.1.2, Command Argument Syntax, under the term "Mailbox".

4446 The following data-type identifiers are defined by XML Schema.

4447 http://www.w3.org/2001/XMLSchema#string
 4448 http://www.w3.org/2001/XMLSchema#boolean
 4449 http://www.w3.org/2001/XMLSchema#integer
 4450 http://www.w3.org/2001/XMLSchema#double
 4451 http://www.w3.org/2001/XMLSchema#time
 4452 http://www.w3.org/2001/XMLSchema#date
 4453 http://www.w3.org/2001/XMLSchema#dateTime
 4454 http://www.w3.org/2001/XMLSchema#anyURI
 4455 http://www.w3.org/2001/XMLSchema#hexBinary
 4456 http://www.w3.org/2001/XMLSchema#base64Binary

4457 The following data-type identifiers correspond to the dayTimeDuration and yearMonthDuration
 4458 data-types defined in [XF Sections 8.2.2 and 8.2.1, respectively].

4459 http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration
 4460 http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration

4461 B.5. Subject attributes

4462 These identifiers indicate **attributes** of a **subject**. When used, they SHALL appear within a
 4463 <Subject> element of the request **context**. They SHALL be accessed via a
 4464 <SubjectAttributeDesignator> or an <AttributeSelector> element pointing into a
 4465 <Subject> element of the request **context**.

4466 At most one of each of these attributes is associated with each subject. Each attribute associated
 4467 with authentication included within a single <Subject> element relates to the same authentication
 4468 event.

4469 This identifier indicates the name of the **subject**. The default format is
 4470 http://www.w3.org/2001/XMLSchema#string. To indicate other formats, use `Data Type` attributes
 4471 listed in B.4

4472 urn:oasis:names:tc:xacml:1.0:subject:subject-id

4473 This identifier indicates the **subject** category. "access-subject" is the default.

4474 urn:oasis:names:tc:xacml:1.0:subject-category

4475 This identifier indicates the security domain of the **subject**. It identifies the administrator and policy
 4476 that manages the name-space in which the **subject** id is administered.

4477 urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier

4478 This identifier indicates a public key used to confirm the **subject's** identity.

4479 urn:oasis:names:tc:xacml:1.0:subject:key-info

4480 This identifier indicates the time at which the **subject** was authenticated.

4481 urn:oasis:names:tc:xacml:1.0:subject:authentication-time

4482 This identifier indicates the method used to authenticate the **subject**.

4483 urn:oasis:names:tc:xacml:1.0:subject:authentication-method

4484 This identifier indicates the time at which the **subject** initiated the **access** request, according to the
4485 **PEP**.

4486 urn:oasis:names:tc:xacml:1.0:subject:request-time

4487 This identifier indicates the time at which the **subject's** current session began, according to the
4488 **PEP**.

4489 urn:oasis:names:tc:xacml:1.0:subject:session-start-time

4490 The following identifiers indicate the location where authentication credentials were activated. They
4491 are intended to support the corresponding entities from the SAML authentication statement.

4492 This identifier indicates that the location is expressed as an IP address.

4493 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address

4494 This identifier indicates that the location is expressed as a DNS name.

4495 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name

4496 Where a suitable attribute is already defined in LDAP [**LDAP-1**, **LDAP-2**], the XACML identifier
4497 SHALL be formed by adding the **attribute** name to the URI of the LDAP specification. For
4498 example, the **attribute** name for the userPassword defined in the rfc2256 SHALL be:

4499 http://www.ietf.org/rfc/rfc2256.txt#userPassword

4500 B.6. Resource attributes

4501 These identifiers indicate **attributes** of the **resource**. When used, they SHALL appear within the
4502 <Resource> element of the request **context**. They SHALL be accessed via a
4503 <ResourceAttributeDesignator> or an <AttributeSelector> element pointing into the
4504 <Resource> element of the request **context**.

4505 This identifier indicates the entire URI of the **resource**.

4506 urn:oasis:names:tc:xacml:1.0:resource:resource-id

4507 A **resource attribute** used to indicate values extracted from the **resource**.

4508 urn:oasis:names:tc:xacml:1.0:resource:resource-content

4509 This identifier indicates the last (rightmost) component of the file name. For example, if the URI is:
4510 "file://home/my/status#pointer", the simple-file-name is "status".

4511 urn:oasis:names:tc:xacml:1.0:resource:simple-file-name

4512 This identifier indicates that the **resource** is specified by an XPath expression.

4513 urn:oasis:names:tc:xacml:1.0:resource:xpath

4514 This identifier indicates a UNIX file-system path.

4515 urn:oasis:names:tc:xacml:1.0:resource:ufs-path

4516 This identifier indicates the scope of the **resource**, as described in Section 7.8.

4517 urn:oasis:names:tc:xacml:1.0:resource:scope

4518 The allowed value for this attribute is of data-type http://www.w3.org/2001/XMLSchema#string, and
4519 is either "Immediate", "Children" or "Descendants".

4520 B.7. Action attributes

4521 These identifiers indicate **attributes** of the **action** being requested. When used, they SHALL appear
4522 within the <Action> element of the request **context**. They SHALL be accessed via an
4523 <ActionAttributeDesignator> or an <AttributeSelector> element pointing into the
4524 <Action> element of the request **context**.

4525 urn:oasis:names:tc:xacml:1.0:action:action-id
4526 Action namespace
4527 urn:oasis:names:tc:xacml:1.0:action:action-namespace
4528 Implied action. This is the value for action-id attribute when action is implied.
4529 urn:oasis:names:tc:xacml:1.0:action:implied-action

4530 B.8. Environment attributes

4531 These identifiers indicate *attributes* of the *environment* within which the *decision request* is to be
4532 evaluated. When used in the *decision request*, they SHALL appear in the <Environment>
4533 element of the request *context*. They SHALL be accessed via an
4534 <EnvironmentAttributeDesignator> or an <AttributeSelector> element pointing into
4535 the <Environment> element of the request *context*.

4536 This identifier indicates the current time at the *PDP*. In practice it is the time at which the request
4537 *context* was created.

4538 urn:oasis:names:tc:xacml:1.0:environment:current-time
4539 urn:oasis:names:tc:xacml:1.0:environment:current-date
4540 urn:oasis:names:tc:xacml:1.0:environment:current-dateTime

4541 B.9. Status codes

4542 The following status code identifiers are defined.

4543 This identifier indicates success.

4544 urn:oasis:names:tc:xacml:1.0:status:ok

4545 This identifier indicates that attributes necessary to make a policy decision were not available.

4546 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

4547 This identifier indicates that some attribute value contained a syntax error, such as a letter in a
4548 numeric field.

4549 urn:oasis:names:tc:xacml:1.0:status:syntax-error

4550 This identifier indicates that an error occurred during policy evaluation. An example would be
4551 division by zero.

4552 urn:oasis:names:tc:xacml:1.0:status:processing-error

4553 B.10. Combining algorithms

4554 The deny-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

4555 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

4556 The deny-overrides policy-combining algorithm has the following value for
4557 policyCombiningAlgId:

4558 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides

4559 The permit-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

4560 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides

4561 The permit-overrides policy-combining algorithm has the following value for
4562 policyCombiningAlgId:

4563 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides

4564 The first-applicable rule-combining algorithm has the following value for ruleCombiningAlgId:
4565 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable
4566 The first-applicable policy-combining algorithm has the following value for
4567 policyCombiningAlgId:
4568 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable
4569 The only-one-applicable-policy policy-combining algorithm has the following value for
4570 policyCombiningAlgId:
4571 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable

4572

Appendix C. Combining algorithms (normative)

4573

This section contains a description of the rule-combining and policy-combining algorithms specified by XACML.

4574

4575

C.1. Deny-overrides

4576

The following specification defines the “Deny-overrides” *rule-combining algorithm* of a *policy*.

4577

4578

4579

4580

4581

4582

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".

4583

4584

4585

4586

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.

4587

4588

4589

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".

4590

The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

4591

4592

4593

4594

4595

4596

4597

4598

4599

4600

4601

4602

4603

4604

4605

4606

4607

4608

4609

4610

4611

4612

4613

4614

4615

4616

4617

4618

4619

4620

```
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;
    }
  }
}
```

```

4621     }
4622     }
4623     if (potentialDeny)
4624     {
4625         return Indeterminate;
4626     }
4627     if (atLeastOnePermit)
4628     {
4629         return Permit;
4630     }
4631     if (atLeastOneError)
4632     {
4633         return Indeterminate;
4634     }
4635     return NotApplicable;
4636 }

```

4637 The following specification defines the “Deny-overrides” *policy-combining algorithm* of a *policy*
4638 **set**.

4639 In the entire set of *policies* in the *policy set*, if any *policy* evaluates to "Deny", then the
4640 result of the *policy* combination SHALL be "Deny". In other words, "Deny" takes
4641 precedence, regardless of the result of evaluating any of the other *policies* in the *policy*
4642 **set**. If all *policies* are found to be "NotApplicable" to the *decision request*, then the
4643 *policy set* SHALL evaluate to "NotApplicable".

4644 If an error occurs while evaluating the *target* of a *policy*, or a reference to a *policy* is
4645 considered invalid or the *policy* evaluation results in "Indeterminate", then the *policy set*
4646 SHALL evaluate to "Deny".

4647 The following pseudo-code represents the evaluation strategy of this *policy-combining algorithm*.

```

4648 Decision denyOverridesPolicyCombiningAlgorithm(Policy policy[])
4649 {
4650     Boolean atLeastOnePermit = false;
4651     for( i=0 ; i < lengthOf(policy) ; i++ )
4652     {
4653         Decision decision = evaluate(policy[i]);
4654         if (decision == Deny)
4655         {
4656             return Deny;
4657         }
4658         if (decision == Permit)
4659         {
4660             atLeastOnePermit = true;
4661             continue;
4662         }
4663         if (decision == NotApplicable)
4664         {
4665             continue;
4666         }
4667         if (decision == Indeterminate)
4668         {
4669             return Deny;
4670         }
4671     }
4672     if (atLeastOnePermit)
4673     {
4674         return Permit;
4675     }
4676     return NotApplicable;
4677 }

```

4678 **Obligations** of the individual *policies* shall be combined as described in Section 7.11.

4679 C.2. Permit-overrides

4680 The following specification defines the "Permit-overrides" *rule-combining algorithm* of a *policy*.

4681 In the entire set of *rules* in the *policy*, if any *rule* evaluates to "Permit", then the result of
4682 the *rule* combination SHALL be "Permit". If any *rule* evaluates to "Deny" and all other
4683 *rules* evaluate to "NotApplicable", then the *policy* SHALL evaluate to "Deny". In other
4684 words, "Permit" takes precedence, regardless of the result of evaluating any of the other
4685 *rules* in the *policy*. If all *rules* are found to be "NotApplicable" to the *decision request*,
4686 then the *policy* SHALL evaluate to "NotApplicable".

4687 If an error occurs while evaluating the *target* or *condition* of a *rule* that contains an *effect*
4688 of "Permit" then the evaluation SHALL continue looking for a result of "Permit". If no other
4689 *rule* evaluates to "Permit", then the *policy* SHALL evaluate to "Indeterminate", with the
4690 appropriate error status.

4691 If at least one *rule* evaluates to "Deny", all other *rules* that do not have evaluation errors
4692 evaluate to "Deny" or "NotApplicable" and all *rules* that do have evaluation errors contain
4693 an *effect* value of "Deny", then the *policy* SHALL evaluate to "Deny".

4694 The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

```
4695 Decision permitOverridesRuleCombiningAlgorithm(Rule rule[])
4696 {
4697     Boolean atLeastOneError = false;
4698     Boolean potentialPermit = false;
4699     Boolean atLeastOneDeny = false;
4700     for( i=0 ; i < lengthOf(rule) ; i++ )
4701     {
4702         Decision decision = evaluate(rule[i]);
4703         if (decision == Deny)
4704         {
4705             atLeastOneDeny = true;
4706             continue;
4707         }
4708         if (decision == Permit)
4709         {
4710             return Permit;
4711         }
4712         if (decision == NotApplicable)
4713         {
4714             continue;
4715         }
4716         if (decision == Indeterminate)
4717         {
4718             atLeastOneError = true;
4719
4720             if (effect(rule[i]) == Permit)
4721             {
4722                 potentialPermit = true;
4723             }
4724             continue;
4725         }
4726     }
4727     if (potentialPermit)
4728     {
4729         return Indeterminate;
4730     }
4731     if (atLeastOneDeny)
4732     {
4733         return Deny;
```

```

4734     }
4735     if (atLeastOneError)
4736     {
4737         return Indeterminate;
4738     }
4739     return NotApplicable;
4740 }

```

4741 The following specification defines the "Permit-overrides" *policy-combining algorithm* of a *policy*
4742 **set**.

4743 In the entire set of *policies* in the *policy set*, if any *policy* evaluates to "Permit", then the
4744 result of the *policy* combination SHALL be "Permit". In other words, "Permit" takes
4745 precedence, regardless of the result of evaluating any of the other *policies* in the *policy*
4746 **set**. If all *policies* are found to be "NotApplicable" to the *decision request*, then the
4747 *policy set* SHALL evaluate to "NotApplicable".

4748 If an error occurs while evaluating the *target* of a *policy*, a reference to a *policy* is
4749 considered invalid or the *policy* evaluation results in "Indeterminate", then the *policy set*
4750 SHALL evaluate to "Indeterminate", with the appropriate error status, provided no other
4751 *policies* evaluate to "Permit" or "Deny".

4752 The following pseudo-code represents the evaluation strategy of this *policy-combining algorithm*.

```

4753 Decision permitOverridesPolicyCombiningAlgorithm(Policy policy[])
4754 {
4755     Boolean atLeastOneError = false;
4756     Boolean atLeastOneDeny = false;
4757     for( i=0 ; i < lengthOf(policy) ; i++ )
4758     {
4759         Decision decision = evaluate(policy[i]);
4760         if (decision == Deny)
4761         {
4762             atLeastOneDeny = true;
4763             continue;
4764         }
4765         if (decision == Permit)
4766         {
4767             return Permit;
4768         }
4769         if (decision == NotApplicable)
4770         {
4771             continue;
4772         }
4773         if (decision == Indeterminate)
4774         {
4775             atLeastOneError = true;
4776             continue;
4777         }
4778     }
4779     if (atLeastOneDeny)
4780     {
4781         return Deny;
4782     }
4783     if (atLeastOneError)
4784     {
4785         return Indeterminate;
4786     }
4787     return NotApplicable;
4788 }

```

4789 **Obligations** of the individual policies shall be combined as described in Section 7.11.

4790

C.3. First-applicable

4791

The following specification defines the "First-Applicable " **rule-combining algorithm** of a **policy**.

4792

4793

4794

4795

4796

4797

4798

Each **rule** SHALL be evaluated in the order in which it is listed in the **policy**. For a particular **rule**, if the **target** matches and the **condition** evaluates to "True", then the evaluation of the **policy** SHALL halt and the corresponding **effect** of the **rule** SHALL be the result of the evaluation of the **policy** (i.e. "Permit" or "Deny"). For a particular **rule** selected in the evaluation, if the **target** evaluates to "False" or the **condition** evaluates to "False", then the next **rule** in the order SHALL be evaluated. If no further **rule** in the order exists, then the **policy** SHALL evaluate to "NotApplicable".

4799

4800

4801

If an error occurs while evaluating the **target** or **condition** of a **rule**, then the evaluation SHALL halt, and the **policy** shall evaluate to "Indeterminate", with the appropriate error status.

4802

The following pseudo-code represents the evaluation strategy of this **rule-combining algorithm**.

4803

4804

4805

4806

4807

4808

4809

4810

4811

4812

4813

4814

4815

4816

4817

4818

4819

4820

4821

4822

4823

4824

4825

4826

```
Decision firstApplicableEffectRuleCombiningAlgorithm(Rule rule[])
{
  for( i = 0 ; i < lengthOf(rule) ; i++ )
  {
    Decision decision = evaluate(rule[i]);
    if (decision == Deny)
    {
      return Deny;
    }
    if (decision == Permit)
    {
      return Permit;
    }
    if (decision == NotApplicable)
    {
      continue;
    }
    if (decision == Indeterminate)
    {
      return Indeterminate;
    }
  }
  return NotApplicable;
}
```

4827

The following specification defines the "First-applicable" **policy-combining algorithm** of a **policy set**.

4828

4829

4830

4831

4832

4833

4834

4835

Each **policy** is evaluated in the order that it appears in the **policy set**. For a particular **policy**, if the **target** evaluates to "True" and the **policy** evaluates to a determinate value of "Permit" or "Deny", then the evaluation SHALL halt and the **policy set** SHALL evaluate to the **effect** value of that **policy**. For a particular **policy**, if the **target** evaluate to "False", or the **policy** evaluates to "NotApplicable", then the next **policy** in the order SHALL be evaluated. If no further **policy** exists in the order, then the **policy set** SHALL evaluate to "NotApplicable".

4836

4837

4838

4839

If an error were to occur when evaluating the **target**, or when evaluating a specific **policy**, the reference to the **policy** is considered invalid, or the **policy** itself evaluates to "Indeterminate", then the evaluation of the **policy-combining algorithm** shall halt, and the **policy set** shall evaluate to "Indeterminate" with an appropriate error status.

4840 The following pseudo-code represents the evaluation strategy of this **policy-combination**
4841 **algorithm**.

```
4842 Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy policy[])
4843 {
4844     for( i = 0 ; i < lengthOf(policy) ; i++ )
4845     {
4846         Decision decision = evaluate(policy[i]);
4847         if(decision == Deny)
4848         {
4849             return Deny;
4850         }
4851         if(decision == Permit)
4852         {
4853             return Permit;
4854         }
4855         if (decision == NotApplicable)
4856         {
4857             continue;
4858         }
4859         if (decision == Indeterminate)
4860         {
4861             return Indeterminate;
4862         }
4863     }
4864     return NotApplicable;
4865 }
```

4866 **Obligations** of the individual policies shall be combined as described in Section 7.11.

4867 C.4. Only-one-applicable

4868 The following specification defines the "Only-one-applicable" **policy-combining algorithm** of a
4869 **policy set**.

4870 In the entire set of policies in the **policy set**, if no **policy** is considered applicable by virtue of their
4871 **targets**, then the result of the policy combination algorithm SHALL be "NotApplicable". If more than
4872 one policy is considered applicable by virtue of their **targets**, then the result of the policy
4873 combination algorithm SHALL be "Indeterminate".

4874 If only one **policy** is considered applicable by evaluation of the **policy targets**, then the result of
4875 the **policy-combining algorithm** SHALL be the result of evaluating the **policy**.

4876 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is considered
4877 invalid or the **policy** evaluation results in "Indeterminate", then the **policy set** SHALL evaluate to
4878 "Indeterminate", with the appropriate error status.

4879 The following pseudo-code represents the evaluation strategy of this policy combining algorithm.

```
4880 Decision onlyOneApplicablePolicyPolicyCombiningAlogrithm(Policy policy[])
4881 {
4882     Boolean          atLeastOne      = false;
4883     Policy           selectedPolicy = null;
4884     ApplicableResult appResult;
4885
4886     for ( i = 0; i < lengthOf(policy) ; i++ )
4887     {
4888         appResult = isApplicable(policy[i]);
4889
4890         if ( appResult == Indeterminate )
4891         {
```



```
4892     return Indeterminate;
4893 }
4894 if( appResult == Applicable )
4895 {
4896     if ( atLeastOne )
4897     {
4898         return Indeterminate;
4899     }
4900     else
4901     {
4902         atLeastOne      = true;
4903         selectedPolicy = policy[i];
4904     }
4905 }
4906 if ( appResult == NotApplicable )
4907 {
4908     continue;
4909 }
4910 }
4911 if ( atLeastOne )
4912 {
4913     return evaluate(selectedPolicy);
4914 }
4915 else
4916 {
4917     return NotApplicable;
4918 }
4919 }
4920
```

4921

Appendix D. Acknowledgments

4922

The following individuals contributed to the development of the specification:

4923

Anne Anderson

4924

Bill Parducci

4925

Carlisle Adams

4926

Daniel Engovatov

4927

Don Flinn

4928

Ernesto Damiani

4929

Gerald Brose

4930

Hal Lockhart

4931

James MacLean

4932

John Merrells

4933

Ken Yagen

4934

Konstantin Beznosov

4935

Michiharu Kudo

4936

Pierangela Samarati

4937

Pirasenna Velandai Thiyagarajan

4938

Polar Humenn

4939

Satoshi Hada

4940

Sekhar Vajjhala

4941

Seth Proctor

4942

Simon Godik

4943

Steve Anderson

4944

Steve Crocker

4945

Suresh Damodaran

4946

Tim Moses

4947

4948

Appendix E. Revision history

Rev	Date	By whom	What
OS V1.0	18 Feb 2003	XACML Technical Committee	OASIS Standard

4949

Appendix F. Notices

4950

4951 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
4952 that might be claimed to pertain to the implementation or use of the technology described in this
4953 document or the extent to which any license under such rights might or might not be available;
4954 neither does it represent that it has made any effort to identify any such rights. Information on
4955 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
4956 website. Copies of claims of rights made available for publication and any assurances of licenses to
4957 be made available, or the result of an attempt made to obtain a general license or permission for
4958 the use of such proprietary rights by implementors or users of this specification, can be obtained
4959 from the OASIS Executive Director.

4960 OASIS has been notified of intellectual property rights claimed in regard to some or all of the
4961 contents of this specification. For more information consult the online list of claimed rights.

4962 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
4963 applications, or other proprietary rights which may cover technology that may be required to
4964 implement this specification. Please address the information to the OASIS Executive Director.

4965 Copyright (C) OASIS Open 2003. All Rights Reserved.

4966 This document and translations of it may be copied and furnished to others, and derivative works
4967 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
4968 published and distributed, in whole or in part, without restriction of any kind, provided that the above
4969 copyright notice and this paragraph are included on all such copies and derivative works. However,
4970 this document itself may not be modified in any way, such as by removing the copyright notice or
4971 references to OASIS, except as needed for the purpose of developing OASIS specifications, in
4972 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights
4973 document must be followed, or as required to translate it into languages other than English.

4974 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
4975 successors or assigns.

4976 This document and the information contained herein is provided on an "AS IS" basis and OASIS
4977 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
4978 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY
4979 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
4980 PARTICULAR PURPOSE.