



1

2

eXtensible Access Control Markup Language (XACML) Version 1.1

3

4

Committee Specification, 07 August 2003

5

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

6

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

7

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

8

Editors:

9

Simon Godik, Overxeer

10

Tim Moses, Entrust

11

Committee members:

12

Anne Anderson, Sun Microsystems

13

Antony Nadalin, IBM

14

Bill Parducci, Overxeer

15

Daniel Engovatov, BEA Systems

16

Hal Lockhart, BEA Systems

17

Michiharu Kudo, IBM

18

Polar Humenn, Self

19

Simon Godik, Overxeer

20

Steve Anderson, OpenNetwork

21

Steve Crocker, Pervasive Security Systems

22

Tim Moses, Entrust

23

24

Abstract:

25

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

26

language.

27

28

Status:

29

This version of the specification is a Committee Specification.

30

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

31

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

32

and send comments there. To subscribe, send an email message to [xacml-comment-](mailto:xacml-comment-request@lists.oasis-open.org)

33

request@lists.oasis-open.org with the word "subscribe" as the body of the message.

34

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

36 Table of contents

37	1.	Introduction (non-normative).....	10
38	1.1.	Glossary	10
39	1.1.1	Preferred terms.....	10
40	1.1.2	Related terms	11
41	1.2.	Notation	12
42	1.3.	Schema organization and namespaces	12
43	2.	Background (non-normative)	13
44	2.1.	Requirements	13
45	2.2.	Rule and policy combining	14
46	2.3.	Combining algorithms.....	14
47	2.4.	Multiple subjects	15
48	2.5.	Policies based on subject and resource attributes.....	15
49	2.6.	Multi-valued attributes	15
50	2.7.	Policies based on resource contents.....	16
51	2.8.	Operators.....	16
52	2.9.	Policy distribution.....	17
53	2.10.	Policy indexing.....	17
54	2.11.	Abstraction layer.....	17
55	2.12.	Actions performed in conjunction with enforcement.....	18
56	3.	Models (non-normative).....	18
57	3.1.	Data-flow model.....	18
58	3.2.	XACML context.....	20
59	3.3.	Policy language model	20
60	3.3.1	Rule	21
61	3.3.2	Policy	23
62	3.3.3	Policy set	24
63	4.	Examples (non-normative).....	25
64	4.1.	Example one.....	25
65	4.1.1	Example policy	25
66	4.1.2	Example request context.....	27
67	4.1.3	Example response context	28
68	4.2.	Example two	28
69	4.2.1	Example medical record instance	29
70	4.2.2	Example request context.....	30
71	4.2.3	Example plain-language rules	32

72	4.2.4	Example XACML rule instances.....	32
73	5.	Policy syntax (normative, with the exception of the schema fragments).....	46
74	5.1.	Element <PolicySet>	46
75	5.2.	Element <Description>	47
76	5.3.	Element <PolicySetDefaults>	47
77	5.4.	Element <XPathVersion>	48
78	5.5.	Element <Target>	48
79	5.6.	Element <Subjects>	49
80	5.7.	Element <Subject>	49
81	5.8.	Element <AnySubject>	49
82	5.9.	Element <SubjectMatch>	49
83	5.10.	Element <Resources>	50
84	5.11.	Element <Resource>.....	50
85	5.12.	Element <AnyResource>.....	51
86	5.13.	Element <ResourceMatch>	51
87	5.14.	Element <Actions>	52
88	5.15.	Element <Action>	52
89	5.16.	Element <AnyAction>	52
90	5.17.	Element <ActionMatch>	52
91	5.18.	Element <PolicySetIdReference>	53
92	5.19.	Element <PolicyIdReference>	53
93	5.20.	Element <Policy>.....	53
94	5.21.	Element <PolicyDefaults>	55
95	5.22.	Element <Rule>	55
96	5.23.	Simple type EffectType.....	56
97	5.24.	Element <Condition>	56
98	5.25.	Element <Apply>	56
99	5.26.	Element <Function>	57
100	5.27.	Complex type AttributeDesignatorType.....	57
101	5.28.	Element <SubjectAttributeDesignator>	58
102	5.29.	Element <ResourceAttributeDesignator>	59
103	5.30.	Element <ActionAttributeDesignator>	60
104	5.31.	Element <EnvironmentAttributeDesignator>	60
105	5.32.	Element <AttributeSelector>	61
106	5.33.	Element <AttributeValue>.....	62
107	5.34.	Element <Obligations>	63
108	5.35.	Element <Obligation>	63

109	5.36.	Element <AttributeAssignment>	64
110	6.	Context syntax (normative with the exception of the schema fragments)	64
111	6.1.	Element <Request>	64
112	6.2.	Element <Subject>	65
113	6.3.	Element <Resource>	66
114	6.4.	Element <ResourceContent>	66
115	6.5.	Element <Action>	67
116	6.6.	Element <Environment>	67
117	6.7.	Element <Attribute>	67
118	6.8.	Element <AttributeValue>	68
119	6.9.	Element <Response>	68
120	6.10.	Element <Result>	69
121	6.11.	Element <Decision>	70
122	6.12.	Element <Status>	70
123	6.13.	Element <StatusCode>	71
124	6.14.	Element <StatusMessage>	71
125	6.15.	Element <StatusDetail>	71
126	7.	Functional requirements (normative)	72
127	7.1.	Policy enforcement point	72
128	7.2.	Base policy	72
129	7.3.	Target evaluation	73
130	7.4.	Condition evaluation	73
131	7.5.	Rule evaluation	73
132	7.6.	Policy evaluation	73
133	7.7.	Policy Set evaluation	74
134	7.8.	Hierarchical resources	75
135	7.9.	Attributes	76
136	7.9.1.	Attribute Matching	76
137	7.9.2.	Attribute Retrieval	76
138	7.9.3.	Environment Attributes	77
139	7.10.	Authorization decision	77
140	7.11.	Obligations	77
141	7.12.	Unsupported functionality	78
142	7.13.	Syntax and type errors	78
143	8.	XACML extensibility points (non-normative)	78
144	8.1.	Extensible XML attribute types	78
145	8.2.	Structured attributes	79

146	9.	Security and privacy considerations (non-normative).....	79
147	9.1.	Threat model	79
148	9.1.1.	Unauthorized disclosure	80
149	9.1.2.	Message replay	80
150	9.1.3.	Message insertion	80
151	9.1.4.	Message deletion	80
152	9.1.5.	Message modification.....	80
153	9.1.6.	NotApplicable results.....	81
154	9.1.7.	Negative rules.....	81
155	9.2.	Safeguards	82
156	9.2.1.	Authentication.....	82
157	9.2.2.	Policy administration.....	82
158	9.2.3.	Confidentiality	82
159	9.2.4.	Policy integrity	83
160	9.2.5.	Policy identifiers.....	83
161	9.2.6.	Trust model.....	84
162	9.2.7.	Privacy.....	84
163	10.	Conformance (normative).....	84
164	10.1.	Introduction.....	84
165	10.2.	Conformance tables	84
166	10.2.1.	Schema elements.....	85
167	10.2.2.	Identifier Prefixes.....	86
168	10.2.3.	Algorithms.....	86
169	10.2.4.	Status Codes	86
170	10.2.5.	Attributes.....	87
171	10.2.6.	Identifiers	87
172	10.2.7.	Data-types	87
173	10.2.8.	Functions	88
174	11.	References	92
175	Appendix A.	Standard data-types, functions and their semantics (normative)	94
176	A.1.	Introduction	94
177	A.2.	Primitive types	94
178	A.3.	Structured types.....	95
179	A.4.	Representations.....	95
180	A.5.	Bags.....	96
181	A.6.	Expressions	96
182	A.7.	Element <AttributeValue>.....	97

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

220	C.5. First-applicable.....	128
221	C.6. Only-one-applicable	130
222	Appendix D. Acknowledgments.....	132
223	Appendix E. Revision history.....	133
224	Appendix F. Notices	134
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 **Type Unification** - The method by which two type expressions are "unified". The type expressions
281 are matched along their structure. Where a type variable appears in one expression it is then
282 "unified" to represent the corresponding structure element of the other expression, be it another
283 variable or subexpression. All variable assignments must remain consistent in both structures.
284 Unification fails if the two expressions cannot be aligned, either by having dissimilar structure, or by
285 having instance conflicts, such as a variable needs to represent both "xs:string" and "xs:integer".
286 For a full explanation of **type unification**, please see [Hancock].

287 1.1.2 Related terms

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

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

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

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

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

295

1.2. Notation

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

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

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

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

307 `Listings of XACML schemas appear like this.`

308

309 `Example code listings appear like this.`

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

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

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

322 1.3. Schema organization and namespaces

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

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

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

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

327 The XML Signature [\[DS\]](#) is imported into the XACML schema and is associated with the following
328 XML namespace:

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

330

2. Background (non-normative)

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

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

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

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

357

2.1. Requirements

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

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

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

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

382 2.2. Rule and policy combining

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

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

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

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

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

402 2.3. Combining algorithms

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

- 410 • Deny-overrides (Ordered and Unordered),
- 411 • Permit-overrides (Ordered and Unordered),

- 412 • First applicable and
- 413 • Only-one-applicable.

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

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

427 2.4. Multiple subjects

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

434 2.5. Policies based on subject and resource attributes

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

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

451 2.6. Multi-valued attributes

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

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

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

460 2.7. Policies based on resource contents

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

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

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

471 2.8. Operators

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

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

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

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

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

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

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

501 2.9. Policy distribution

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

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

513 2.10. Policy indexing

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

519 Two approaches are supported:

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

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

529 2.11. Abstraction layer

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

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

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

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

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

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

552 2.12. Actions performed in conjunction with enforcement

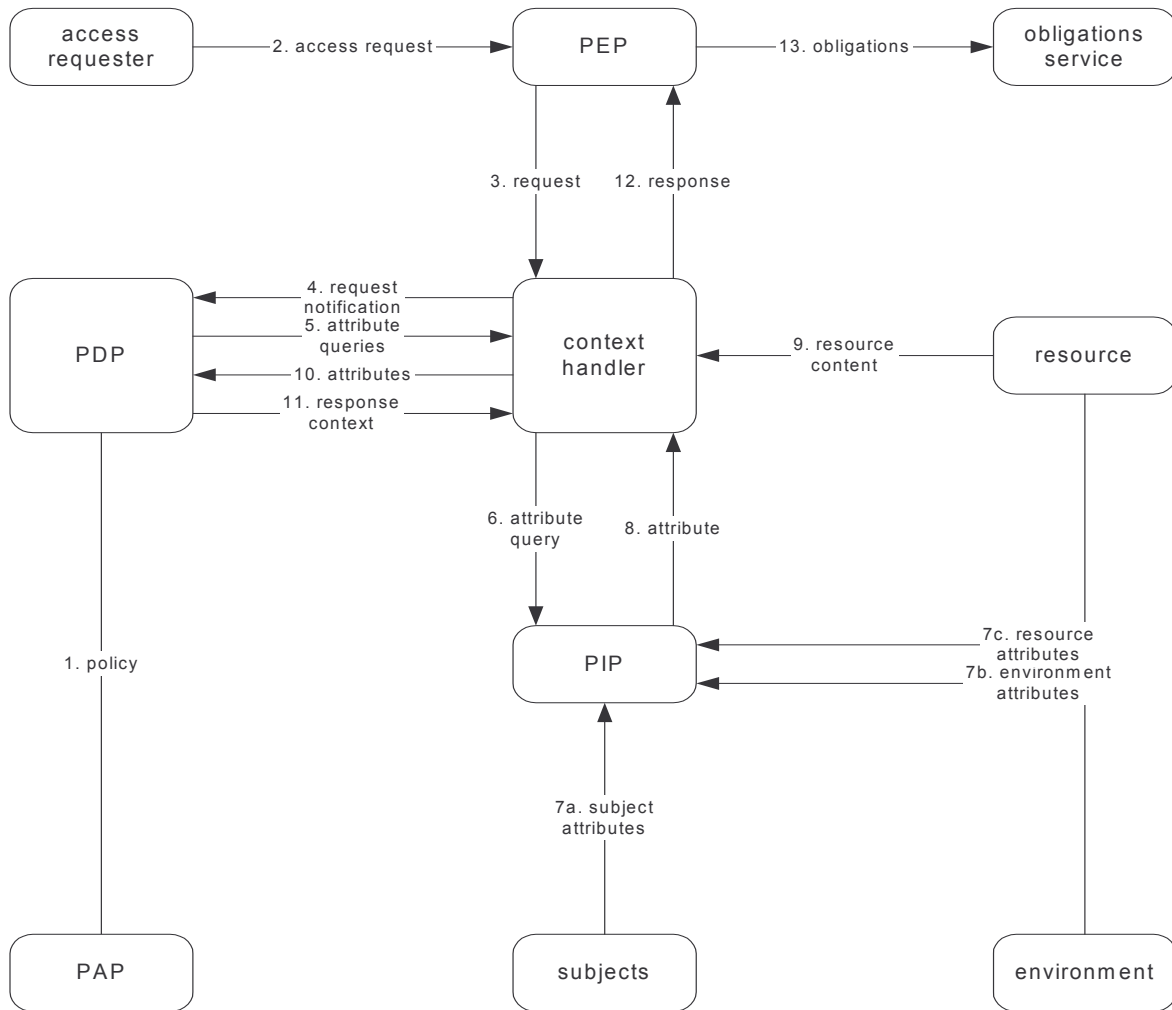
553 In many applications, policies specify actions that MUST be performed, either instead of, or in
554 addition to, actions that MAY be performed. This idea was described by Sloman [Sloman94].
555 XACML provides facilities to specify actions that MUST be performed in conjunction with policy
556 evaluation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

562 3. Models (non-normative)

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

564 3.1. Data-flow model

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



566

567

Figure 1 - Data-flow diagram

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

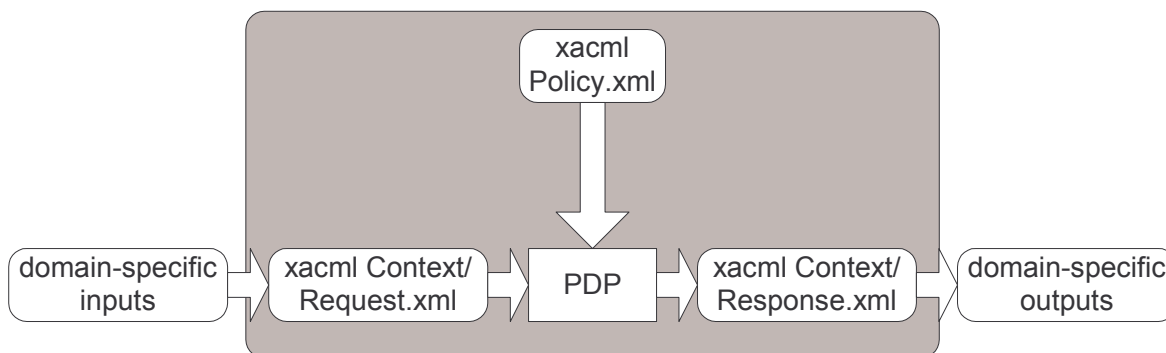
573 The model operates by the following steps.

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

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

594 3.2. XACML context

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



606
 607 **Figure 2 - XACML context**

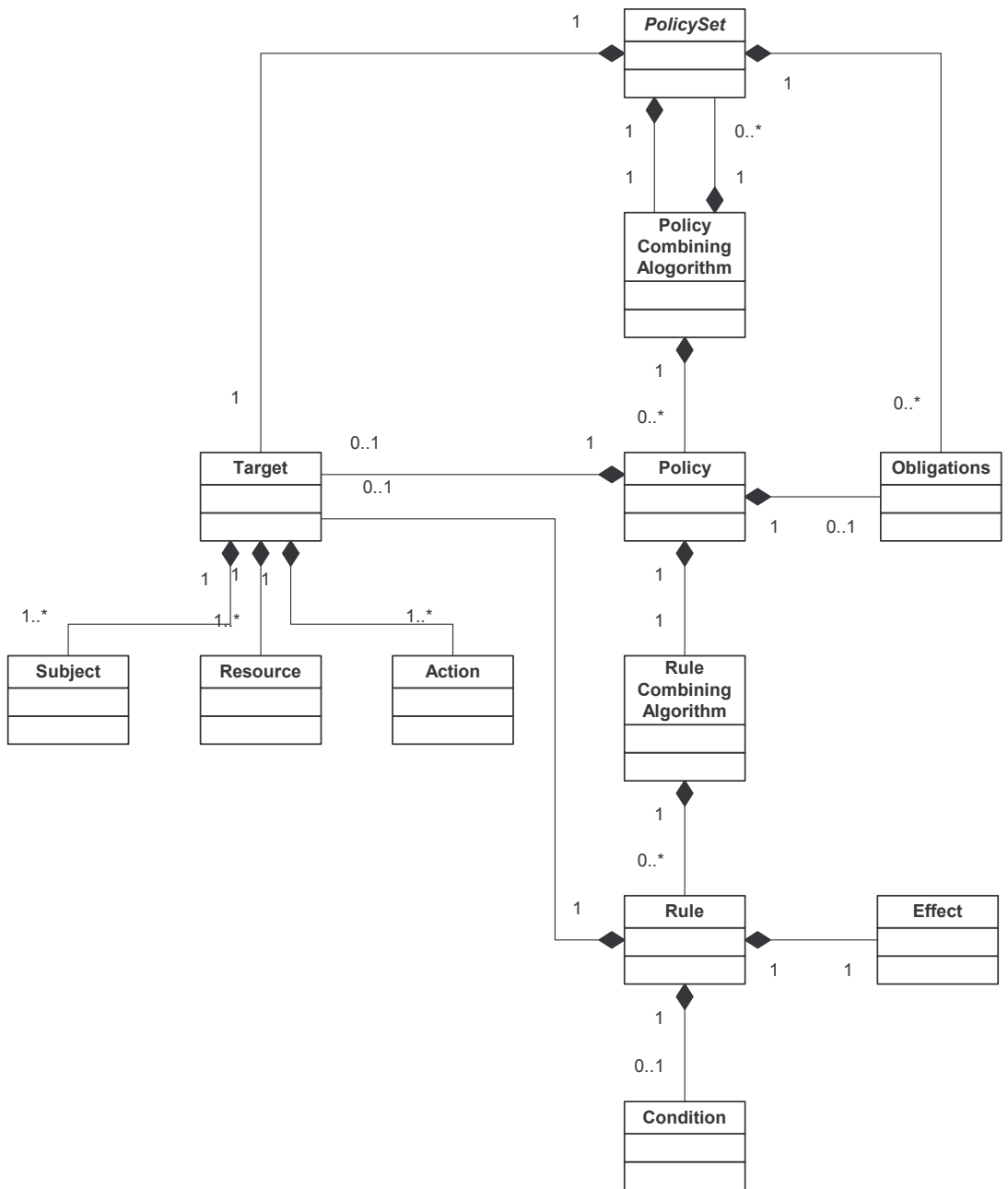
608 Note: The **PDP** may be implemented such that it uses a processed form of the XML files.
 609 See Section 7.9 for a more detailed discussion of the request **context**.

610 3.3. Policy language model

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

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

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



616

617

Figure 3 - Policy language model

618

3.3.1 Rule

619

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:

622

- 623 • a *target*;
- 624 • an *effect*; and
- 625 • a *condition*.

626 These are discussed in the following sub-sections.

627 3.3.1.1. Rule target

628 The *target* defines the set of:

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

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

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

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

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

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

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

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

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

665 All three options are supported in XACML.

666 3.3.1.2. Effect

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

669 3.3.1.3. Condition

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

672 3.3.2 Policy

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

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

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

681 3.3.2.1. Policy target

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

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

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

703 3.3.2.2. Rule-combining algorithm

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

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

709 3.3.2.3. Obligations

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

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

714 3.3.3 Policy set

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

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

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

722 3.3.3.1. Policy-combining algorithm

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

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

728 3.3.3.2. Obligations

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

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

733

734

4. Examples (non-normative)

735

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**.

736

737

738

739

4.1. Example one

740

4.1.1 Example policy

741

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

742

743

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

744

745

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**.

746

747

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">
```

748

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

749

750

[p02] introduces the XACML Policy itself.

751

[p03-p05] are XML namespace declarations.

752

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

753

[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**.

754

755

[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**.

756

757

758

759

760

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

761

[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>
```

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

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

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

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

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

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

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

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

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

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

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

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

795 4.1.2 Example request context

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

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

801 In XACML, the information in the **decision request** is formatted into a **request context** statement
802 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">
```

803 [c01-c05] are the header for the **request context**, and are used the same way as the header for the
804 **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>
```

805 The `<Subject>` element contains one or more **attributes** of the entity making the **access** request.
806 There can be multiple **subjects**, and each **subject** can have multiple **attributes**. In this case, in
807 [c06-c11], there is only one **subject**, and the **subject** has only one **attribute**: the **subject's** identity,
808 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>
```

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

811 per **decision request**. Lines [c13-c16] contain the one **attribute** of the **resource**
812 to which Bart Simpson has requested **access**: the **resource** unix file-system path-
813 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>
```

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

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

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

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

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

828 4.1.3 Example response context

829 As a result, there is no **rule** in this **policy** that returns a "Permit" result for this request. The **rule-**
830 **combining algorithm** for the **policy** specifies that, in this case, a result of "NotApplicable" should
831 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">
```

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

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

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

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

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

838 4.2. Example two

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

842

4.2.1 Example medical record instance

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

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

```

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

```

927 4.2.2 Example request context

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

```

931 [01] <?xml version="1.0" encoding="UTF-8"?>
932 [02] <Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
933 [03] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
934 [04] <Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-
935 category:access-subject">
936 [05]   <Attribute AttributeId=
937 [06]     "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
938 [07]     DataType=
939 [08]       "urn:oasis:names:tc:xacml:1.0:data-type:x500name"
940 [09]     Issuer="www.medico.com"
941 [10]     IssueInstant="2001-12-17T09:30:47-05:00">
942 [11]     <AttributeValue>CN=Julius Hibbert</AttributeValue>
943 [12]   </Attribute>
944 [13]   <Attribute AttributeId=
945 [14]     "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
946 [15]     DataType="http://www.w3.org/2001/XMLSchema#string"
947 [16]     Issuer="www.medico.com"
948 [17]     IssueInstant="2001-12-17T09:30:47-05:00">
949 [18]     <AttributeValue>physician</AttributeValue>
950 [19]   </Attribute>
951 [20]   <Attribute AttributeId=
952 [21]     "urn:oasis:names:tc:xacml:1.0:example:attribute:physician-id"
953 [22]     DataType="http://www.w3.org/2001/XMLSchema#string"
954 [23]     Issuer="www.medico.com"
955 [24]     IssueInstant="2001-12-17T09:30:47-05:00">
956 [25]     <AttributeValue>jh1234</AttributeValue>
957 [26]   </Attribute>
958 [27] </Subject>
959 [28] <Resource>
960 [29]   <ResourceContent>
961 [30]     <md:record
962 [31]       xmlns:md="//http:www.medico.com/schemas/record.xsd">

```

```

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

```

1002 [02]-[03] Standard namespace declarations.

1003 [04]-[27] **Subject** attributes are placed in the **Subject** section of the **Request**. Each **attribute**
1004 consists of the **attribute** meta-data and the **attribute** value.

1005 [04] Each **Subject** element has **SubjectCategory** xml attribute. The value of this attribute
1006 describes the role that the **subject** plays in making the **decision request**. The value of "access-
1007 subject" denotes the identity for which the request was issued.

1008 [05]-[12] **Subject** subject-id **attribute**.

1009 [13]-[19] **Subject** role **attribute**.

1010 [20]-[26] **Subject** physician-id **attribute**.

1011 [28]-[62] **Resource** attributes are placed in the **Resource** section of the **Request**. Each **attribute**
1012 consists of **attribute** meta-data and an **attribute** value.

1013 [29]-[36] **Resource** content. The XML document that is being requested is placed here.

1014 [38]-[46] **Resource** identifier.

1015 [47]-[61] The **Resource** is identified with an Xpointer expression that names the URI of the file that
1016 is accessed, the target namespace of the document, and the XPath location path to the specific
1017 element.

1018 [47]-[54] The XPath location path in the “resource-id” attribute is extracted and placed in the
1019 xpath attribute.

1020 [55]-[61] **Resource** target-namespace **attribute**.

1021 [63]-[69] **Action attributes** are placed in the Action section of the Request.

1022 [64]-[68] **Action** identifier.

1023 4.2.3 Example plain-language rules

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

1025 Rule 1: A person, identified by his or her patient number, may read any record for which he
1026 or she is the designated patient.

1027 Rule 2: A person may read any record for which he or she is the designated parent or
1028 guardian, and for which the patient is under 16 years of age.

1029 Rule 3: A physician may write to any medical element for which he or she is the designated
1030 primary care physician, provided an email is sent to the patient.

1031 Rule 4: An administrator shall not be permitted to read or write to medical elements of a
1032 patient record.

1033 These **rules** may be written by different **PAPs** operating independently, or by a single **PAP**.

1034 4.2.4 Example XACML rule instances

1035 4.2.4.1. Rule 1

1036 Rule 1 illustrates a simple **rule** with a single <Condition> element. The following XACML
1037 <Rule> instance expresses Rule 1:

```
1038 [01] <?xml version="1.0" encoding="UTF-8"?>  
1039 [02] <Rule  
1040 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"  
1041 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
1042 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"  
1043 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"  
1044 [07]   RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"  
1045 [08]   Effect="Permit">  
1046 [09]   <Description>  
1047 [10]     A person may read any medical record in the  
1048 [11]     http://www.medico.com/schemas/record.xsd namespace  
1049 [12]     for which he or she is a designated patient  
1050 [13]   </Description>  
1051 [14]   <Target>  
1052 [15]     <Subjects>  
1053 [16]       <AnySubject/>  
1054 [17]     </Subjects>  
1055 [18]     <Resources>  
1056 [20]       <Resource>  
1057 [21]         <!-- match document target namespace -->
```



```

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

```

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

1115 [07] **Rule** identifier.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1200 4.2.4.2. Rule 2

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

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

```

1206 [02] <Rule
1207 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1208 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1209 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1210 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1211 [07]   RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
1212 [08]   Effect="Permit">
1213 [09]   <Description>
1214 [10]     A person may read any medical record in the
1215 [11]     http://www.medico.com/records.xsd namespace
1216 [12]     for which he or she is the designated parent or guardian,
1217 [13]     and for which the patient is under 16 years of age
1218 [14]   </Description>
1219 [15]   <Target>
1220 [16]     <Subjects>
1221 [17]       <AnySubject/>
1222 [18]     </Subjects>
1223 [19]     <Resources>
1224 [20]       <Resource>
1225 [21]         <!-- match document target namespace -->
1226 [22]         <ResourceMatch
1227 [23]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1228 [24]             <AttributeValue
1229 [25]               DataType="http://www.w3.org/2001/XMLSchema#string">
1230 [26]                 http://www.medico.com/schemas/record.xsd
1231 [27]               </AttributeValue>
1232 [28]               <ResourceAttributeDesignator AttributeId=
1233 [29]                 "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1234 [30]               DataType="http://www.w3.org/2001/XMLSchema#string"/>
1235 [31]             </ResourceMatch>
1236 [32]             <!-- match requested xml element -->
1237 [33]             <ResourceMatch
1238 [34]               MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1239 [35]                 <AttributeValue
1240 [36]                   DataType="http://www.w3.org/2001/XMLSchema#string">/md:record</AttributeV
1241 [37]                   alue>
1242 [38]                 <ResourceAttributeDesignator AttributeId=
1243 [39]                   "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1244 [40]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1245 [41]               </ResourceMatch>
1246 [42]             </Resource>
1247 [43]           </Resources>
1248 [44]           <Actions>
1249 [45]             <Action>
1250 [46]               <!-- match 'read' action -->
1251 [47]               <ActionMatch
1252 [48]                 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1253 [49]                   <AttributeValue
1254 [50]                     DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1255 [51]                   <ActionAttributeDesignator AttributeId=
1256 [52]                     "urn:oasis:names:tc:xacml:1.0:action:action-id"
1257 [53]                   DataType="http://www.w3.org/2001/XMLSchema#string"/>
1258 [54]                 </ActionMatch>
1259 [55]               </Action>
1260 [56]             </Actions>
1261 [57]           </Target>
1262 [58]         <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
1263 [59]           <!-- compare parent-guardian-id subject attribute with
1264 [60]             the value in the document -->
1265 [61]           <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1266 [62]             equal">
1267 [63]             <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1268 [64]             and-only">

```

```

1269 [53] <!-- parent-guardian-id subject attribute -->
1270 [54] <SubjectAttributeDesignator AttributeId=
1271 [55] "urn:oasis:names:tc:xacml:1.0:examples:attribute:
1272 [56] parent-guardian-id"
1273 [57] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1274 [58] </Apply>
1275 [59] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1276 [60] and-only">
1277 [61] <!-- parent-guardian-id element in the document -->
1278 [62] <AttributeSelector RequestContextPath=
1279 [63] "//md:record/md:parentGuardian/md:parentGuardianId/text()"
1280 [64] DataType="http://www.w3.org/2001/XMLSchema#string">
1281 [65] </AttributeSelector>
1282 [66] </Apply>
1283 [67] </Apply>
1284 [68] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-or-
1285 [69] equal">
1286 [70] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-
1287 [71] and-only">
1288 [72] <EnvironmentAttributeDesignator AttributeId=
1289 [73] "urn:oasis:names:tc:xacml:1.0:environment:current-date"
1290 [74] DataType="http://www.w3.org/2001/XMLSchema#date"/>
1291 [75] </Apply>
1292 [76] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-
1293 [77] yearMonthDuration">
1294 [78] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-
1295 [79] one-and-only">
1296 [80] <!-- patient dob recorded in the document -->
1297 [81] <AttributeSelector RequestContextPath=
1298 [82] "//md:record/md:patient/md:patientDoB/text()"
1299 [83] DataType="http://www.w3.org/2001/XMLSchema#date">
1300 [84] </AttributeSelector>
1301 [85] </Apply>
1302 [86] </Apply>
1303 [87] <AttributeSelector RequestContextPath=
1304 [88] "urn:oasis:names:tc:xacml:1.0:environment:current-date"
1305 [89] DataType="http://www.w3.org/2001/XMLSchema#date"/>
1306 [90] </AttributeSelector>
1307 [91] </Apply>
1308 [92] </Apply>
1309 [93] </Condition>
1310 [94] </Rule>

```

1310 [02]-[47] **Rule** declaration and **rule target**. See Rule 1 in Section 4.2.4.1 for the detailed
1311 explanation of these elements.

1312 [48]-[82] The `Condition` element. **Condition** must evaluate to “True” for the **rule** to be applicable.
1313 This **condition** evaluates the truth of the statement: the requestor is the designated parent or
1314 guardian and the patient is under 16 years of age.

1315 [48] The `Condition` is using the “urn:oasis:names:tc:xacml:1.0:function:and”
1316 function. This is a boolean function that takes one or more boolean arguments (2 in this case) and
1317 performs the logical “AND” operation to compute the truth value of the expression.

1318 [51]-[65] The truth of the first part of the condition is evaluated: The requestor is the designated
1319 parent or guardian. The `Apply` element contains a function invocation. The function name is
1320 contained in the `FunctionId` attribute. The comparison is done with
1321 “urn:oasis:names:tc:xacml:1.0:function:string-equal” that takes 2 arguments of
1322 “http://www.w3.org/2001/XMLSchema#string” data-type.

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

1326 that the **subject attribute** "urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id" in
 1327 the request **context** contains one and only one value.
 1328 "urn:oasis:names:tc:xacml:1.0:function:string-equal" takes an argument
 1329 expression that evaluates to a **bag** of "http://www.w3.org/2001/XMLSchema#string"
 1330 values.

1331 [54] Value of the **subject attribute**
 1332 "urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id" is
 1333 selected from the request **context** with the <SubjectAttributeDesignator> element. This
 1334 expression evaluates to a bag of "http://www.w3.org/2001/XMLSchema#string" values.

1335 [58] "urn:oasis:names:tc:xacml:1.0:function:string-one-and-only" is used to
 1336 ensure that the **bag** of values selected by it's argument contains one and only one value of data-
 1337 type "http://www.w3.org/2001/XMLSchema#string".

1338 [60] The value of the md:parentGuardianId element is selected from the **resource** content with
 1339 the AttributeSelector element. AttributeSelector is a free-form XPath expression,
 1340 pointing into the request **context**. The RequestContextPath XML attribute contains an XPath
 1341 expression over the request **context**. Note that all namespace prefixes in the XPath expression
 1342 are resolved with standard namespace declarations. The AttributeSelector evaluates to the
 1343 **bag** of values of data-type "http://www.w3.org/2001/XMLSchema#string".

1344 [66]-[83] The expression: "the patient is under 16 years of age" is evaluated. The patient is under
 1345 16 years of age if the current date is less than the date computed by adding 16 to the patient's date
 1346 of birth.

1347 [66] "urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal" is used to
 1348 compute the difference of two dates.

1349 [67] "urn:oasis:names:tc:xacml:1.0:function:date-one-and-only" is used to ensure
 1350 that the **bag** of values selected by its argument contains one and only one value of data-type
 1351 "http://www.w3.org/2001/XMLSchema#date".

1352 [68]-[69] Current date is evaluated by selecting the
 1353 "urn:oasis:names:tc:xacml:1.0:environment:current-date" **environment attribute**.

1354 [71] "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" is
 1355 used to compute the date by adding 16 to the patient's date of birth. The first argument is a
 1356 "http://www.w3.org/2001/XMLSchema#date", and the second argument is an
 1357 "http://www.w3.org/TR/2002/WD-xquery-operators-
 1358 20020816#yearMonthDuration".

1359 [73] "urn:oasis:names:tc:xacml:1.0:function:date-one-and-only" is used to ensure
 1360 that the **bag** of values selected by it's argument contains one and only one value of data-type
 1361 "http://www.w3.org/2001/XMLSchema#date".

1362 [75]-[76] The <AttributeSelector> element selects the patient's date of birth by taking the
 1363 XPath expression over the document content.

1364 [79]-[81] Year Month Duration of 16 years.

4.2.4.3. Rule 3

1365
 1366 Rule 3 illustrates the use of an **obligation**. The XACML <Rule> element syntax does not include
 1367 an element suitable for carrying an **obligation**, therefore Rule 3 has to be formatted as a
 1368 <Policy> element.

1369 [01] <?xml version="1.0" encoding="UTF-8"?>

```

1370 [02] <Policy
1371 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1372 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1373 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1374 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1375 [07]   PolicyId="urn:oasis:names:tc:xacml:examples:policyid:3"
1376 [08]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
1377 [09]     rule-combining-algorithm:deny-overrides">
1378 [10] <Description>
1379 [11]   Policy for any medical record in the
1380 [12]   http://www.medico.com/schemas/record.xsd namespace
1381 [13] </Description>
1382 [14] <Target>
1383 [15]   <Subjects>
1384 [16]     <AnySubject/>
1385 [17]   </Subjects>
1386 [18]   <Resources>
1387 [19]     <Resource>
1388 [20]       <!-- match document target namespace -->
1389 [21]       <ResourceMatch
1390 [22]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1391 [23]           <AttributeValue
1392 [24]             DataType="http://www.w3.org/2001/XMLSchema#string">
1393 [25]               http://www.medico.com/schemas/record.xsd
1394 [26]             </AttributeValue>
1395 [27]             <ResourceAttributeDesignator AttributeId=
1396 [28]               "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1397 [29]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1398 [30]           </ResourceMatch>
1399 [31]         </Resource>
1400 [32]       </Resources>
1401 [33]     </Subjects>
1402 [34]   </Target>
1403 [35] <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:3"
1404 [36]   Effect="Permit">
1405 [37]   <Description>
1406 [38]     A physician may write any medical element in a record
1407 [39]     for which he or she is the designated primary care
1408 [40]     physician, provided an email is sent to the patient
1409 [41]   </Description>
1410 [42]   <Target>
1411 [43]     <Subjects>
1412 [44]       <Subject>
1413 [45]         <!-- match subject group attribute -->
1414 [46]         <SubjectMatch
1415 [47]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1416 [48]             <AttributeValue
1417 [49]               DataType="http://www.w3.org/2001/XMLSchema#string">physician</AttributeVa
1418 [50]             lue>
1419 [51]             <SubjectAttributeDesignator AttributeId=
1420 [52]               "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
1421 [53]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1422 [54]           </SubjectMatch>
1423 [55]         </Subject>
1424 [56]       </Subjects>
1425 [57]     </Resources>
1426 [58]   </Target>
1427 [59]   <Resources>
1428 [60]     <Resource>
1429 [61]       <!-- match requested xml element -->
1430 [62]       <ResourceMatch
1431 [63]         MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">

```

```

1432 [56]         <AttributeValue
1433         DataType="http://www.w3.org/2001/XMLSchema#string">
1434 [57]         /md:record/md:medical
1435 [58]         </AttributeValue>
1436 [59]         <ResourceAttributeDesignator AttributeId=
1437 [60]         "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1438         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1439 [61]         </ResourceMatch>
1440 [62]         </Resource>
1441 [63]         </Resources>
1442 [64]         <Actions>
1443 [65]         <Action>
1444 [66]         <!-- match action -->
1445 [67]         <ActionMatch
1446         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1447 [68]         <AttributeValue
1448         DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
1449 [069]         <ActionAttributeDesignator AttributeId=
1450 [070]         "urn:oasis:names:tc:xacml:1.0:action:action-id"
1451         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1452 [071]         </ActionMatch>
1453 [072]         </Action>
1454 [073]         </Actions>
1455 [074]         </Target>
1456 [075]         <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1457         equal">
1458 [076]         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1459         and-only">
1460 [077]         <!-- physician-id subject attribute -->
1461 [078]         <SubjectAttributeDesignator AttributeId=
1462 [079]         "urn:oasis:names:tc:xacml:1.0:example:
1463 [080]         attribute:physician-id"
1464         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1465 [081]         </Apply>
1466 [082]         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1467         and-only">
1468 [083]         <AttributeSelector RequestContextPath=
1469 [084]         "//md:record/md:primaryCarePhysician/md:registrationID/text()"
1470 [085]         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1471 [086]         </Apply>
1472 [087]         </Condition>
1473 [089] </Rule>
1474 [090] <Obligations>
1475 [091] <!-- send e-mail message to the document owner -->
1476 [092] <Obligation ObligationId=
1477 [093]         "urn:oasis:names:tc:xacml:example:obligation:email"
1478 [094]         FulfillOn="Permit">
1479 [095]         <AttributeAssignment AttributeId=
1480 [096]         "urn:oasis:names:tc:xacml:1.0:example:attribute:mailto"
1481 [097]         DataType="http://www.w3.org/2001/XMLSchema#string">
1482 [098]         <AttributeSelector RequestContextPath=
1483 [099]         "//md:/record/md:patient/md:patientContact/md:email"
1484 [100]         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1485 [101]         </AttributeAssignment>
1486 [102]         <AttributeAssignment AttributeId=
1487 [103]         "urn:oasis:names:tc:xacml:1.0:example:attribute:text"
1488 [104]         DataType="http://www.w3.org/2001/XMLSchema#string">
1489 [105]         <AttributeValue
1490         DataType="http://www.w3.org/2001/XMLSchema#string">
1491 [106]         Your medical record has been accessed by:
1492 [107]         </AttributeValue>
1493 [108]         </AttributeAssignment>
1494 [109]         <AttributeAssignment AttributeId=

```



```

1495 [110]         "urn:oasis:names:tc:xacml:example:attribute:text"
1496 [111]         DataType="http://www.w3.org/2001/XMLSchema#string">
1497 [112]         <SubjectAttributeDesignator AttributeId=
1498 [113]         "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
1499         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1500 [114]         </AttributeAssignment>
1501 [115]     </Obligation>
1502 [116] </Obligations>
1503 [117] </Policy>

```

1504 [01]-[09] The `Policy` element includes standard namespace declarations as well as policy specific
1505 parameters, such as `PolicyId` and `RuleCombiningAlgId`.

1506 [07] **Policy** identifier. This parameter is used for the inclusion of the `Policy` in the `PolicySet`
1507 element.

1508 [08]-[09] **Rule combining algorithm** identifier. This parameter is used to compute the combined
1509 outcome of **rule effects** for **rules** that are applicable to the **decision request**.

1510 [10-13] Free-form description of the **policy**.

1511 [14]-[33] **Policy target**. The **policy target** defines a set of applicable decision requests. The
1512 structure of the `Target` element in the `Policy` is identical to the structure of the `Target` element
1513 in the `Rule`. In this case, the **policy target** is a set of all XML documents conforming to the
1514 "http://www.medico.com/schemas/record.xsd" target namespace. For the detailed description of
1515 the `Target` element see Rule 1, Section 4.2.4.1.

1516 [34]-[89] The only `Rule` element included in this `Policy`. Two parameters are specified in the **rule**
1517 header: `RuleId` and `Effect`. For the detailed description of the `Rule` structure see Rule 1,
1518 Section 4.2.4.1.

1519 [41]-[74] A **rule target** narrows down a **policy target**. **Decision requests** with the value of
1520 "urn:oasis:names:tc:xacml:1.0:example:attribute:role" **subject attribute** equal to
1521 "physician" [42]-[51], and that access elements of the medical record that "xpath-node-match"
1522 the "/md:record/md:medical" XPath expression [52]-[63], and that have the value of the
1523 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** equal to "read".

1524 [65]-[73] match the **target** of this **rule**. For a detailed description of the rule target see example 1,
1525 Section 4.2.4.1.

1526 [75]-[87] The `Condition` element. For the **rule** to be applicable to the authorization request,
1527 **condition** must evaluate to True. This **rule condition** compares the value of the
1528 "urn:oasis:names:tc:xacml:1.0:examples:attribute:physician-id" **subject**
1529 **attribute** with the value of the `physician id` element in the medical record that is being
1530 accessed. For a detailed explanation of rule condition see Rule 1, Section 4.2.4.1.

1531 [90]-[116] The `Obligations` element. **Obligations** are a set of operations that must be
1532 performed by the **PEP** in conjunction with an **authorization decision**. An **obligation** may be
1533 associated with a positive or negative **authorization decision**.

1534 [92]-[115] The `Obligation` element consists of the `ObligationId`, the authorization decision
1535 value for which it must fulfill, and a set of attribute assignments.

1536 [92]-[93] `ObligationId` identifies an **obligation**. **Obligation** names are not interpreted by the
1537 **PDP**.

1538 [94] `FulfillOn` attribute defines an **authorization decision** value for which this **obligation** must
1539 be fulfilled.

1540 [95]-[101] **Obligation** may have one or more parameters. The **obligation** parameter
 1541 “urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto” is assigned the value
 1542 from the content of the xml document.

1543 [95-96] AttributeId declares
 1544 “urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto” **obligation** parameter.

1545 [97] The **obligation** parameter data-type is defined.

1546 [98]-[100] The **obligation** parameter value is selected from the content of the XML document that is
 1547 being accessed with the XPath expression over request **context**.

1548 [102]-[108] The **obligation** parameter
 1549 “urn:oasis:names:tc:xacml:1.0:examples:attribute:text” of data-type
 1550 “http://www.w3.org/2001/XMLSchema#string” is assigned the literal value “Your
 1551 medical record has been accessed by:”

1552 [109]-[114] The **obligation** parameter
 1553 “urn:oasis:names:tc:xacml:1.0:examples:attribute:text” of the
 1554 “http://www.w3.org/2001/XMLSchema#string” data-type is assigned the value of the
 1555 “urn:oasis:names:tc:xacml:1.0:subject:subject-id” **subject attribute**.

1556 4.2.4.4. Rule 4

1557 Rule 4 illustrates the use of the "Deny" Effect value, and a Rule with no Condition element.

```

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

```

```

1593 [30]         http://www.medico.com/schemas/record.xsd
1594 [31]         </AttributeValue>
1595 [32]         <ResourceAttributeDesignator AttributeId=
1596 [33]         "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1597         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1598 [34]         </ResourceMatch>
1599 [35]         <!-- match requested xml element -->
1600 [36]         <ResourceMatch
1601         MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1602 [37]         <AttributeValue
1603         DataType="http://www.w3.org/2001/XMLSchema#string">
1604 [38]         /md:record/md:medical
1605 [39]         </AttributeValue>
1606 [40]         <ResourceAttributeDesignator AttributeId=
1607 [41]         "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1608         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1609 [42]         </ResourceMatch>
1610 [43]         </Resource>
1611 [44]     </Resources>
1612 [45]     <Actions>
1613 [46]         <Action>
1614 [47]             <!-- match 'read' action -->
1615 [48]             <ActionMatch
1616             MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1617 [49]             <AttributeValue
1618             DataType="http://www.w3.org/2001/XMLSchema#string">
1619                 read
1620             </AttributeValue>
1621 [50]             <ActionAttributeDesignator AttributeId=
1622 [51]             "urn:oasis:names:tc:xacml:1.0:action:action-id"
1623             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1624 [52]             </ActionMatch>
1625 [53]         </Action>
1626 [54]         <Action>
1627 [55]             <!-- match 'write' action -->
1628 [56]             <ActionMatch
1629             MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1630 [57]             <AttributeValue
1631             DataType="http://www.w3.org/2001/XMLSchema#string">
1632                 write
1633             </AttributeValue>
1634 [58]             <ActionAttributeDesignator AttributeId=
1635 [59]             "urn:oasis:names:tc:xacml:1.0:action:action-id"
1636             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1637 [60]             </ActionMatch>
1638 [61]         </Action>
1639 [62]     </Actions>
1640 [63] </Target>
1641 [64] </Rule>

```

1642 [01]-[08] The **Rule** element declaration. The most important parameter here is *Effect*. See Rule
1643 1, Section 4.2.4.1 for a detailed explanation of the **Rule** structure.

1644 [08] **Rule Effect**. Every **rule** that evaluates to “True” emits **rule effect** as its value that will be
1645 combined later on with other **rule effects** according to the **rule combining algorithm**. This **rule**
1646 **Effect** is “Deny” meaning that according to this rule, access must be denied.

1647 [09]-[13] Free form description of the **rule**.

1648 [14]-[63] **Rule target**. The **Rule target** defines a set of **decision requests** that are applicable to
1649 the **rule**. This **rule** is matched by:

- 1650 • a **decision request** with **subject attribute**
- 1651 "urn:oasis:names:tc:xacml:1.0:examples:attribute:role" equal to
- 1652 "administrator";
- 1653 • the value of **resource attribute**
- 1654 "urn:oasis:names:tc:xacml:1.0:resource:target-namespace" is equal to
- 1655 "http://www.medico.com/schemas/record.xsd"
- 1656 • the value of the requested XML element matches the XPath expression
- 1657 "/md:record/md:medical";
- 1658 • the value of **action attribute** "urn:oasis:names:tc:xacml:1.0:action:action-id" is equal to
- 1659 "read"

1660 See Rule 1, Section 4.2.4.1 for the detailed explanation of the `Target` element.

1661 This **rule** does not have a `Condition` element.

1662 4.2.4.5. Example PolicySet

1663 This section uses the examples of the previous sections to illustrate the process of combining

1664 **policies**. The policy governing read access to medical elements of a record is formed from each of

1665 the four **rules** described in Section 4.2.3. In plain language, the combined rule is:

- 1666 • Either the requestor is the patient; or
- 1667 • the requestor is the parent or guardian and the patient is under 16; or
- 1668 • the requestor is the primary care physician and a notification is sent to the patient; and
- 1669 • the requestor is not an administrator.

1670 The following XACML `<PolicySet>` illustrates the combined **policies**. **Policy 3** is included by

1671 reference and **policy 2** is explicitly included.

```

1672 [01] <?xml version="1.0" encoding="UTF-8"?>
1673 [02] <PolicySet
1674 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1675 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1676 [05]   PolicySetId=
1677 [06]   "urn:oasis:names:tc:xacml:1.0:examples:policysetid:1"
1678 [07]   PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
1679 [071] policy-combining-algorithm:deny-overrides"/>
1680 [08] <Description>
1681 [09]   Example policy set.
1682 [10] </Description>
1683 [11] <Target>
1684 [12]   <Subjects>
1685 [13]     <Subject>
1686 [14]       <!-- any subject -->
1687 [15]       <AnySubject/>
1688 [16]     </Subject>
1689 [17]   </Subjects>
1690 [18]   <Resources>
1691 [19]     <Resource>
1692 [20]       <!-- any resource in the target namespace -->
1693 [21]       <ResourceMatch
1694 [22]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1695 [23]           <AttributeValue
1696 [24]             DataType="http://www.w3.org/2001/XMLSchema#string">
1697 [25]               http://www.medico.com/records.xsd

```

```

1698 [24]         </AttributeValue>
1699 [25]         <ResourceAttributeDesignator AttributeId=
1700 [26]         "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1701         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1702 [27]         </ResourceMatch>
1703 [28]         </Resource>
1704 [29]     </Resources>
1705 [30]     <Actions>
1706 [31]         <Action>
1707 [32]             <!-- any action -->
1708 [33]             <AnyAction/>
1709 [34]         </Action>
1710 [35]     </Actions>
1711 [36] </Target>
1712 [37] <!-- include policy from the example 3 by reference -->
1713 [38] <PolicyIdReference>
1714 [39]     urn:oasis:names:tc:xacml:1.0:examples:policyid:3
1715 [40] </PolicyIdReference>
1716 [41]     <!-- policy 2 combines rules from the examples 1, 2,
1717 [42]     and 4 is included by value. -->
1718 [43] <Policy
1719 [44]     PolicyId="urn:oasis:names:tc:xacml:examples:policyid:2"
1720 [45]     RuleCombiningAlgId=
1721 [46]     "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
1722     overrides">
1723 [47]     <Description>
1724 [48]         Policy for any medical record in the
1725 [49]         http://www.medico.com/schemas/record.xsd namespace
1726 [50]     </Description>
1727 [51]     <Target> ... </Target>
1728 [52]     <Rule
1729 [53]         RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"
1730 [54]         Effect="Permit"> ... </Rule>
1731 [55]     <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
1732 [56]         Effect="Permit"> ... </Rule>
1733 [57]     <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:4"
1734 [58]         Effect="Deny"> ... </Rule>
1735 [59]     <Obligations> ... </Obligations>
1736 [60] </Policy>
1737 [61] </PolicySet>
1738

```

1739 [02]-[07] PolicySet declaration. Standard XML namespace declarations are included as well as
1740 PolicySetId, and **policy combining algorithm** identifier.

1741 [05]-[06] PolicySetId is used for identifying this **policy set** and for possible inclusion of this
1742 **policy set** into another **policy set**.

1743 [07] **Policy combining algorithm** identifier. Policies in the **policy set** are combined according to
1744 the specified **policy combining algorithm** identifier when the **authorization decision** is
1745 computed.

1746 [08]-[10] Free form description of the **policy set**.

1747 [11]-[36] PolicySet Target element defines a set of **decision requests** that are applicable to
1748 this PolicySet.

1749 [38]-[40] PolicyIdReference includes **policy** by id.

1750 [43]-[60] **Policy 2** is explicitly included in this **policy set**.

1751 5. Policy syntax (normative, with the exception of 1752 the schema fragments)

1753 5.1. Element <PolicySet>

1754 The <PolicySet> element is a top-level element in the XACML policy schema. <PolicySet> is
1755 an aggregation of other *policy sets* and *policies*. *Policy sets* MAY be included in an enclosing
1756 <PolicySet> element either directly using the <PolicySet> element or indirectly using the
1757 <PolicySetIdReference> element. *Policies* MAY be included in an enclosing <PolicySet>
1758 element either directly using the <Policy> element or indirectly using the <PolicyIdReference>
1759 element.

1760 If a <PolicySet> element contains references to other *policy sets* or *policies* in the form of
1761 URLs, then these references MAY be resolvable.

1762 *Policies* included in the <PolicySet> element MUST be combined using the algorithm specified
1763 by the PolicyCombiningAlgId attribute. <PolicySet> is treated exactly like a <Policy> in all
1764 the *policy combining algorithms*.

1765 The <Target> element defines the applicability of the <PolicySet> to a set of *decision*
1766 *requests*. If the <Target> element within <PolicySet> matches the *request context*, then the
1767 <PolicySet> element MAY be used by the *PDP* in making its *authorization decision*.

1768 The <Obligations> element contains a set of *obligations* that MUST be fulfilled by the *PEP* in
1769 conjunction with the *authorization decision*. If the *PEP* does not understand any of the
1770 *obligations*, then it MUST act as if the *PDP* had returned a “Deny” *authorization decision* value.

```
1771 <xs:element name="PolicySet" type="xacml:PolicySetType"/>  
1772 <xs:complexType name="PolicySetType">  
1773 <xs:sequence>  
1774 <xs:element ref="xacml:Description" minOccurs="0"/>  
1775 <xs:element ref="xacml:PolicySetDefaults" minOccurs="0"/>  
1776 <xs:element ref="xacml:Target"/>  
1777 <xs:choice minOccurs="0" maxOccurs="unbounded">  
1778 <xs:element ref="xacml:PolicySet"/>  
1779 <xs:element ref="xacml:Policy"/>  
1780 <xs:element ref="xacml:PolicySetIdReference"/>  
1781 <xs:element ref="xacml:PolicyIdReference"/>  
1782 </xs:choice>  
1783 <xs:element ref="xacml:Obligations" minOccurs="0"/>  
1784 </xs:sequence>  
1785 <xs:attribute name="PolicySetId" type="xs:anyURI" use="required"/>  
1786 <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI"  
1787 use="required"/>  
1788 </xs:complexType>
```

1789 The <PolicySet> element is of **PolicySetType** complex type.

1790 The <PolicySet> element contains the following attributes and elements:

1791 PolicySetId [Required]

1792 *Policy set* identifier. It is the responsibility of the *PAP* to ensure that no two *policies*
1793 visible to the *PDP* have the same identifier. This MAY be achieved by following a
1794 predefined URN or URI scheme. If the *policy set* identifier is in the form of a URL, then it
1795 MAY be resolvable.

1796

1797 PolicyCombiningAlgId [Required]

1798 The identifier of the *policy-combining algorithm* by which the <PolicySet>
 1799 components MUST be combined. Standard *policy-combining algorithms* are listed in
 1800 Appendix C. Standard *policy-combining algorithm* identifiers are listed in Section B.10.

1801 <Description> [Optional]

1802 A free-form description of the <PolicySet>.

1803 <PolicySetDefaults> [Optional]

1804 A set of default values applicable to the <PolicySet>. The scope of the
 1805 <PolicySetDefaults> element SHALL be the enclosing *policy set*.

1806 <Target> [Required]

1807 The <Target> element defines the applicability of a <PolicySet> to a set of *decision*
 1808 *requests*.

1809 The <Target> element MAY be declared by the creator of the <PolicySet> or it MAY be
 1810 computed from the <Target> elements of the referenced <Policy> elements, either as
 1811 an intersection or as a union.

1812 <PolicySet> [Any Number]

1813 A *policy set* component that is included in this *policy set*.

1814 <Policy> [Any Number]

1815 A *policy* component that is included in this *policy set*.

1816 <PolicySetIdReference> [Any Number]

1817 A reference to a <PolicySet> component that MUST be included in this *policy set*. If
 1818 <PolicySetIdReference> is a URL, then it MAY be resolvable.

1819 <PolicyIdReference> [Any Number]

1820 A reference to a <Policy> component that MUST be included in this *policy set*. If the
 1821 <PolicyIdReference> is a URL, then it MAY be resolvable.

1822 <Obligations> [Optional]

1823 Contains the set of <Obligation> elements. See Section 7.11 for a description of how
 1824 the set of *obligations* to be returned by the *PDP* shall be determined.

1825 5.2. Element <Description>

1826 The <Description> element is used for a free-form description of the <PolicySet> element,
 1827 <Policy> element and <Rule> element. The <Description> element is of **xs:string** simple
 1828 type.

```
1829 <xs:element name="Description" type="xs:string"/>
```

1830 5.3. Element <PolicySetDefaults>

1831 The <PolicySetDefaults> element SHALL specify default values that apply to the
 1832 <PolicySet> element.

```

1833 <xs:element name="PolicySetDefaults" type="xacml:DefaultsType"/>
1834 <xs:complexType name="DefaultsType">
1835   <xs:sequence>
1836     <xs:choice>
1837       <xs:element ref="xacml:XPathVersion" minOccurs="0"/>
1838     </xs:choice>
1839   </xs:sequence>
1840 </xs:complexType>

```

1841 <PolicySetDefaults> element is of **DefaultsType** complex type.

1842 The <PolicySetDefaults> element contains the following elements:

1843 <XPathVersion> [Optional]

1844 Default XPath version.

1845 **5.4. Element <XPathVersion>**

1846 The <XPathVersion> element SHALL specify the version of the XPath specification to be used by
1847 <AttributeSelector> elements.

```

1848 <xs:element name="XPathVersion" type="xs:anyURI"/>

```

1849 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**
1850 or **policy** contains <AttributeSelector> elements or XPath-based functions.

1852 **5.5. Element <Target>**

1853 The <Target> element identifies the set of **decision requests** that the parent element is intended
1854 to evaluate. The <Target> element SHALL appear as a child of <PolicySet>, <Policy> and
1855 <Rule> elements. It contains definitions for **subjects**, **resources** and **actions**.

1856 The <Target> element SHALL contain a **conjunctive sequence** of <Subjects>, <Resources>
1857 and <Actions> elements. For the parent of the <Target> element to be applicable to the
1858 **decision request**, there MUST be at least one positive match between each section of the
1859 <Target> element and the corresponding section of the <xacml-context:Request> element.

```

1860 <xs:element name="Target" type="xacml:TargetType"/>
1861 <xs:complexType name="TargetType">
1862   <xs:sequence>
1863     <xs:element ref="xacml:Subjects"/>
1864     <xs:element ref="xacml:Resources"/>
1865     <xs:element ref="xacml:Actions"/>
1866   </xs:sequence>
1867 </xs:complexType>

```

1868 The <Target> element is of **TargetType** complex type.

1869 The <Target> element contains the following elements:

1870 <Subjects> [Required]

1871 Matching specification for the **subject attributes** in the **context**.

1872 <Resources> [Required]

1873 Matching specification for the **resource attributes** in the **context**.

1874

1875 <Actions> [Required]

1876 Matching specification for the *action attributes* in the *context*.

1877 5.6. Element <Subjects>

1878 The <Subjects> element SHALL contain a *disjunctive sequence* of <Subject> elements.

```
1879 <xs:element name="Subjects" type="xacml:SubjectsType"/>
1880 <xs:complexType name="SubjectsType">
1881   <xs:choice>
1882     <xs:element ref="xacml:Subject" maxOccurs="unbounded"/>
1883     <xs:element ref="xacml:AnySubject"/>
1884   </xs:choice>
1885 </xs:complexType>
```

1886 The <Subjects> element is of **SubjectsType** complex type.

1887 The <Subjects> element contains the following elements:

1888 <Subject> [One To Many, Required Choice]

1889 See Section 5.7.

1890 <AnySubject> [Required Choice]

1891 See Section 5.8.

1892 5.7. Element <Subject>

1893 The <Subject> element SHALL contain a *conjunctive sequence* of <SubjectMatch> elements.

```
1895 <xs:element name="Subject" type="xacml:SubjectType"/>
1896 <xs:complexType name="SubjectType">
1897   <xs:sequence>
1898     <xs:element ref="xacml:SubjectMatch" maxOccurs="unbounded"/>
1899   </xs:sequence>
1900 </xs:complexType>
```

1901 The <Subject> element is of **SubjectType** complex type.

1902 The <Subject> element contains the following elements:

1903 <SubjectMatch> [One to Many]

1904 A *conjunctive sequence* of individual matches of the *subject attributes* in the *context*
1905 and the embedded *attribute* values.

1906 5.8. Element <AnySubject>

1907 The <AnySubject> element SHALL match any *subject attribute* in the *context*.

```
1908 <xs:element name="AnySubject"/>
```

1909 5.9. Element <SubjectMatch>

1910 The <SubjectMatch> element SHALL identify a set of *subject*-related entities by matching
1911 *attribute* values in a <xacml-context:Subject> element of the *context* with the embedded
1912 *attribute* value.

```

1913 <xs:element name="SubjectMatch" type="xacml:SubjectMatchType"/>
1914 <xs:complexType name="SubjectMatchType">
1915   <xs:sequence>
1916     <xs:element ref="xacml:AttributeValue"/>
1917     <xs:choice>
1918       <xs:element ref="xacml:SubjectAttributeDesignator"/>
1919       <xs:element ref="xacml:AttributeSelector"/>
1920     </xs:choice>
1921   </xs:sequence>
1922   <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
1923 </xs:complexType>

```

1924 The <SubjectMatch> element is of **SubjectMatchType** complex type.

1925 The <SubjectMatch> element contains the following attributes and elements:

1926 MatchId [Required]

1927 Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI** with
1928 legal values documented in Section A.12.

1929 <AttributeValue> [Required]

1930 Embedded **attribute** value.

1931 <SubjectAttributeDesignator> [Required choice]

1932 Identifies one or more **attribute** values in a <Subject> element of the **context**.

1933 <AttributeSelector> [Required choice]

1934 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
1935 expression SHOULD resolve to an **attribute** in a <Subject> element of the **context**.

1936 **5.10. Element <Resources>**

1937 The <Resources> element SHALL contain a **disjunctive sequence** of <Resource> elements.

```

1938 <xs:element name="Resources" type="xacml:ResourcesType"/>
1939 <xs:complexType name="ResourcesType">
1940   <xs:choice>
1941     <xs:element ref="xacml:Resource" maxOccurs="unbounded"/>
1942     <xs:element ref="xacml:AnyResource"/>
1943   </xs:choice>
1944 </xs:complexType>

```

1945 The <Resources> element is of **ResourcesType** complex type.

1946 The <Resources> element contains the following elements:

1947 <Resource> [One To Many, Required Choice]

1948 See Section 5.11.

1949 <AnyResource> [Required Choice]

1950 See Section 5.12.

1951 **5.11. Element <Resource>**

1952 The <Resource> element SHALL contain a **conjunctive sequence** of <ResourceMatch>
1953 elements.

```

1954 <xs:element name="Resource" type="xacml:ResourceType"/>
1955 <xs:complexType name="ResourceType">
1956   <xs:sequence>
1957     <xs:element ref="xacml:ResourceMatch" maxOccurs="unbounded"/>
1958   </xs:sequence>
1959 </xs:complexType>

```

1960 The <Resource> element is of **ResourceType** complex type.

1961 The <Resource> element contains the following elements:

1962 <ResourceMatch> [One to Many]

1963 A **conjunctive sequence** of individual matches of the **resource attributes** in the **context**
1964 and the embedded **attribute** values.

1965 5.12. Element <AnyResource>

1966 The <AnyResource> element SHALL match any **resource attribute** in the **context**.

```

1967 <xs:element name="AnyResource"/>

```

1968 5.13. Element <ResourceMatch>

1969 The <ResourceMatch> element SHALL identify a set of **resource**-related entities by matching
1970 **attribute** values in the <xacml-context:Resource> element of the **context** with the embedded
1971 **attribute** value.

```

1972 <xs:element name="ResourceMatch" type="xacml:ResourceMatchType"/>
1973 <xs:complexType name="ResourceMatchType">
1974   <xs:sequence>
1975     <xs:element ref="xacml:AttributeValue"/>
1976     <xs:choice>
1977       <xs:element ref="xacml:ResourceAttributeDesignator"/>
1978       <xs:element ref="xacml:AttributeSelector"/>
1979     </xs:choice>
1980   </xs:sequence>
1981   <xs:attribute name="MatchId" type="xs:anyMatch" use="required"/>
1982 </xs:complexType>

```

1983 The <ResourceMatch> element is of **ResourceMatchType** complex type.

1984 The <ResourceMatch> element contains the following attributes and elements:

1985 MatchId [Required]

1986 Specifies a matching function. Values of this attribute MUST be of type **xs:anyURI**, with
1987 legal values documented in Section A.12.

1988 <AttributeValue> [Required]

1989 Embedded **attribute** value.

1990 <ResourceAttributeDesignator> [Required Choice]

1991 Identifies one or more **attribute** values in the <Resource> element of the **context**.

1992 <AttributeSelector> [Required Choice]

1993 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
1994 expression SHOULD resolve to an **attribute** in the <Resource> element of the **context**.

1995 **5.14. Element <Actions>**

1996 The <Actions> element SHALL contain a **disjunctive sequence** of <Action> elements.

```
1997 <xs:element name="Actions" type="xacml:ActionTypes"/>
1998 <xs:complexType name="ActionTypes">
1999   <xs:choice>
2000     <xs:element ref="xacml:Action" maxOccurs="unbounded"/>
2001     <xs:element ref="xacml:AnyAction"/>
2002   </xs:choice>
2003 </xs:complexType>
```

2004 The <Actions> element is of **ActionTypes** complex type.

2005 The <Actions> element contains the following elements:

2006 <Action> [One To Many, Required Choice]

2007 See Section 5.15.

2008 <AnyAction> [Required Choice]

2009 See Section 5.16.

2010 **5.15. Element <Action>**

2011 The <Action> element SHALL contain a **conjunctive sequence** of <ActionMatch> elements.

```
2012 <xs:element name="Action" type="xacml:ActionType"/>
2013 <xs:complexType name="ActionType">
2014   <xs:sequence>
2015     <xs:element ref="xacml:ActionMatch" maxOccurs="unbounded"/>
2016   </xs:sequence>
2017 </xs:complexType>
```

2018 The <Action> element is of **ActionType** complex type.

2019 The <Action> element contains the following elements:

2020 <ActionMatch> [One to Many]

2021 A **conjunctive sequence** of individual matches of the **action** attributes in the **context** and
2022 the embedded **attribute** values.

2023 **5.16. Element <AnyAction>**

2024 The <AnyAction> element SHALL match any **action attribute** in the **context**.

```
2025 <xs:element name="AnyAction"/>
```

2027 **5.17. Element <ActionMatch>**

2028 The <ActionMatch> element SHALL identify a set of **action**-related entities by matching **attribute**
2029 values in the <xacml-context:Action> element of the **context** with the embedded **attribute**
2030 value.

```
2031 <xs:element name="ActionMatch" type="xacml:ActionMatchType"/>
2032 <xs:complexType name="ActionMatchType">
2033   <xs:sequence>
2034     <xs:element ref="xacml:AttributeValue"/>
```

```

2035     <xs:choice>
2036         <xs:element ref="xacml:ActionAttributeDesignator"/>
2037         <xs:element ref="xacml:AttributeSelector"/>
2038     </xs:choice>
2039 </xs:sequence>
2040 <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
2041 </xs:complexType>

```

2042 The <ActionMatch> element is of **ActionMatchType** complex type.

2043 The <ActionMatch> element contains the following attributes and elements:

2044 MatchId [Required]

2045 Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI**, with
 2046 legal values documented in Section A.12.

2047 <AttributeValue> [Required]

2048 Embedded **attribute** value.

2049 <ActionAttributeDesignator> [Required Choice]

2050 Identifies one or more **attribute** values in the <Action> element of the **context**.

2051 <AttributeSelector> [Required Choice]

2052 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
 2053 expression SHOULD resolve to an **attribute** in the <Action> element of the **context**.

2054 **5.18. Element <PolicySetIdReference>**

2055 The <PolicySetIdReference> element SHALL be used to reference a <PolicySet> element
 2056 by id. If <PolicySetIdReference> is a URL, then it MAY be resolvable to the <PolicySet>. The
 2057 mechanism for resolving a **policy set** reference to the corresponding **policy set** is outside the
 2058 scope of this specification.

```

2059 <xs:element name="PolicySetIdReference" type="xs:anyURI"/>

```

2060 Element <PolicySetIdReference> is of **xs:anyURI** simple type.

2061 **5.19. Element <PolicyIdReference>**

2062 The <xacml:PolicyIdReference> element SHALL be used to reference a <Policy> element
 2063 by id. If <PolicyIdReference> is a URL, then it MAY be resolvable to the <Policy>. The
 2064 mechanism for resolving a **policy** reference to the corresponding **policy** is outside the scope of this
 2065 specification.

```

2066 <xs:element name="PolicyIdReference" type="xs:anyURI"/>

```

2067 Element <PolicyIdReference> is of **xs:anyURI** simple type.

2068 **5.20. Element <Policy>**

2069 The <Policy> element is the smallest entity that SHALL be presented to the **PDP** for evaluation.

2070 The main components of this element are the <Target>, <Rule> and <Obligations> elements
 2071 and the RuleCombiningAlgId attribute.

2072 The <Target> element SHALL define the applicability of the <Policy> to a set of **decision**
2073 **requests**.

2074 **Rules** included in the <Policy> element MUST be combined by the algorithm specified by the
2075 RuleCombiningAlgId attribute.

2076 The <Obligations> element SHALL contain a set of **obligations** that MUST be fulfilled by the
2077 **PDP** in conjunction with the **authorization decision**.

```
2078 <xs:element name="Policy" type="xacml:PolicyType"/>
2079 <xs:complexType name="PolicyType">
2080 <xs:sequence>
2081 <xs:element ref="xacml:Description" minOccurs="0"/>
2082 <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>
2083 <xs:element ref="xacml:Target"/>
2084 <xs:element ref="xacml:Rule" minOccurs="0" maxOccurs="unbounded"/>
2085 <xs:element ref="xacml:Obligations" minOccurs="0"/>
2086 </xs:sequence>
2087 <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
2088 <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>
2089 </xs:complexType>
```

2090 The <Policy> element is of **PolicyType** complex type.

2091 The <Policy> element contains the following attributes and elements:

2092 PolicyId [Required]

2093 **Policy** identifier. It is the responsibility of the **PAP** to ensure that no two **policies** visible to
2094 the **PDP** have the same identifier. This MAY be achieved by following a predefined URN or
2095 URI scheme. If the **policy** identifier is in the form of a URL, then it MAY be resolvable.

2096 RuleCombiningAlgId [Required]

2097 The identifier of the rule-combining algorithm by which the <Policy> components MUST
2098 be combined. Standard rule-combining algorithms are listed in Appendix C. Standard rule-
2099 combining algorithm identifiers are listed in Section B.10.

2100 <Description> [Optional]

2101 A free-form description of the **policy**. See Section 5.2 Element <Description>.

2102 <PolicyDefaults> [Optional]

2103 Defines a set of default values applicable to the **policy**. The scope of the
2104 <PolicyDefaults> element SHALL be the enclosing policy.

2105 <Target> [Required]

2106 The <Target> element SHALL define the applicability of a <Policy> to a set of **decision**
2107 **requests**.

2108 The <Target> element MAY be declared by the creator of the <Policy> element, or it
2109 MAY be computed from the <Target> elements of the referenced <Rule> elements either
2110 as an intersection or as a union.

2111 <Rule> [Any Number]

2112 A sequence of authorizations that MUST be combined according to the
2113 RuleCombiningAlgId attribute. **Rules** whose <Target> elements match the **decision**
2114 **request** MUST be considered. **Rules** whose <Target> elements do not match the
2115 **decision request** SHALL be ignored.

2116 <Obligations> [Optional]

2117 A **conjunctive sequence** of **obligations** that MUST be fulfilled by the **PEP** in conjunction
2118 with the **authorization decision**. See Section 7.11 for a description of how the set of
2119 **obligations** to be returned by the **PDP** SHALL be determined.

2120 **5.21. Element <PolicyDefaults>**

2121 The <PolicyDefaults> element SHALL specify default values that apply to the <Policy>
2122 element.

```
2123       <xs:element name="PolicyDefaults" type="xacml:DefaultsType"/>  
2124       <xs:complexType name="DefaultsType">  
2125         <xs:sequence>  
2126         <xs:choice>  
2127         <xs:element ref="xacml:XPathVersion" minOccurs="0"/>  
2128         </xs:choice>  
2129         </xs:sequence>  
2130       </xs:complexType>
```

2131 <PolicyDefaults> element is of **DefaultsType** complex type.

2132 The <PolicyDefaults> element contains the following elements:

2133 <XPathVersion> [Optional]

2134 Default XPath version.

2135 **5.22. Element <Rule>**

2136 The <Rule> element SHALL define the individual **rules** in the **policy**. The main components of
2137 this element are the <Target> and <Condition> elements and the **Effect** attribute.

```
2138       <xs:element name="Rule" type="xacml:RuleType"/>  
2139       <xs:complexType name="RuleType">  
2140         <xs:sequence>  
2141         <xs:element ref="xacml:Description" minOccurs="0"/>  
2142         <xs:element ref="xacml:Target" minOccurs="0"/>  
2143         <xs:element ref="xacml:Condition" minOccurs="0"/>  
2144         </xs:sequence>  
2145         <xs:attribute name="RuleId" type="xs:anyURI" use="required"/>  
2146         <xs:attribute name="Effect" type="xacml:EffectType" use="required"/>  
2147       </xs:complexType>
```

2148 The <Rule> element is of **RuleType** complex type.

2149 The <Rule> element contains the following attributes and elements:

2150 RuleId [Required]

2151 A URN identifying this **rule**.

2152 Effect [Required]

2153 **Rule effect.** Values of this attribute are either "Permit" or "Deny".

2154 <Description> [Optional]

2155 A free-form description of the **rule**.

2156

2157 <Target> [Optional]
2158 Identifies the set of **decision requests** that the <Rule> element is intended to evaluate. If
2159 this element is omitted, then the **target** for the <Rule> SHALL be defined by the
2160 <Target> element of the enclosing <Policy> element. See Section 5.5 for details.

2161 <Condition> [Optional]
2162 A **predicate** that MUST be satisfied for the **rule** to be assigned its `Effect` value. A
2163 **condition** is a boolean function over a combination of **subject**, **resource**, **action** and
2164 **environment attributes** or other functions.

2165 5.23. Simple type EffectType

2166 The **EffectType** simple type defines the values allowed for the `Effect` attribute of the <Rule>
2167 element and for the `FulfillOn` attribute of the <Obligation> element.

```
2168 <xs:simpleType name="EffectType">  
2169 <xs:restriction base="xs:string">  
2170 <xs:enumeration value="Permit"/>  
2171 <xs:enumeration value="Deny"/>  
2172 </xs:restriction>  
2173 </xs:simpleType>
```

2174 5.24. Element <Condition>

2175 The <Condition> element is a boolean function over **subject**, **resource**, **action** and
2176 **environment attributes** or functions of **attributes**. If the <Condition> element evaluates to
2177 "True", then the enclosing <Rule> element is assigned its `Effect` value.

```
2178 <xs:element name="Condition" type="xacml:ApplyType"/>
```

2179 The <Condition> element is of **ApplyType** complex type.

2180 5.25. Element <Apply>

2181 The <Apply> element denotes application of a function to its arguments, thus encoding a function
2182 call. The <Apply> element can be applied to any combination of <Apply>,
2183 <AttributeValue>,<SubjectAttributeDesignator>,
2184 <ResourceAttributeDesignator>,<ActionAttributeDesignator>,
2185 <EnvironmentAttributeDesignator> and <AttributeSelector> arguments.

```
2186 <xs:element name="Apply" type="xacml:ApplyType"/>  
2187 <xs:complexType name="ApplyType">  
2188 <xs:choice minOccurs="0" maxOccurs="unbounded">  
2189 <xs:element ref="xacml:Function"/>  
2190 <xs:element ref="xacml:Apply"/>  
2191 <xs:element ref="xacml:AttributeValue"/>  
2192 <xs:element ref="xacml:SubjectAttributeDesignator"/>  
2193 <xs:element ref="xacml:ResourceAttributeDesignator"/>  
2194 <xs:element ref="xacml:ActionAttributeDesignator"/>  
2195 <xs:element ref="xacml:EnvironmentAttributeDesignator"/>  
2196 <xs:element ref="xacml:AttributeSelector"/>  
2197 </xs:choice>  
2198 <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>  
2199 </xs:complexType>
```

2200 The <Apply> element is of **ApplyType** complex type.

2201 The <Apply> element contains the following attributes and elements:

- 2202 `FunctionId` [Required]
- 2203 The URN of a function. XACML-defined functions are described in Appendix A.
- 2204 `<Function>` [Optional]
- 2205 The name of a function that is applied to the elements of a **bag**. See Section A14.11.
- 2206 `<Apply>` [Optional]
- 2207 A nested function-call argument.
- 2208 `<AttributeValue>` [Optional]
- 2209 A literal value argument.
- 2210 `<SubjectAttributeDesignator>` [Optional]
- 2211 A **subject attribute** argument.
- 2212 `<ResourceAttributeDesignator>` [Optional]
- 2213 A **resource attribute** argument.
- 2214 `<ActionAttributeDesignator>` [Optional]
- 2215 An **action attribute** argument.
- 2216 `<EnvironmentAttributeDesignator>` [Optional]
- 2217 An **environment attribute** argument.
- 2218 `<AttributeSelector>` [Optional]
- 2219 An **attribute** selector argument.

2220 **5.26. Element <Function>**

- 2221 The `Function` element SHALL be used to name a function that is applied by the higher-order **bag**
- 2222 functions to every element of a **bag**. The higher-order **bag** functions are described in Section
- 2223 A14.11.

```
2224       <xs:element name="Function" type="xacml:FunctionType"/>
2225       <xs:complexType name="FunctionType">
2226        <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>
2227       </xs:complexType>
```

- 2228 The `Function` element is of **FunctionType** complex type.

- 2229 The `Function` element contains the following attributes:

- 2230 `FunctionId` [Required]

- 2231 The identifier for the function that is applied to the elements of a **bag** by the higher-order **bag**
- 2232 functions.

2233 **5.27. Complex type AttributeDesignatorType**

- 2234 The **AttributeDesignatorType** complex type is the type for elements and extensions that identify
- 2235 **attributes**. An element of this type contains properties by which it MAY be matched to **attributes**
- 2236 in the request **context**.

2237 In addition, elements of this type MAY control behaviour in the event that no matching **attribute** is
2238 present in the **context**.

2239 Elements of this type SHALL NOT alter the match semantics of named **attributes**, but MAY narrow
2240 the search space.

```
2241 <xs:complexType name="AttributeDesignatorType">  
2242   <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2243   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>  
2244   <xs:attribute name="Issuer" type="xs:string" use="optional"/>  
2245   <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"  
2246   default="false"/>  
2247 </xs:complexType>
```

2248 A named **attribute** SHALL match an **attribute** if the values of their respective `AttributeId`,
2249 `DataType` and `Issuer` attributes match. The **attribute** designator's `AttributeId` MUST match,
2250 by URI equality, the `AttributeId` of the **attribute**. The **attribute** designator's `DataType` MUST
2251 match, by URI equality, the `DataType` of the same **attribute**.

2252 If the `Issuer` attribute is present in the **attribute** designator, then it MUST match, by string
2253 equality, the `Issuer` of the same **attribute**. If the `Issuer` is not present in the **attribute**
2254 designator, then the matching of the **attribute** to the named **attribute** SHALL be governed by
2255 `AttributeId` and `DataType` attributes alone.

2256 The `<AttributeDesignatorType>` contains the following attributes:

2257 `AttributeId` [Required]

2258 This attribute SHALL specify the `AttributeId` with which to match the **attribute**.

2259 `DataType` [Required]

2260 This attribute SHALL specify the data-type with which to match the **attribute**.

2261 `Issuer` [Optional]

2262 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **attribute**.

2263 `MustBePresent` [Optional]

2264 This attribute governs whether the element returns "Indeterminate" in the case where the
2265 named **attribute** is absent. If the *named attribute* is absent and `MustBePresent` is "True",
2266 then this element SHALL result in "Indeterminate". The default value SHALL be "False".

2267 **5.28. Element <SubjectAttributeDesignator>**

2268 The `<SubjectAttributeDesignator>` element is of the **SubjectAttributeDesignatorType**.
2269 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType**
2270 complex type. It is the base type for elements and extensions that refer to *named categorized*
2271 **subject attributes**. A *named categorized subject attribute* is defined as follows:

2272 A **subject** is represented by a `<Subject>` element in the `<xacml-context:Request>` element.
2273 Each `<Subject>` element SHALL contain the XML attribute `SubjectCategory`. This attribute is
2274 called the *subject category attribute*.

2275 A *categorized subject* is a **subject** that is identified by a particular *subject category attribute*.

2276 A **subject attribute** is an **attribute** of a particular **subject**, i.e. contained within a `<Subject>`
2277 element.

2278 A named **subject attribute** is a named **attribute** for a **subject**.

2279 A named categorized **subject attribute** is a named **subject attribute** for a particular **categorized**
2280 **subject**.

2281 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType** with a
2282 **SubjectCategory** attribute. The **SubjectAttributeDesignatorType** extends the match
2283 semantics of the **AttributeDesignatorType** such that it narrows the **attribute** search space to the
2284 specific **categorized subject** such that the value of this element's **SubjectCategory** attribute
2285 matches, by URI equality, the value of the <Request> element's **subject category attribute**.

2286 If there are multiple **subjects** with the same **SubjectCategory** xml attribute, then they SHALL be
2287 treated as if they were one **categorized subject**.

2288 Elements and extensions of the **SubjectAttributeDesignatorType** complex type determine the
2289 presence of select **attribute values** associated with **named categorized subject attributes**.
2290 Elements and extensions of the **SubjectAttributeDesignatorType** SHALL NOT alter the match
2291 semantics of **named categorized subject attributes**, but MAY narrow the search space.

```

2292 <xs:complexType name="SubjectAttributeDesignatorType">
2293   <xs:complexContent>
2294     <xs:extension base="xacml:AttributeDesignatorType">
2295       <xs:attribute name="SubjectCategory"
2296         type="xs:anyURI"
2297         use="optional"
2298         default="
2299           urn:oasis:names:tc:xacml:1.0:subject-category:access-
2300 subject"/>
2301     </xs:extension>
2302   </xs:complexContent>
2303 </xs:complexType>

```

2304 The <SubjectAttributeDesignatorType> complex type contains the following attribute in
2305 addition to the attributes of the **AttributeDesignatorType** complex type:

2306 **SubjectCategory** [Optional]

2307 This attribute SHALL specify the **categorized subject** from which to match **named subject**
2308 **attributes**. If **SubjectCategory** is not present, then its default value of
2309 "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" SHALL be
2310 used.

2311 5.29. Element <ResourceAttributeDesignator>

2312 The <ResourceAttributeDesignator> element retrieves a **bag** of values for a **named**
2313 **resource attribute**. A **resource attribute** is an **attribute** contained within the <Resource>
2314 element of the <xacml-context:Request> element. A **named resource attribute** is a **named**
2315 **attribute** that matches a **resource attribute**. A **named resource attribute** SHALL be considered
2316 **present** if there is at least one **resource attribute** that matches the criteria set out below. A
2317 **resource attribute** value is an **attribute** value that is contained within a **resource attribute**.

2318 The <ResourceAttributeDesignator> element SHALL return a **bag** containing all the
2319 **resource attribute** values that are matched by the **named resource attribute**. The
2320 **MustBePresent** attribute governs whether this element returns an empty **bag** or "Indeterminate"
2321 in the case that the **named resource attribute** is absent. If the **named resource attribute** is not
2322 present and the **MustBePresent** attribute is "False" (its default value), then this element SHALL
2323 evaluate to an empty **bag**. If the **named resource attribute** is not present and the
2324 **MustBePresent** attribute is "True", then this element SHALL evaluate to "Indeterminate".
2325 Regardless of the **MustBePresent** attribute, if it cannot be determined whether the **named**

2326 *resource attribute* is present or not in the **request context**, or the value of the *named resource*
2327 **attribute** is unavailable, then the expression SHALL evaluate to “Indeterminate”.

2328 A *named resource attribute* SHALL match a **resource attribute** as per the match semantics
2329 specified in the **AttributeDesignatorType** complex type [Section 5.27]

2330 The <ResourceAttributeDesignator> MAY appear in the <ResourceMatch> element and
2331 MAY be passed to the <Apply> element as an argument.

```
2332 <xs:element name="ResourceAttributeDesignator"  
2333           type="xacml:AttributeDesignatorType"/>
```

2334 The <ResourceAttributeDesignator> element is of the **AttributeDesignatorType**
2335 complex type.

2336 **5.30. Element <ActionAttributeDesignator>**

2337 The <ActionAttributeDesignator> element retrieves a **bag** of values for a *named action*
2338 **attribute**. An *action attribute* is an **attribute** contained within the <Action> element of the
2339 <xacml-context:Request> element. A *named action attribute* has specific criteria (described
2340 below) with which to match an *action attribute*. A *named action attribute* SHALL be considered
2341 *present*, if there is at least one *action attribute* that matches the criteria. An *action attribute value*
2342 is an **attribute value** that is contained within an *action attribute*.

2343 The <ActionAttributeDesignator> element SHALL return a **bag** of all the *action attribute*
2344 values that are matched by the *named action attribute*. The `MustBePresent` attribute governs
2345 whether this element returns an empty **bag** or “Indeterminate” in the case that the *named action*
2346 **attribute** is absent. If the *named action attribute* is not present and the `MustBePresent` attribute
2347 is “False” (its default value), then this element SHALL evaluate to an empty **bag**. If the *named*
2348 **action attribute** is not present and the `MustBePresent` attribute is “True”, then this element
2349 SHALL evaluate to “Indeterminate”. Regardless of the `MustBePresent` attribute, if it cannot be
2350 determined whether the *named action attribute* is present or not present in the request **context**, or
2351 the value of the *named action attribute* is unavailable, then the expression SHALL evaluate to
2352 “Indeterminate”.

2353 A *named action attribute* SHALL match an *action attribute* as per the match semantics specified
2354 in the **AttributeDesignatorType** complex type [Section 5.27].

2355 The <ActionAttributeDesignator> MAY appear in the <ActionMatch> element and MAY
2356 be passed to the <Apply> element as an argument.

```
2357 <xs:element name="ActionAttributeDesignator"  
2358           type="xacml:AttributeDesignatorType"/>
```

2359 The <ActionAttributeDesignator> element is of the **AttributeDesignatorType** complex
2360 type.

2361 **5.31. Element <EnvironmentAttributeDesignator>**

2362 The <EnvironmentAttributeDesignator> element retrieves a **bag** of values for a *named*
2363 **environment attribute**. An *environment attribute* is an **attribute** contained within the
2364 <Environment> element of the <xacml-context:Request> element. A *named environment*
2365 **attribute** has specific criteria (described below) with which to match an *environment attribute*. A
2366 *named environment attribute* SHALL be considered *present*, if there is at least one *environment*
2367 **attribute** that matches the criteria. An *environment attribute value* is an **attribute value** that is
2368 contained within an *environment attribute*.

2369 The <EnvironmentAttributeDesignator> element SHALL evaluate to a **bag** of all the
2370 **environment attribute** values that are matched by the *named environment attribute*. The
2371 `MustBePresent` attribute governs whether this element returns an empty **bag** or “Indeterminate”
2372 in the case that the *named environment attribute* is absent. If the *named environment attribute*
2373 is not present and the `MustBePresent` attribute is “False” (its default value), then this element
2374 SHALL evaluate to an empty **bag**. If the *named environment attribute* is not present and the
2375 `MustBePresent` attribute is “True”, then this element SHALL evaluate to “Indeterminate”.
2376 Regardless of the `MustBePresent` attribute, if it cannot be determined whether the *named*
2377 **environment attribute** is present or not present in the request **context**, or the value of the *named*
2378 **environment attribute** is unavailable, then the expression SHALL evaluate to “Indeterminate”.

2379 A *named environment attribute* SHALL match an **environment attribute** as per the match
2380 semantics specified in the **AttributeDesignatorType** complex type [Section 5.27].

2381 The <EnvironmentAttributeDesignator> MAY be passed to the <Apply> element as an
2382 argument.

```
2383 <xs:element name="EnvironmentAttributeDesignator"  
2384           type="xacml:AttributeDesignatorType"/>
```

2385 The <EnvironmentAttributeDesignator> element is of the **AttributeDesignatorType**
2386 complex type.

2387 **5.32. Element <AttributeSelector>**

2388 The `AttributeSelector` element's `RequestContextPath` XML attribute SHALL contain a
2389 legal XPath expression whose context node is the <xacml-context:Request> element. The
2390 `AttributeSelector` element SHALL evaluate to a **bag** of values whose data-type is specified by
2391 the element's `DataType` attribute. If the `DataType` specified in the `AttributeSelector` is a
2392 primitive data type defined in [XF] or [XS], then the value returned by the XPath expression SHALL
2393 be converted to the `DataType` specified in the `AttributeSelector` using the constructor
2394 function below [XF Section 4] that corresponds to the `DataType`. If an error results from using the
2395 constructor function, then the value of the `AttributeSelector` SHALL be "Indeterminate".

```
2396  
2397     xs:string()  
2398     xs:boolean()  
2399     xs:integer()  
2400     xs:double()  
2401     xs:dateTime()  
2402     xs:date()  
2403     xs:time()  
2404     xs:hexBinary()  
2405     xs:base64Binary()  
2406     xs:anyURI()  
2407     xf:yearMonthDuration()  
2408     xf:dayTimeDuration()  
2409
```

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

2415
2416 Each selected node by the specified XPath expression MUST be either a text node, an attribute
2417 node, a processing instruction node, or a comment node. The string representation of the value of
2418 each selected node MUST be converted to an **attribute** value of the specified data type, and the

2419 result of the `AttributeSelector` is the **bag** of the **attribute** values generated from all the
2420 selected nodes.

2421

2422 If the selected node is different from the node types listed above (a text node, an attribute node, a
2423 processing instruction node, or a comment node), then the result of that **policy** SHALL be

2424 "Indeterminate" with a `Status Code` value of

2425 "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

2426 Support for the `<AttributeSelector>` element is OPTIONAL.

2427

```
2428 <xs:element name="AttributeSelector" type="xacml:AttributeSelectorType"/>
2429 <xs:complexType name="AttributeSelectorType">
2430   <xs:attribute name="RequestContextPath" type="xs:string" use="required"/>
2431   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
2432   <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"
2433   default="false"
2434 </xs:complexType>
```

2435 The `<AttributeSelector>` element is of **AttributeSelectorType** complex type.

2436 The `<AttributeSelector>` element has the following attributes:

2437 `RequestContextPath` [Required]

2438 An XPath expression whose context node is the `<xacml-context:Request>` element.

2439 There SHALL be no restriction on the XPath syntax.

2440 `DataType` [Required]

2441 The bag of values returned by the `AttributeSelector` SHALL be of this data type.

2442 `MustBePresent` [Optional]

2443 Whether or not the designated **attribute** must be present in the **context**. If the XPath expression
2444 selects no node, and the `MustBePresent` attribute is TRUE, then the result SHALL be
2445 "Indeterminate" and the status code SHALL be

2446 "urn:oasis:names:tc:xacml:1.0:status:missing-attribute". If the XPath

2447 expression selects no node, and the `MustBePresent` attribute is missing or FALSE, then the
2448 result SHALL be an empty **bag**. If the XPath expression selects at least one node and the

2449 selected node(s) could be successfully converted to a **bag** of values of the specified data-type,
2450 then the result SHALL be the **bag**, regardless of the value of the `MustBePresent` attribute. If

2451 the XPath expression selects at least one node, but there is an error in converting one or more
2452 of the nodes to values of the specified data-type, then the result SHALL be "Indeterminate" and

2453 the status code SHALL be "urn:oasis:names:tc:xacml:1.0:status:processing-
2454 error", regardless of the value of the `MustBePresent` attribute.

2455 **5.33. Element `<AttributeValue>`**

2456 The `<AttributeValue>` element SHALL contain a literal **attribute** value.

```
2457 <xs:element name="AttributeValue" type="xacml:AttributeValueType"/>
2458 <xs:complexType name="AttributeValueType" mixed="true">
2459   <xs:sequence>
2460     <xs:any namespace="##any" processContents="lax" minOccurs="0"
2461     maxOccurs="unbounded"/>
2462   </xs:sequence>
2463   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
2464   <xs:anyAttribute namespace="##any" processContents="lax"/>
2465 </xs:complexType>
```

2466 The <AttributeValue> element is of **AttributeValueType** complex type.

2467 The <AttributeValue> element has the following attributes:

2468 DataType [Required]

2469 The data-type of the *attribute* value.

2470 **5.34. Element <Obligations>**

2471 The <Obligations> element SHALL contain a set of <Obligation> elements.

2472 Support for the <Obligations> element is OPTIONAL.

```
2473 <xs:element name="Obligations" type="xacml:ObligationsType"/>
2474 <xs:complexType name="ObligationsType">
2475   <xs:sequence>
2476     <xs:element ref="xacml:Obligation" maxOccurs="unbounded"/>
2477   </xs:sequence>
2478 </xs:complexType>
```

2479 The <Obligations> element is of **ObligationsType** complexType.

2480 The <Obligations> element contains the following element:

2481 <Obligation> [One to Many]

2482 A sequence of *obligations*

2483 **5.35. Element <Obligation>**

2484 The <Obligation> element SHALL contain an identifier for the *obligation* and a set of *attributes* that form arguments of the action defined by the *obligation*. The FulfillOn attribute SHALL indicate the *effect* for which this *obligation* applies.

```
2487 <xs:element name="Obligation" type="xacml:ObligationType"/>
2488 <xs:complexType name="ObligationType">
2489   <xs:sequence>
2490     <xs:element ref="xacml:AttributeAssignment" maxOccurs="unbounded"/>
2491   </xs:sequence>
2492   <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
2493   <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required"/>
2494 </xs:complexType>
```

2495 The <Obligation> element is of **ObligationType** complexType. See Section 7.11 for a description of how the set of *obligations* to be returned by the PDP is determined.

2497 The <Obligation> element contains the following elements and attributes:

2498 ObligationId [Required]

2499 **Obligation** identifier. The value of the *obligation* identifier SHALL be interpreted by the *PEP*.

2501 FulfillOn [Required]

2502 The *effect* for which this *obligation* applies.

2503 <AttributeAssignment> [One To Many]

2504 **Obligation** arguments assignment. The values of the *obligation* arguments SHALL be interpreted by the *PEP*.

2506

5.36. Element <AttributeAssignment>

2507 The <AttributeAssignment> element SHALL contain an `AttributeId` and the corresponding
2508 **attribute** value. The `AttributeId` is part of **attribute** meta-data, and is used when the **attribute**
2509 cannot be referenced by its location in the `<xacml-context:Request>`. This situation may arise
2510 in an <Obligation> element if the **obligation** includes parameters. The
2511 <AttributeAssignment> element MAY be used in any way consistent with the schema syntax,
2512 which is a sequence of “any”. The value specified SHALL be understood by the PEP, but it is not
2513 further specified by XACML. See section 7,11 “Obligations”.

```
2514 <xs:element name="AttributeAssignment"  
2515 type="xacml:AttributeAssignmentType"/>  
2516 <xs:complexType name="AttributeAssignmentType" mixed="true">  
2517 <xs:complexContent>  
2518 <xs:extension base="xacml:AttributeValueType">  
2519 <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2520 </xs:extension>  
2521 </xs:complexContent>  
2522 </xs:complexType>
```

2523 The <AttributeAssignment> element is of **AttributeAssignmentType** complex type.

2524 The <AttributeAssignment> element contains the following attributes:

2525 `AttributeId` [Required]

2526 The **attribute** Identifier

2527 6. Context syntax (normative with the exception of 2528 the schema fragments)

2529 6.1. Element <Request>

2530 The <Request> element is a top-level element in the XACML **context** schema. The <Request>
2531 element is an abstraction layer used by the **policy** language. Any proprietary system using the
2532 XACML specification MUST transform its **decision request** into the form of an XACML **context**
2533 <Request>.

2534 The <Request> element contains <Subject>, <Resource>, <Action> and <Environment>
2535 elements. There may be multiple <Subject> elements. Each child element contains a sequence
2536 of <xacml-context:Attribute> elements associated with the **subject**, **resource**, **action** and
2537 **environment** respectively.

```
2538 <xs:element name="Request" type="xacml-context:RequestType"/>  
2539 <xs:complexType name="RequestType">  
2540 <xs:sequence>  
2541 <xs:element ref="xacml-context:Subject" maxOccurs="unbounded"/>  
2542 <xs:element ref="xacml-context:Resource"/>  
2543 <xs:element ref="xacml-context:Action"/>  
2544 <xs:element ref="xacml-context:Environment" minOccurs="0"/>  
2545 </xs:sequence>  
2546 </xs:complexType>
```

2547 The <Request> element is of **RequestType** complex type.

2548 The <Request> element contains the following elements:

- 2549 <Subject> [One to Many]
- 2550 Specifies information about a **subject** of the request **context** by listing a sequence of
 2551 <Attribute> elements associated with the **subject**. One or more <Subject> elements
 2552 are allowed. A **subject** is an entity associated with the **access** request. One **subject**
 2553 might represent the human user that initiated the application from which the request was
 2554 issued. Another **subject** might represent the application's executable code that created the
 2555 request. Another **subject** might represent the machine on which the application was
 2556 executing. Another **subject** might represent the entity that is to be the recipient of the
 2557 **resource**. Attributes of each of these entities MUST be enclosed in a separate
 2558 <Subject> element.
- 2559 <Resource> [Required]
- 2560 Specifies information about the resource for which access is being requested by listing a
 2561 sequence of <Attribute> elements associated with the resource. It MAY include a
 2562 <ResourceContent> element.
- 2563 <Action> [Required]
- 2564 Specifies the requested **action** to be performed on the **resource** by listing a set of
 2565 <Attribute> elements associated with the **action**.
- 2566 <Environment> [Optional]
- 2567 Contains a set of <Attribute> elements of the **environment**. These <Attribute>
 2568 elements MAY form a part of **policy** evaluation.

2569 6.2. Element <Subject>

2570 The <Subject> element specifies a **subject** by listing a sequence of <Attribute> elements
 2571 associated with the **subject**.

```

2572 <xs:element name="Subject" type="xacml-context:SubjectType"/>
2573 <xs:complexType name="SubjectType">
2574   <xs:sequence>
2575     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2576 maxOccurs="unbounded"/>
2577   </xs:sequence>
2578   <xs:attribute name="SubjectCategory" type="xs:anyURI" use="optional"
2579 default="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"/>
2580 </xs:complexType>

```

2581 The <Subject> element is of **SubjectType** complex type.

2582 The <Subject> element contains the following elements:

2583 SubjectCategory [Optional]

2584 This attribute indicates the role that the parent <Subject> played in the formation of the
 2585 access request. If this attribute is not present in a given <Subject> element, then the
 2586 default value of "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" SHALL be
 2587 used, indicating that the parent <Subject> element represents the entity ultimately
 2588 responsible for initiating the **access** request.

2589 If more than one <Subject> element contains a "urn:oasis:names:tc:xacml:1.0:subject-
 2590 category" attribute with the same value, then the PDP SHALL treat the contents of those
 2591 elements as if they were contained in the same <Subject> element.

2592 <Attribute> [Any Number]

- 2593 A sequence of attributes that apply to the subject.
- 2594 Typically, a <Subject> element will contain an <Attribute> with an AttributeId of
2595 "urn:oasis:names:tc:xacml:1.0:subject:subject-id", containing the identity of the **subject**.
- 2596 A <Subject> element MAY contain additional <Attribute> elements.

2597 6.3. Element <Resource>

- 2598 The <Resource> element specifies information about the **resource** to which **access** is requested,
2599 by listing a sequence of <Attribute> elements associated with the **resource**. It MAY include the
2600 **resource** content.

```
2601 <xs:element name="Resource" type="xacml-context:ResourceType"/>
2602 <xs:complexType name="ResourceType">
2603 <xs:sequence>
2604 <xs:element ref="xacml-context:ResourceContent" minOccurs="0"/>
2605 <xs:element ref="xacml-context:Attribute" minOccurs="0"
2606 maxOccurs="unbounded"/>
2607 </xs:sequence>
2608 </xs:complexType>
```

- 2609 The <Resource> element is of **ResourceType** complex type.

- 2610 The <Resource> element contains the following elements:

2611 <ResourceContent> [Optional]

2612 The **resource** content.

2613 <Attribute> [Any Number]

2614 A sequence of **resource attributes**. The <Resource> element MUST contain one and
2615 only one <Attribute> with an AttributeId of
2616 "urn:oasis:names:tc:xacml:1.0:resource:resource-id". This **attribute**
2617 specifies the identity of the **resource** to which **access** is requested.

2618 A <Resource> element MAY contain additional <Attribute> elements.

2619 6.4. Element <ResourceContent>

- 2620 The <ResourceContent> element is a notional placeholder for the **resource** content. If an
2621 XACML **policy** references the contents of the **resource**, then the <ResourceContent> element
2622 SHALL be used as the reference point.

```
2623 <xs:complexType name="ResourceContentType" mixed="true">
2624 <xs:sequence>
2625 <xs:any namespace="##any" processContents="lax" minOccurs="0"
2626 maxOccurs="unbounded"/>
2627 </xs:sequence>
2628 <xs:anyAttribute namespace="##any" processContents="lax"/>
2629 </xs:complexType>
```

- 2630 The <ResourceContent> element is of **ResourceContentType** complex type.

- 2631 The <ResourceContent> element allows arbitrary elements and attributes.

2632

6.5. Element <Action>

2633 The <Action> element specifies the requested **action** on the **resource**, by listing a set of
2634 <Attribute> elements associated with the **action**.

```
2635 <xs:element name="Action" type="xacml-context:ActionType"/>
2636 <xs:complexType name="ActionType">
2637   <xs:sequence>
2638     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2639     maxOccurs="unbounded"/>
2640   </xs:sequence>
2641 </xs:complexType>
```

2642 The <Action> element is of **ActionType** complex type.

2643 The <Action> element contains the following elements:

2644 <Attribute> [Any Number]

2645 List of **attributes** of the **action** to be performed on the **resource**.

2646 6.6. Element <Environment>

2647 The <Environment> element contains a set of **attributes** of the **environment**. These **attributes**
2648 MAY form part of the **policy** evaluation.

2649

```
2650 <xs:element name="Environment" type="xacml-context:EnvironmentType"/>
2651 <xs:complexType name="EnvironmentType">
2652   <xs:sequence>
2653     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2654     maxOccurs="unbounded"/>
2655   </xs:sequence>
2656 </xs:complexType>
```

2657 The <Environment> element is of **EnvironmentType** complex type.

2658 The <Environment> element contains the following elements:

2659 <Attribute> [Any Number]

2660 A list of **environment attributes**. Environment **attributes** are **attributes** that are not
2661 associated with either the **resource**, the **action** or any of the **subjects** of the **access**
2662 request.

2663 6.7. Element <Attribute>

2664 The <Attribute> element is the central abstraction of the request **context**. It contains an
2665 **attribute** value and **attribute** meta-data. The **attribute** meta-data comprises the **attribute**
2666 identifier, the **attribute** issuer and the **attribute** issue instant. **Attribute** designators and **attribute**
2667 selectors in the **policy** MAY refer to **attributes** by means of this meta-data.

```
2668 <xs:element name="Attribute" type="xacml-context:AttributeType"/>
2669 <xs:complexType name="AttributeType">
2670   <xs:sequence>
2671     <xs:element ref="xacml-context:AttributeValue"/>
2672   </xs:sequence>
2673   <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
2674   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
2675   <xs:attribute name="Issuer" type="xs:string" use="optional"/>
```

2676 `<xs:attribute name="IssueInstant" type="xs:dateTime" use="optional"/>`
2677 `</xs:complexType>`

2678 The <Attribute> element is of **AttributeType** complex type.

2679 The <Attribute> element contains the following attributes and elements:

2680 AttributeId [Required]

2681 **Attribute** identifier. A number of identifiers are reserved by XACML to denote commonly
2682 used **attributes**.

2683 DataType [Required]

2684 The data-type of the contents of the <AttributeValue> element. This SHALL be either
2685 a primitive type defined by the XACML 1.0 specification or a type defined in a namespace
2686 declared in the <xacml-context> element.

2687 Issuer [Optional]

2688 **Attribute** issuer. This attribute value MAY be an x500Name that binds to a public key, or it
2689 may be some other identifier exchanged out-of-band by issuing and relying parties.

2690 IssueInstant [Optional]

2691 The date and time at which the **attribute** was issued.

2692

2693 <AttributeValue> [Required]

2694 Exactly one **attribute** value. The mandatory **attribute** value MAY have contents that are empty,
2695 occur once, or occur multiple times.

2696 **6.8. Element <AttributeValue>**

2697 The <AttributeValue> element contains the value of an **attribute**.

```
2698 <xs:element name="AttributeValue" type="xacml-context:AttributeValueType"/>
2699 <xs:complexType name="AttributeValueType" mixed="true">
2700 <xs:sequence>
2701 <xs:any namespace="##any" processContents="lax" minOccurs="0"
2702 maxOccurs="unbounded"/>
2703 </xs:sequence>
2704 <xs:anyAttribute namespace="##any" processContents="lax"/>
2705 </xs:complexType>
```

2706 The <AttributeValue> element is of **AttributeValueType** type.

2707 The data-type of the <AttributeValue> MAY be specified by using the `DataType` attribute of
2708 the parent <Attribute> element.

2709 **6.9. Element <Response>**

2710 The <Response> element is a top-level element in the XACML **context** schema. The
2711 <Response> element is an abstraction layer used by the **policy** language. Any proprietary
2712 system using the XACML specification MUST transform an XACML **context** <Response> into the
2713 form of its **authorization decision**.

2714 The <Response> element encapsulates the **authorization decision** produced by the **PDP**. It
2715 includes a sequence of one or more results, with one <Result> element per requested **resource**.
2716 Multiple results MAY be returned when the value of the “urn:oasis:xacml:1.0:resource:scope”
2717 resource **attribute** in the request **context** is “Descendants” or “Children”. Support for multiple
2718 results is OPTIONAL.

```
2719 <xs:element name="Response" type="xacml-context:ResponseType"/>  
2720 <xs:complexType name="ResponseType">  
2721 <xs:sequence>  
2722 <xs:element ref="xacml-context:Result" maxOccurs="unbounded"/>  
2723 </xs:sequence>  
2724 </xs:complexType>
```

2725 The <Response> element is of **ResponseType** complex type.

2726 The <Response> element contains the following elements:

2727 <Result> [One to Many]

2728 An authorization decision result.

2729 6.10. Element <Result>

2730 The <Result> element represents an **authorization decision** result for the **resource** specified by
2731 the ResourceId **attribute**. It MAY include a set of **obligations** that MUST be fulfilled by the **PEP**.
2732 If the **PEP** does not understand an **obligation**, then it MUST act as if the **PDP** had denied **access**
2733 to the requested **resource**.

```
2734  
2735 <xs:element name="Result" type="xacml-context:ResultType"/>  
2736 <xs:complexType name="ResultType">  
2737 <xs:sequence>  
2738 <xs:element ref="xacml-context:Decision"/>  
2739 <xs:element ref="xacml-context:Status"/>  
2740 <xs:element ref="xacml:Obligations" minOccurs="0"/>  
2741 </xs:sequence>  
2742 <xs:attribute name="ResourceId" type="xs:string" use="optional"/>  
2743 </xs:complexType>
```

2744 The <Result> element is of **ResultType** complex type.

2745 The <Result> element contains the following attributes and elements:

2746 ResourceId [Optional]

2747 The identifier of the requested **resource**. If this attribute is omitted, then the **resource**
2748 identity is specified by the “urn:oasis:names:tc:xacml:1.0:resource:resource-
2749 id” **resource attribute** in the corresponding <Request> element.

2750 <Decision> [Required]

2751 The **authorization decision**: “Permit”, “Deny”, “Indeterminate” or “NotApplicable”.

2752 <Status> [Required]

2753 Indicates whether errors occurred during evaluation of the **decision request**, and
2754 optionally, information about those errors.

2755 <xacml:Obligations> [Optional]

2756 A list of **obligations** that MUST be fulfilled by the **PEP**. If the **PEP** does not understand an
2757 **obligation**, then it MUST act as if the **PDP** had denied **access** to the requested **resource**.
2758 See Section 7.11 for a description of how the set of **obligations** to be returned by the PDP
2759 is determined.

2760 6.11. Element <Decision>

2761 The <Decision> element contains the result of **policy** evaluation.

```
2762 <xs:element name="Decision" type="xacml-context:DecisionType"/>  
2763 <xs:simpleType name="DecisionType">  
2764 <xs:restriction base="xs:string">  
2765 <xs:enumeration value="Permit"/>  
2766 <xs:enumeration value="Deny"/>  
2767 <xs:enumeration value="Indeterminate"/>  
2768 <xs:enumeration value="NotApplicable"/>  
2769 </xs:restriction>  
2770 </xs:simpleType>
```

2771 The <Decision> element is of **DecisionType** simple type.

2772 The values of the <Decision> element have the following meanings:

2773 "Permit": the requested **access** is permitted.

2774 "Deny": the requested **access** is denied.

2775 "Indeterminate": the **PDP** is unable to evaluate the requested **access**. Reasons for such
2776 inability include: missing **attributes**, network errors while retrieving **policies**, division by
2777 zero during **policy** evaluation, syntax errors in the **decision request** or in the **policy**, etc..

2778 "NotApplicable": the **PDP** does not have any **policy** that applies to this **decision request**.

2779 6.12. Element <Status>

2780 The <Status> element represents the status of the **authorization decision** result.

```
2781 <xs:element name="Status" type="xacml-context:StatusType"/>  
2782 <xs:complexType name="StatusType">  
2783 <xs:sequence>  
2784 <xs:element ref="xacml-context:StatusCode"/>  
2785 <xs:element ref="xacml-context:StatusMessage" minOccurs="0"/>  
2786 <xs:element ref="xacml-context:StatusDetail" minOccurs="0"/>  
2787 </xs:sequence>  
2788 </xs:complexType>
```

2789 The <Status> element is of **StatusType** complex type.

2790 The <Status> element contains the following elements:

2791 <StatusCode> [Required]

2792 Status code.

2793 <StatusMessage> [Optional]

2794 A status message describing the status code.

2795 <StatusDetail> [Optional]

2796 Additional status information.

2797

6.13. Element <StatusCode>

2798 The <StatusCode> element contains a major status code value and an optional sequence of
2799 minor status codes.

```

2800 <xs:element name="StatusCode" type="xacml-context:StatusCodeType"/>
2801 <xs:complexType name="StatusCodeType">
2802   <xs:sequence>
2803     <xs:element ref="xacml-context:StatusCode" minOccurs="0"/>
2804   </xs:sequence>
2805   <xs:attribute name="Value" type="xs:anyURI" use="required"/>
2806 </xs:complexType>

```

2807 The <StatusCode> element is of **StatusCodeType** complex type.

2808 The <StatusCode> element contains the following attributes and elements:

2809 Value [Required]

2810 See Section B.9 for a list of values.

2811 <StatusCode> [Any Number]

2812 Minor status code. This status code qualifies its parent status code.

2813 6.14. Element <StatusMessage>

2814 The <StatusMessage> element is a free-form description of the status code.

```

2815 <xs:element name="StatusMessage" type="xs:string"/>

```

2816 The <StatusMessage> element is of **xs:string** type.

2817 6.15. Element <StatusDetail>

2818 The <StatusDetail> element qualifies the <Status> element with additional information.

```

2819 <xs:element name="StatusDetail" type="xacml-context:StatusDetailType"/>
2820 <xs:complexType name="StatusDetailType">
2821   <xs:sequence>
2822     <xs:any namespace="##any" processContents="lax" minOccurs="0"
2823     maxOccurs="unbounded"/>
2824   </xs:sequence>
2825 </xs:complexType>

```

2826 The <StatusDetail> element is of **StatusDetailType** complex type.

2827 The <StatusDetail> element allows arbitrary XML content.

2828 Inclusion of a <StatusDetail> element is optional. However, if a **PDP** returns one of the
2829 following XACML-defined <StatusCode> values and includes a <StatusDetail> element, then
2830 the following rules apply.

2831 urn:oasis:names:tc:xacml:1.0:status:ok

2832 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “ok” status value.

2833 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

2834 A **PDP** MAY choose not to return any <StatusDetail> information or MAY choose to return a
2835 <StatusDetail> element containing one or more <xacml-context:Attribute> elements. If
2836 the **PDP** includes <AttributeValue> elements in the <Attribute> element, then this indicates

2837 the acceptable values for that **attribute**. If no <AttributeValue> elements are included, then
2838 this indicates the names of **attributes** that the **PDP** failed to resolve during its evaluation. The list
2839 of **attributes** may be partial or complete. There is no guarantee by the **PDP** that supplying the
2840 missing values or **attributes** will be sufficient to satisfy the **policy**.

2841 urn:oasis:names:tc:xacml:1.0:status:syntax-error

2842 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “syntax-error” status
2843 value. A syntax error may represent either a problem with the **policy** being used or with the
2844 request **context**. The **PDP** MAY return a <StatusMessage> describing the problem.

2845 urn:oasis:names:tc:xacml:1.0:status:processing-error

2846 A **PDP** MUST NOT return <StatusDetail> element in conjunction with the “processing-error”
2847 status value. This status code indicates an internal problem in the **PDP**. For security reasons, the
2848 **PDP** MAY choose to return no further information to the **PEP**. In the case of a divide-by-zero error
2849 or other computational error, the **PDP** MAY return a <StatusMessage> describing the nature of
2850 the error.

2851 7. Functional requirements (normative)

2852 This section specifies certain functional requirements that are not directly associated with the
2853 production or consumption of a particular XACML element.

2854 7.1. Policy enforcement point

2855 This section describes the requirements for the **PEP**.

2856 An application functions in the role of the **PEP** if it guards access to a set of **resources** and asks
2857 the **PDP** for an **authorization decision**. The **PEP** MUST abide by the **authorization decision** in
2858 the following way:

2859 A **PEP** SHALL allow access to the **resource** only if a valid XACML response of "Permit" is returned
2860 by the **PDP**. The **PEP** SHALL deny access to the **resource** in all other cases. An XACML
2861 response of "Permit" SHALL be considered valid only if the **PEP** understands all of the **obligations**
2862 contained in the response.

2863 7.2. Base policy

2864 A **PDP** SHALL represent one **policy** or **policy set**, called its **base policy**. This base **policy** MAY be
2865 a <Policy> element containing a <Target> element that matches every possible **decision**
2866 **request**, or (for instance) it MAY be a <Policy> element containing a <Target> element that
2867 matches only a specific **subject**. In such cases, the base policy SHALL form the root-node of a
2868 tree of policies connected by <PolicyIdReference> and <PolicySetIdReference>
2869 elements to all the **rules** that may be applicable to any **decision request** that the **PDP** is capable
2870 of evaluating.

2871 In the case of a **PDP** that retrieves **policies** according to the **decision request** that it is processing,
2872 the base policy SHALL contain a <Policy> element containing a <Target> element that matches
2873 every possible **decision request** and a PolicyCombiningAlgId attribute with the value “Only-
2874 one-applicable”. In other words, the **PDP** SHALL return an error if it retrieves policies that do not
2875 form a single tree.

2876

7.3. Target evaluation

2877 The **target** value SHALL be "Match" if the **subject**, **resource** and **action** specified in the **target** all
 2878 match values in the request **context**. The **target** value SHALL be "No-match" if one or more of the
 2879 **subject**, **resource** and **action** specified in the **target** do not match values in the request **context**.
 2880 The value of a <SubjectMatch>, <ResourceMatch> or <ActionMatch> element, in which a
 2881 referenced **attribute** value cannot be obtained, depends on the value of the `MustBePresent`
 2882 attribute of the <AttributeDesignator> or <AttributeSelector> element. If the
 2883 `MustBePresent` attribute is "True", then the result of the <SubjectMatch>, <ResourceMatch>
 2884 or <ActionMatch> element SHALL be "Indeterminate" in this case. If the `MustBePresent`
 2885 attribute is "False" or missing, then the result of the <SubjectMatch>, <ResourceMatch> or
 2886 <ActionMatch> element SHALL be "No-match".

2887

7.4. Condition evaluation

2888 The **condition** value SHALL be "True" if the <Condition> element is absent, or if it evaluates to
 2889 "True" for the **attribute** values supplied in the request **context**. Its value is "False" if the
 2890 <Condition> element evaluates to "False" for the **attribute** values supplied in the request
 2891 **context**. If any **attribute** value referenced in the **condition** cannot be obtained, then the **condition**
 2892 SHALL evaluate to "Indeterminate".

2893

7.5. Rule evaluation

2894 A **rule** has a value that can be calculated by evaluating its contents. **Rule** evaluation involves
 2895 separate evaluation of the **rule's target** and **condition**. The **rule** truth table is shown in Table 1.

2896

2897

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

2898

Table 1 - Rule truth table

2899 If the **target** value is "No-match" or "Indeterminate" then the **rule** value SHALL be "NotApplicable"
 2900 or "Indeterminate", respectively, regardless of the value of the **condition**. For these cases,
 2901 therefore, the **condition** need not be evaluated in order to determine the **rule** value.

2902 If the **target** value is "Match" and the **condition** value is "True", then the **effect** specified in the **rule**
 2903 SHALL determine the **rule** value.

2904

7.6. Policy evaluation

2905 The value of a **policy** SHALL be determined only by its contents, considered in relation to the
 2906 contents of the **request context**. A **policy's** value SHALL be determined by evaluation of the
 2907 **policy's target** and **rules**, according to the specified **rule-combining algorithm**.

2908 The **policy's target** SHALL be evaluated to determine the applicability of the **policy**. If the **target**
 2909 evaluates to "Match", then the value of the **policy** SHALL be determined by evaluation of the
 2910 **policy's rules**, according to the specified **rule-combining algorithm**. If the **target** evaluates to
 2911 "No-match", then the value of the **policy** SHALL be "NotApplicable". If the **target** evaluates to
 2912 "Indeterminate", then the value of the **policy** SHALL be "Indeterminate".

2913 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"

2914

Table 2 - Policy truth table

2915 A **rules** value of "At least one rule value is its Effect" SHALL be used if the <Rule> element is
 2916 absent, or if one or more of the **rules** contained in the **policy** is applicable to the **decision request**
 2917 (i.e., returns a value of "Effect"; see Section 7.5). A **rules** value of "All rule values are
 2918 'NotApplicable'" SHALL be used if no **rule** contained in the **policy** is applicable to the request and if
 2919 no **rule** contained in the **policy** returns a value of "Indeterminate". If no **rule** contained in the
 2920 **policy** is applicable to the request but one or more **rule** returns a value of "Indeterminate", then
 2921 **rules** value SHALL evaluate to "At least one rule value is 'Indeterminate'".

2922 If the **target** value is "No-match" or "Indeterminate" then the **policy** value SHALL be
 2923 "NotApplicable" or "Indeterminate", respectively, regardless of the value of the **rules**. For these
 2924 cases, therefore, the **rules** need not be evaluated in order to determine the **policy** value.

2925 If the **target** value is "Match" and the **rules** value is "At least one rule value is its Effect" or "At least
 2926 one rule value is 'Indeterminate'", then the **rule-combining algorithm** specified in the **policy**
 2927 SHALL determine the **policy** value.

2928 7.7. Policy Set evaluation

2929 The value of a **policy set** SHALL be determined by its contents, considered in relation to the
 2930 contents of the **request context**. A **policy set's** value SHALL be determined by evaluation of the
 2931 **policy set's target**, **policies** and **policy sets**, according to the specified **policy-combining**
 2932 **algorithm**.

2933 The **policy set's target** SHALL be evaluated to determine the applicability of the **policy set**. If the
 2934 **target** evaluates to "Match" then the value of the **policy set** SHALL be determined by evaluation of
 2935 the **policy set's policies** and **policy sets**, according to the specified **policy-combining algorithm**.
 2936 If the **target** evaluates to "No-match", then the value of the **policy set** shall be "NotApplicable". If
 2937 the **target** evaluates to "Indeterminate", then the value of the **policy set** SHALL be "Indeterminate".

2938 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"

Table 3 – Policy set truth table

2939

2940 A **policies** value of "At least one policy value is its **Decision**" SHALL be used if there are no
 2941 contained or referenced **policies** or **policy sets**, or if one or more of the **policies** or **policy sets**
 2942 contained in or referenced by the **policy set** is applicable to the **decision request** (i.e., returns a
 2943 value determined by its **rule-combining algorithm**; see Section 7.6). A **policies** value of "All
 2944 policy values are 'NotApplicable'" SHALL be used if no **policy** or **policy set** contained in or
 2945 referenced by the **policy set** is applicable to the request and if no **policy** or **policy set** contained in
 2946 or referenced by the **policy set** returns a value of "Indeterminate". If no **policy** or **policy set**
 2947 contained in or referenced by the **policy set** is applicable to the request but one or more **policy** or
 2948 **policy set** returns a value of "Indeterminate", then **policies** SHALL evaluate to "At least one policy
 2949 value is 'Indeterminate'".

2950 If the **target** value is "No-match" or "Indeterminate" then the **policy set** value SHALL be
 2951 "NotApplicable" or "Indeterminate", respectively, regardless of the value of the **policies**. For these
 2952 cases, therefore, the **policies** need not be evaluated in order to determine the **policy set** value.

2953 If the **target** value is "Match" and the **policies** value is "At least one policy value is its **Decision**" or
 2954 "At least one policy value is 'Indeterminate'", then the **policy-combining algorithm** specified in the
 2955 **policy set** SHALL determine the **policy set** value.

7.8. Hierarchical resources

2956

2957 It is often the case that a **resource** is organized as a hierarchy (e.g. file system, XML document).
 2958 Some access requesters may request **access** to an entire subtree of a **resource** specified by a
 2959 node. XACML allows the **PEP** (or **context handler**) to specify whether the **decision request** is
 2960 just for a single **resource** or for a subtree below the specified **resource**. The latter is equivalent to
 2961 repeating a single request for each node in the entire subtree. When a request **context** contains a
 2962 resource attribute of type

2963 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2964 with a value of "Immediate", or if it does not contain that **attribute**, then the **decision request**
 2965 SHALL be interpreted to apply to just the single **resource** specified by the
 2966 "urn:oasis:names:tc:xacml:1.0:resource:resource-id" **attribute**.

2967 When the

2968 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2969 **attribute** has the value "Children", the **decision request** SHALL be interpreted to apply to the
2970 specified **resource** and its immediate children **resources**.

2971 When the
2972 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2973 **attribute** has the value "Descendants", the **decision request** SHALL be interpreted to apply to
2974 both the specified **resource** and all its descendant **resources**.

2975 In the case of "Children" and "Descendants", the **authorization decision** MAY include multiple
2976 results for the multiple sub-nodes in the **resource** sub-tree.

2977 An XACML **authorization response** MAY contain multiple <Result> elements.

2978 Note that the method by which the **PDP** discovers whether the **resource** is hierarchically organized
2979 or not is outside the scope of XACML.

2980 In the case where a child or descendant **resource** cannot be accessed, the <Result> element
2981 associated with the parent element SHALL contain a <StatusCode> Value of
2982 "urn:oasis:names:tc:xacml:1.0:status:processing-error".

2983 **7.9. Attributes**

2984 **Attributes** are specified in the request **context**, regardless of whether or not they appeared in the
2985 original **decision request**, and are referred to in the **policy** by **subject**, **resource**, **action** and
2986 **environment attribute** designators and **attribute** selectors. A **named attribute** is the term used for
2987 the criteria that the specific **subject**, **resource**, **action** and **environment attribute** designators and
2988 selectors use to refer to **attributes** in the **subject**, **resource**, **action** and **environment** elements of
2989 the request **context**, respectively.

2990 **7.9.1. Attribute Matching**

2991 A **named attribute** has specific criteria with which to match **attributes** in the **context**. An **attribute**
2992 specifies **AttributeId**, **DataType** and **Issuer** attributes, and each **named attribute** also
2993 specifies **AttributeId**, **DataType** and optional **Issuer** attributes. A **named attribute** SHALL
2994 match an **attribute** if the values of their respective **AttributeId**, **DataType** and optional **Issuer**
2995 attributes match within their particular element, e.g. **subject**, **resource**, **action** or **environment**, of
2996 the **context**. The **AttributeId** of the named **attribute** MUST match, by URI equality, the
2997 **AttributeId** of the context **attribute**. The **DataType** of the named **attribute** MUST match, by
2998 URI equality, the **DataType** of the same context **attribute**. If **Issuer** is supplied in the named
2999 **attribute**, then it MUST match, by string equality, the **Issuer** of the same context **attribute**. If
3000 **Issuer** is not supplied in the **named attribute**, then the matching of the context **attribute** to the
3001 **named attribute** SHALL be governed by **AttributeId** and **DataType** alone, regardless of the
3002 presence, absence, or actual value of **Issuer**. In the case of an **attribute** selector, the matching
3003 of the **attribute** to the **named attribute** SHALL be governed by the XPath expression and
3004 **DataType**.

3005 **7.9.2. Attribute Retrieval**

3006 The **PDP** SHALL request the values of **attributes** in the request **context** from the **context handler**.
3007 The **PDP** SHALL reference the **attributes** as if they were in a physical request **context** document,
3008 but the **context handler** is responsible for obtaining and supplying the requested values. The
3009 **context handler** SHALL return the values of **attributes** that match the **attribute** designator or
3010 **attribute** selector and form them into a **bag** of values with the specified data-type. If no **attributes**

3011 from the request **context** match, then the **attribute** SHALL be considered missing. If the **attribute**
3012 is missing, then `MustBePresent` governs whether the **attribute** designator or **attribute** selector
3013 returns an empty **bag** or an "Indeterminate" result. If `MustBePresent` is "False" (default value),
3014 then a missing **attribute** SHALL result in an empty **bag**. If `MustBePresent` is "True", then a
3015 missing **attribute** SHALL result in "Indeterminate". This "Indeterminate" result SHALL be handled
3016 in accordance with the specification of the encompassing expressions, **rules**, **policies** and **policy**
3017 **sets**. If the result is "Indeterminate", then the `AttributeId`, `DataType` and `Issuer` of the
3018 **attribute** MAY be listed in the **authorization decision** as described in Section 7.10. However, a
3019 **PDP** MAY choose not to return such information for security reasons.

3020 7.9.3. Environment Attributes

3021 **Environment attributes** are listed in Section B.8. If a value for one of these **attributes** is supplied
3022 in the **decision request**, then the **context handler** SHALL use that value. Otherwise, the **context**
3023 **handler** SHALL supply a value. For the date and time **attributes**, the supplied value SHALL have
3024 the semantics of "date and time that apply to the **decision request**".

3025 7.10. Authorization decision

3026 Given a valid XACML **policy** or **policy set**, a compliant XACML **PDP** MUST evaluate the **policy** as
3027 specified in Sections 5, 0 and 4.2. The **PDP** MUST return a response **context**, with one
3028 `<Decision>` element of value "Permit", "Deny", "Indeterminate" or "NotApplicable".

3029 If the **PDP** cannot make a decision, then an "Indeterminate" `<Decision>` element contents SHALL
3030 be returned. The **PDP** MAY return a `<Decision>` element contents of "Indeterminate" with a
3031 status code of:

3032 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3033 signifying that more information is needed. In this case, the `<Status>` element MAY list the
3034 names and data-types of any **attributes** of the **subjects**, **resource**, **action**, or **environment** that
3035 are needed by the **PDP** to refine its decision. A **PEP** MAY resubmit a refined request **context** in
3036 response to a `<Decision>` element contents of "Indeterminate" with a status code of

3037 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3038 by adding **attribute** values for the **attribute** names that were listed in the previous response. When
3039 the **PDP** returns a `<Decision>` element contents of "Indeterminate", with a status code of

3040 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3041 it MUST NOT list the names and data-types of any **attribute** of the **subject**, **resource**, **action**, or
3042 **environment** for which values were supplied in the original request. Note, this requirement forces
3043 the **PDP** to eventually return an **authorization decision** of "Permit", "Deny" or "Indeterminate" with
3044 some other status code, in response to successively-refined requests.

3045 7.11. Obligations

3046 A **policy** or **policy set** may contain one or more **obligations**. When such a **policy** or **policy set** is
3047 evaluated, an **obligation** SHALL be passed up to the next level of evaluation (the enclosing or
3048 referencing **policy set** or **authorization decision**) only if the **effect** of the **policy** or **policy set**
3049 being evaluated matches the value of the `xacml:FulfillOn` attribute of the **obligation**.
3050

3051 As a consequence of this procedure, no **obligations** SHALL be returned to the **PEP** if the **policies**
3052 or **policy sets** from which they are drawn are not evaluated, or if their evaluated result is

3053 "Indeterminate" or "NotApplicable", or if the **decision** resulting from evaluating the **policy** or **policy set** does not match the **decision** resulting from evaluating an enclosing **policy set**.

3054
3055
3056 If the **PDP's** evaluation is viewed as a tree of **policy sets** and **policies**, each of which returns
3057 "Permit" or "Deny", then the set of **obligations** returned by the **PDP** to the **PEP** will include only the
3058 **obligations** associated with those paths where the **effect** at each level of evaluation is the same as
3059 the **effect** being returned by the **PDP**.

3060 A **PEP** that receives a valid XACML response of "Permit" with **obligations** SHALL be responsible
3061 for fulfilling *all* of those **obligations**. A **PEP** that receives an XACML response of "Deny" with
3062 **obligations** SHALL be responsible for fulfilling all of the **obligations** that it *understands*.

3063 **7.12. Unsupported functionality**

3064 If the **PDP** attempts to evaluate a **policy set** or **policy** that contains an optional element type or
3065 feature that the **PDP** does not support, then the **PDP** SHALL return a <Decision> value of
3066 "Indeterminate". If a <StatusCode> element is also returned, then its value SHALL be
3067 "urn:oasis:names:tc:xacml:1.0:status:syntax-error" in the case of an unsupported element type, and
3068 "urn:oasis:names:tc:xacml:1.0:status:processing-error" in the case of an unsupported feature.

3069 **7.13. Syntax and type errors**

3070 If a **policy** that contains invalid syntax is evaluated by the XACML **PDP** at the time a **decision request**
3071 is received, then the result of that **policy** SHALL be "Indeterminate" with a StatusCode
3072 value of "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

3073 If a **policy** that contains invalid static data-types is evaluated by the XACML **PDP** at the time a
3074 **decision request** is received, then the result of that **policy** SHALL be "Indeterminate" with a
3075 StatusCode value of "urn:oasis:names:tc:xacml:1.0:status:processing-error".

3076 **8. XACML extensibility points (non-normative)**

3077 This section describes the points within the XACML model and schema where extensions can be
3078 added

3079 **8.1. Extensible XML attribute types**

3080 The following XML attributes have values that are URIs. These may be extended by the creation of
3081 new URIs associated with new semantics for these attributes.

3082 AttributeId,

3083 AttributeValue,

3084 DataType,

3085 FunctionId,

3086 MatchId,

3087 ObligationId,

3088 PolicyCombiningAlgId,

3089 RuleCombiningAlgId,

3090 StatusCode,
3091 SubjectCategory.
3092 See Section 5 for definitions of these attribute types.

3093 **8.2. Structured attributes**

3094 An XACML <AttributeValue> element MAY contain an instance of a structured XML data-type.
3095 Section A.3 describes a number of standard techniques to identify data items within such a
3096 structured attribute. Listed here are some additional techniques that require XACML extensions.

- 3097 1. For a given structured data-type, a community of XACML users MAY define new attribute
3098 identifiers for each leaf sub-element of the structured data-type that has a type conformant
3099 with one of the XACML-defined primitive data-types. Using these new attribute identifiers,
3100 the **PEPs** or **context handlers** used by that community of users can flatten instances of
3101 the structured data-type into a sequence of individual <Attribute> elements. Each such
3102 <Attribute> element can be compared using the XACML-defined functions. Using this
3103 method, the structured data-type itself never appears in an <AttributeValue> element.
- 3104 2. A community of XACML users MAY define a new function that can be used to compare a
3105 value of the structured data-type against some other value. This method may only be used
3106 by **PDPs** that support the new function.

3107 **9. Security and privacy considerations (non-** 3108 **normative)**

3109 This section identifies possible security and privacy compromise scenarios that should be
3110 considered when implementing an XACML-based system. The section is informative only. It is left
3111 to the implementer to decide whether these compromise scenarios are practical in their
3112 environment and to select appropriate safeguards.

3113 **9.1. Threat model**

3114 We assume here that the adversary has access to the communication channel between the
3115 XACML actors and is able to interpret, insert, delete and modify messages or parts of messages.

3116 Additionally, an actor may use information from a former transaction maliciously in subsequent
3117 transactions. It is further assumed that **rules** and **policies** are only as reliable as the actors that
3118 create and use them. Thus it is incumbent on each actor to establish appropriate trust in the other
3119 actors upon which it relies. Mechanisms for trust establishment are outside the scope of this
3120 specification.

3121 The messages that are transmitted between the actors in the XACML model are susceptible to
3122 attack by malicious third parties. Other points of vulnerability include the **PEP**, the **PDP** and the
3123 **PAP**. While some of these entities are not strictly within the scope of this specification, their
3124 compromise could lead to the compromise of **access control** enforced by the **PEP**.

3125 It should be noted that there are other components of a distributed system that may be
3126 compromised, such as an operating system and the domain-name system (DNS) that are outside
3127 the scope of this discussion of threat models. Compromise in these components may also lead to a
3128 policy violation.

3129 The following sections detail specific compromise scenarios that may be relevant to an XACML
3130 system.

3131 **9.1.1. Unauthorized disclosure**

3132 XACML does not specify any inherent mechanisms for confidentiality of the messages exchanged
3133 between actors. Therefore, an adversary could observe the messages in transit. Under certain
3134 security policies, disclosure of this information is a violation. Disclosure of **attributes** or the types
3135 of **decision requests** that a **subject** submits may be a breach of privacy policy. In the commercial
3136 sector, the consequences of unauthorized disclosure of personal data may range from
3137 embarrassment to the custodian to imprisonment and large fines in the case of medical or financial
3138 data.

3139 Unauthorized disclosure is addressed by confidentiality mechanisms.

3140 **9.1.2. Message replay**

3141 A message replay attack is one in which the adversary records and replays legitimate messages
3142 between XACML actors. This attack may lead to denial of service, the use of out-of-date
3143 information or impersonation.

3144 Prevention of replay attacks requires the use of message freshness mechanisms.

3145 Note that encryption of the message does not mitigate a replay attack since the message is just
3146 replayed and does not have to be understood by the adversary.

3147 **9.1.3. Message insertion**

3148 A message insertion attack is one in which the adversary inserts messages in the sequence of
3149 messages between XACML actors.

3150 The solution to a message insertion attack is to use mutual authentication and a message
3151 sequence integrity mechanism between the actors. It should be noted that just using SSL mutual
3152 authentication is not sufficient. This only proves that the other party is the one identified by the
3153 subject of the X.509 certificate. In order to be effective, it is necessary to confirm that the certificate
3154 subject is authorized to send the message.

3155 **9.1.4. Message deletion**

3156 A message deletion attack is one in which the adversary deletes messages in the sequence of
3157 messages between XACML actors. Message deletion may lead to denial of service. However, a
3158 properly designed XACML system should not render an incorrect authorization decision as a result
3159 of a message deletion attack.

3160 The solution to a message deletion attack is to use a message integrity mechanism between the
3161 actors.

3162 **9.1.5. Message modification**

3163 If an adversary can intercept a message and change its contents, then they may be able to alter an
3164 **authorization decision**. Message integrity mechanisms can prevent a successful message
3165 modification attack.

3166

9.1.6. NotApplicable results

3167 A result of "NotApplicable" means that the *PDP* did not have a policy whose target matched the
3168 information in the *decision request*. In general, we highly recommend using a "default-deny"
3169 policy, so that when a *PDP* would have returned "NotApplicable", a result of "Deny" is returned
3170 instead.

3171 In some security models, however, such as is common in many Web Servers, a result of
3172 "NotApplicable" is treated as equivalent to "Permit". There are particular security considerations
3173 that must be taken into account for this to be safe. These are explained in the following
3174 paragraphs.

3175 If "NotApplicable" is to be treated as "Permit", it is vital that the matching algorithms used by the
3176 policy to match elements in the decision request are closely aligned with the data syntax used by
3177 the applications that will be submitting the decision request. A failure to match will be treated as
3178 "Permit", so an unintended failure to match may allow unintended access.

3179 A common example of this is a Web Server. Commercial http responders allow a variety of
3180 syntaxes to be treated equivalently. The "%" can be used to represent characters by hex value.
3181 The URL path "/../" provides multiple ways of specifying the same value. Multiple character sets
3182 may be permitted and, in some cases, the same printed character can be represented by different
3183 binary values. Unless the matching algorithm used by the policy is sophisticated enough to catch
3184 these variations, unintended access may be permitted.

3185 It is safe to treat "NotApplicable" as "Permit" only in a closed environment where all applications
3186 that formulate a decision request can be guaranteed to use the exact syntax expected by the
3187 policies used by the *PDP*. In a more open environment, where decision requests may be received
3188 from applications that may use any legal syntax, it is strongly recommended that "NotApplicable"
3189 NOT be treated as "Permit" unless matching rules have been very carefully designed to match all
3190 possible applicable inputs, regardless of syntax or type variations.

3191

9.1.7. Negative rules

3192 A negative *rule* is one that is based on a *predicate* not being "True". If not used with care,
3193 negative *rules* can lead to policy violation, therefore some authorities recommend that they not be
3194 used. However, negative *rules* can be extremely efficient in certain cases, so XACML has chosen
3195 to include them. Nevertheless, it is recommended that they be used with care and avoided if
3196 possible.

3197 A common use for negative *rules* is to deny *access* to an individual or subgroup when their
3198 membership in a larger group would otherwise permit them access. For example, we might want to
3199 write a *rule* that allows all Vice Presidents to see the unpublished financial data, except for Joe,
3200 who is only a Ceremonial Vice President and can be indiscreet in his communications. If we have
3201 complete control of the administration of *subject attributes*, a superior approach would be to
3202 define "Vice President" and "Ceremonial Vice President" as distinct groups and then define *rules*
3203 accordingly. However, in some environments this approach may not be feasible. (It is worth noting
3204 in passing that, generally speaking, referring to individuals in *rules* does not scale well. Generally,
3205 shared *attributes* are preferred.)

3206 If not used with care, negative *rules* can lead to policy violation in two common cases. They are:
3207 when *attributes* are suppressed and when the base group changes. An example of suppressed
3208 *attributes* would be if we have a policy that *access* should be permitted, *unless* the *subject* is a
3209 credit risk. If it is possible that the *attribute* of being a credit risk may be unknown to the *PDP* for
3210 some reason, then unauthorized *access* may be permitted. In some environments, the *subject*
3211 may be able to suppress the publication of *attributes* by the application of privacy controls, or the
3212 server or repository that contains the information may be unavailable for accidental or intentional
3213 reasons.

3214 An example of a changing base group would be if there is a policy that everyone in the engineering
3215 department may change software source code, except for secretaries. Suppose now that the
3216 department was to merge with another engineering department and the intent is to maintain the
3217 same policy. However, the new department also includes individuals identified as administrative
3218 assistants, who ought to be treated in the same way as secretaries. Unless the policy is altered,
3219 they will unintentionally be permitted to change software source code. Problems of this type are
3220 easy to avoid when one individual administers all **policies**, but when administration is distributed,
3221 as XACML allows, this type of situation must be explicitly guarded against.

3222 **9.2. Safeguards**

3223 **9.2.1. Authentication**

3224 Authentication provides the means for one party in a transaction to determine the identity of the
3225 other party in the transaction. Authentication may be in one direction, or it may be bilateral.

3226 Given the sensitive nature of **access control** systems, it is important for a **PEP** to authenticate the
3227 identity of the **PDP** to which it sends **decision requests**. Otherwise, there is a risk that an
3228 adversary could provide false or invalid **authorization decisions**, leading to a policy violation.

3229 It is equally important for a **PDP** to authenticate the identity of the **PEP** and assess the level of trust
3230 to determine what, if any, sensitive data should be passed. One should keep in mind that even
3231 simple "Permit" or "Deny" responses could be exploited if an adversary were allowed to make
3232 unlimited requests to a **PDP**.

3233 Many different techniques may be used to provide authentication, such as co-located code, a
3234 private network, a VPN or digital signatures. Authentication may also be performed as part of the
3235 communication protocol used to exchange the **contexts**. In this case, authentication may be
3236 performed at the message level or at the session level.

3237 **9.2.2. Policy administration**

3238 If the contents of **policies** are exposed outside of the **access control** system, potential **subjects**
3239 may use this information to determine how to gain unauthorized **access**.

3240 To prevent this threat, the repository used for the storage of **policies** may itself require **access**
3241 **control**. In addition, the <Status> element should be used to return values of missing **attributes**
3242 only when exposure of the identities of those **attributes** will not compromise security.

3243 **9.2.3. Confidentiality**

3244 Confidentiality mechanisms ensure that the contents of a message can be read only by the desired
3245 recipients and not by anyone else who encounters the message while it is in transit. There are two
3246 areas in which confidentiality should be considered: one is confidentiality during transmission; the
3247 other is confidentiality within a <Policy> element.

3248 **9.2.3.1. Communication confidentiality**

3249 In some environments it is deemed good practice to treat all data within an **access control** system
3250 as confidential. In other environments, **policies** may be made freely available for distribution,
3251 inspection and audit. The idea behind keeping **policy** information secret is to make it more difficult
3252 for an adversary to know what steps might be sufficient to obtain unauthorized **access**. Regardless
3253 of the approach chosen, the security of the **access control** system should not depend on the
3254 secrecy of the **policy**.

3255 Any security concerns or requirements related to transmitting or exchanging XACML <Policy>
3256 elements are outside the scope of the XACML standard. While it is often important to ensure that
3257 the integrity and confidentiality of <Policy> elements is maintained when they are exchanged
3258 between two parties, it is left to the implementers to determine the appropriate mechanisms for their
3259 environment.

3260 Communications confidentiality can be provided by a confidentiality mechanism, such as SSL.
3261 Using a point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points
3262 is compromised.

3263 **9.2.3.2. Statement level confidentiality**

3264 In some cases, an implementation may want to encrypt only parts of an XACML <Policy>
3265 element.

3266 The XML Encryption Syntax and Processing Candidate Recommendation from W3C can be used
3267 to encrypt all or parts of an XML document. This specification is recommended for use with
3268 XACML.

3269 It should go without saying that if a repository is used to facilitate the communication of cleartext
3270 (i.e., unencrypted) **policy** between the **PAP** and **PDP**, then a secure repository should be used to
3271 store this sensitive data.

3272 **9.2.4. Policy integrity**

3273 The XACML **policy**, used by the **PDP** to evaluate the request **context**, is the heart of the system.
3274 Therefore, maintaining its integrity is essential. There are two aspects to maintaining the integrity of
3275 the **policy**. One is to ensure that <Policy> elements have not been altered since they were
3276 originally created by the **PAP**. The other is to ensure that <Policy> elements have not been
3277 inserted or deleted from the set of **policies**.

3278 In many cases, both aspects can be achieved by ensuring the integrity of the actors and
3279 implementing session-level mechanisms to secure the communication between actors. The
3280 selection of the appropriate mechanisms is left to the implementers. However, when **policy** is
3281 distributed between organizations to be acted on at a later time, or when the **policy** travels with the
3282 protected resource, it would be useful to sign the **policy**. In these cases, the XML Signature
3283 Syntax and Processing standard from W3C is recommended to be used with XACML.

3284 Digital signatures should only be used to ensure the integrity of the statements. Digital signatures
3285 should not be used as a method of selecting or evaluating **policy**. That is, the **PDP** should not
3286 request a **policy** based on who signed it or whether or not it has been signed (as such a basis for
3287 selection would, itself, be a matter of policy). However, the **PDP** must verify that the key used to
3288 sign the **policy** is one controlled by the purported issuer of the **policy**. The means to do this are
3289 dependent on the specific signature technology chosen and are outside the scope of this document.

3290 **9.2.5. Policy identifiers**

3291 Since **policies** can be referenced by their identifiers, it is the responsibility of the **PAP** to ensure
3292 that these are unique. Confusion between identifiers could lead to misidentification of the
3293 **applicable policy**. This specification is silent on whether a **PAP** must generate a new identifier
3294 when a **policy** is modified or may use the same identifier in the modified **policy**. This is a matter of
3295 administrative practice. However, care must be taken in either case. If the identifier is reused,
3296 there is a danger that other **policies** or **policy sets** that reference it may be adversely affected.
3297 Conversely, if a new identifier is used, these other **policies** may continue to use the prior **policy**,
3298 unless it is deleted. In either case the results may not be what the **policy** administrator intends.

3299

9.2.6. Trust model

3300 Discussions of authentication, integrity and confidentiality mechanisms necessarily assume an
3301 underlying trust model: how can one actor come to believe that a given key is uniquely associated
3302 with a specific, identified actor so that the key can be used to encrypt data for that actor or verify
3303 signatures (or other integrity structures) from that actor? Many different types of trust model exist,
3304 including strict hierarchies, distributed authorities, the Web, the bridge and so on.

3305 It is worth considering the relationships between the various actors of the **access control** system in
3306 terms of the interdependencies that do and do not exist.

- 3307 • None of the entities of the authorization system are dependent on the **PEP**. They may
3308 collect data from it, for example authentication, but are responsible for verifying it.
- 3309 • The correct operation of the system depends on the ability of the **PEP** to actually enforce
3310 **policy** decisions.
- 3311 • The **PEP** depends on the **PDP** to correctly evaluate **policies**. This in turn implies that the
3312 **PDP** is supplied with the correct inputs. Other than that, the **PDP** does not depend on the
3313 **PEP**.
- 3314 • The **PDP** depends on the **PAP** to supply appropriate policies. The **PAP** is not dependent
3315 on other components.

3316

9.2.7. Privacy

3317 It is important to be aware that any transactions that occur with respect to **access control** may
3318 reveal private information about the actors. For example, if an XACML **policy** states that certain
3319 data may only be read by **subjects** with “Gold Card Member” status, then any transaction in which
3320 a **subject** is permitted **access** to that data leaks information to an adversary about the **subject's**
3321 status. Privacy considerations may therefore lead to encryption and/or to **access control policies**
3322 surrounding the enforcement of XACML **policy** instances themselves: confidentiality-protected
3323 channels for the request/response protocol messages, protection of **subject attributes** in storage
3324 and in transit, and so on.

3325 Selection and use of privacy mechanisms appropriate to a given environment are outside the scope
3326 of XACML. The decision regarding whether, how and when to deploy such mechanisms is left to
3327 the implementers associated with the environment.

10. Conformance (normative)

3328

10.1. Introduction

3329

3330 The XACML specification addresses the following aspect of conformance:

3331 The XACML specification defines a number of functions, etc. that have somewhat specialist
3332 application, therefore they are not required to be implemented in an implementation that claims to
3333 conform with the OASIS standard.

10.2. Conformance tables

3334

3335 This section lists those portions of the specification that **MUST** be included in an implementation of
3336 a **PDP** that claims to conform with XACML v1.0. A set of test cases has been created to assist in
3337 this process. These test cases are hosted by Sun Microsystems and can be located from the

3338 XACML Web page. The site hosting the test cases contains a full description of the test cases and
 3339 how to execute them.

3340 Note: "M" means mandatory-to-implement. "O" means optional.

3341 **10.2.1. Schema elements**

3342 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

3343 **10.2.2. Identifier Prefixes**

3344 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

3345 **10.2.3. Algorithms**

3346 The implementation MUST include the rule- and policy-combining algorithms associated with the
3347 following identifiers that are marked "M".

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

3348 **10.2.4. Status Codes**

3349 Implementation support for the urn:oasis:names:tc:xacml:1.0:context:status element is optional, but
3350 if the element is supported, then the following status codes must be supported and must be used in
3351 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
--	---

3352 10.2.5. Attributes

3353 The implementation MUST support the attributes associated with the following attribute identifiers
 3354 as specified by XACML. If values for these **attributes** are not present in the **decision request**,
 3355 then their values MUST be supplied by the **PDP**. So, unlike most other **attributes**, their semantics
 3356 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

3357 10.2.6. Identifiers

3358 The implementation MUST use the attributes associated with the following identifiers in the way
 3359 XACML has defined. This requirement pertains primarily to implementations of a **PAP** or **PEP** that
 3360 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

3361 10.2.7. Data-types

3362 The implementation MUST support the data-types associated with the following identifiers marked
 3363 "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

3364

10.2.8. Functions

3365

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

3366

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

3367

11. References

3368

- 3369 [DS] D. Eastlake et al., *XML-Signature Syntax and Processing*,
3370 <http://www.w3.org/TR/xmlsig-core/>, World Wide Web Consortium.
- 3371 [Hancock] Hancock, "Polymorphic Type Checking", in Simon L. Peyton Jones,
3372 "Implementation of Functional Programming Languages", Section 8,
3373 Prentice-Hall International, 1987
- 3374 [Haskell] Haskell, a purely functional language. Available at
3375 <http://www.haskell.org/>
- 3376 [Hinton94] Hinton, H, M, Lee,, E, S, The Compatibility of Policies, Proceedings 2nd
3377 ACM Conference on Computer and Communications Security, Nov 1994,
3378 Fairfax, Virginia, USA.
- 3379 [IEEE754] IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-
3380 7653-8, IEEE Product No. SH10116-TBR
- 3381 [Kudo00] Kudo M and Hada S, XML document security based on provisional
3382 authorization, Proceedings of the Seventh ACM Conference on Computer
3383 and Communications Security, Nov 2000, Athens, Greece, pp 87-96.
- 3384 [LDAP-1] RFC2256, A summary of the X500(96) User Schema for use with LDAPv3,
3385 Section 5, M Wahl, December 1997 <http://www.ietf.org/rfc/rfc2798.txt>
- 3386 [LDAP-2] RFC2798, Definition of the inetOrgPerson, M. Smith, April 2000
3387 <http://www.ietf.org/rfc/rfc2798.txt>
- 3388 [MathML] Mathematical Markup Language (MathML), Version 2.0, W3C
3389 Recommendation, 21 February 2001. Available at:
3390 <http://www.w3.org/TR/MathML2/>
- 3391 [Perritt93] Perritt, H. Knowbots, Permissions Headers and Contract Law, Conference
3392 on Technological Strategies for Protecting Intellectual Property in the
3393 Networked Multimedia Environment, April 1993. Available at:
3394 <http://www.ifla.org/documents/infopol/copyright/perh2.txt>
- 3395 [RBAC] Role-Based Access Controls, David Ferraiolo and Richard Kuhn, 15th
3396 National Computer Security Conference, 1992. Available at:
3397 <http://csrc.nist.gov/rbac>
- 3398 [RegEx] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001,
3399 Appendix D. Available at: <http://www.w3.org/TR/xmlschema-0/>
- 3400 [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
3401 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997
- 3402 [SAML] Security Assertion Markup Language available from [http://www.oasis-
3403 open.org/committees/security/#documents](http://www.oasis-open.org/committees/security/#documents)
- 3404 [Sloman94] Sloman, M. Policy Driven Management for Distributed Systems. Journal
3405 of Network and Systems Management, Volume 2, part 4. Plenum Press.
3406 1994.
- 3407 [XF] XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft
3408 16 August 2002. Available at: [http://www.w3.org/TR/2002/WD-xquery-
3409 operators-20020816](http://www.w3.org/TR/2002/WD-xquery-operators-20020816)
- 3410 [XS] XML Schema, parts 1 and 2. Available at:
3411 <http://www.w3.org/TR/xmlschema-1/> and
3412 <http://www.w3.org/TR/xmlschema-2/>
- 3413 [XPath] XML Path Language (XPath), Version 1.0, W3C Recommendation 16
3414 November 1999. Available at: <http://www.w3.org/TR/xpath>

3415
3416
3417

[XSLT]

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

3418 **Appendix A. Standard data-types, functions and**
3419 **their semantics (normative)**

3420 **A.1. Introduction**

3421 This section contains a specification of the data-types and functions used in XACML to create
3422 **predicates** for a **rule's condition** and **target** matches.

3423 This specification combines the various standards set forth by IEEE and ANSI for string
3424 representation of numeric values, as well as the evaluation of arithmetic functions.

3425 This section describes the primitive data-types, **bags** and construction of expressions using
3426 XACML constructs. Finally, each standard function is named and its operational semantics are
3427 described.

3428 **A.2. Primitive types**

3429 Although XML instances represent all data-types as strings, an XACML **PDP** must reason about
3430 types of data that, while they have string representations, are not just strings. Types such as
3431 boolean, integer and double **MUST** be converted from their XML string representations to values
3432 that can be compared with values in their domain of discourse, such as numbers. The following
3433 primitive data-types are specified for use with XACML and have explicit data representations:

- 3434 • <http://www.w3.org/2001/XMLSchema#string>
- 3435 • <http://www.w3.org/2001/XMLSchema#boolean>
- 3436 • <http://www.w3.org/2001/XMLSchema#integer>
- 3437 • <http://www.w3.org/2001/XMLSchema#double>
- 3438 • <http://www.w3.org/2001/XMLSchema#time>
- 3439 • <http://www.w3.org/2001/XMLSchema#date>
- 3440 • <http://www.w3.org/2001/XMLSchema#dateTime>
- 3441 • <http://www.w3.org/2001/XMLSchema#anyURI>
- 3442 • <http://www.w3.org/2001/XMLSchema#hexBinary>
- 3443 • <http://www.w3.org/2001/XMLSchema#base64Binary>
- 3444 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration>
- 3445 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration>
- 3446 • <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
- 3447 • <urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name>

3448 A.3. Structured types

3449 An XACML `<AttributeValue>` element MAY contain an instance of a structured XML data-type,
3450 for example `<ds:KeyInfo>`. XACML 1.0 supports several ways for comparing such
3451 `<AttributeValue>` elements.

3452 1. In some cases, such an `<AttributeValue>` element MAY be compared using one of the
3453 XACML string functions, such as “`regexp-string-match`”, described below. This requires
3454 that the structured data `<AttributeValue>` be given the
3455 `DataType="http://www.w3.org/2001/XMLSchema#string"`. For example, a structured data-
3456 type that is actually a `ds:KeyInfo/KeyName` would appear in the Context as:

```
3457 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3458   &lt;ds:KeyName&gt;jhibbert-key&lt;/ds:KeyName&gt;  
3459 </AttributeValue>
```

3460 In general, this method will not be adequate unless the structured data-type is quite simple.

3461 2. An `<AttributeSelector>` element MAY be used to select the value of a leaf sub-
3462 element of the structured data-type by means of an XPath expression. That value MAY
3463 then be compared using one of the supported XACML functions appropriate for its primitive
3464 data-type. This method requires support by the **PDP** for the optional XPath expressions
3465 feature.

3466 3. An `<AttributeSelector>` element MAY be used to select the value of any node in the
3467 structured data-type by means of an XPath expression. This node MAY then be compared
3468 using one of the XPath-based functions described in Section A14.13. This method requires
3469 support by the **PDP** for the optional XPath expressions and XPath functions features.

3470 A.4. Representations

3471 An XACML **PDP** SHALL be capable of converting string representations into various primitive data-
3472 types. For integers and doubles, XACML SHALL use the conversions described in [IEEE754].

3473 This document combines the various standards set forth by IEEE and ANSI for string
3474 representation of numeric values.

3475 XACML defines two additional data-types; these are “`urn:oasis:names:tc:xacml:1.0:data-
3476 type:x500Name`” and “`urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name`”. These types
3477 represent identifiers for **subjects** and appear in several standard applications, such as TLS/SSL
3478 and electronic mail.

3479 The “`urn:oasis:names:tc:xacml:1.0:data-type:x500Name`” primitive type represents an X.500
3480 Distinguished Name. The string representation of an X.500 distinguished name is specified in IETF
3481 RFC 2253 “Lightweight Directory Access Protocol (v3): UTF-8 String Representation of
3482 Distinguished Names”.¹

3483 The “`urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name`” primitive type represents electronic mail
3484 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.

3485 An RFC822 name consists of a *local-part* followed by "@" followed by a *domain-part*. The *local-*
3486 *part* is case-sensitive, while the *domain-part* (which is usually a DNS host name) is not case-
3487 sensitive.²

3488 A.5. Bags

3489 XACML defines implicit collections of its primitive types. XACML refers to a collection of values that
3490 are of a single primitive type as a **bag**. **Bags** of primitive types are needed because selections of
3491 nodes from an XML **resource** or XACML request **context** may return more than one value.

3492 The <AttributeSelector> element uses an XPath expression to specify the selection of data
3493 from an XML **resource**. The result of an XPath expression is termed a *node-set*, which contains all
3494 the leaf nodes from the XML **resource** that match the predicate in the XPath expression. Based on
3495 the various indexing functions provided in the XPath specification, it SHALL be implied that a
3496 resultant node-set is the collection of the matching nodes. XACML also defines the
3497 <AttributeDesignator> **element** to have the same matching methodology for attributes in the
3498 XACML request **context**.

3499 The values in a **bag** are not ordered, and some of the values may be duplicates. There SHALL be
3500 no notion of a **bag** containing **bags**, or a **bag** containing values of differing types. I.e. a **bag** in
3501 XACML SHALL contain only values that are of the same primitive type.

3502 A.6. Expressions

3503 XACML specifies expressions in terms of the following elements, of which the <Apply> and
3504 <Condition> elements recursively compose greater expressions. Valid expressions shall be type
3505 correct, which means that the types of each of the elements contained within <Apply> and
3506 <Condition> elements shall agree with the respective argument types of the function that is
3507 named by the `FunctionId` attribute. The resultant type of the <Apply> or <Condition>
3508 element shall be the resultant type of the function, which may be narrowed to a primitive data-type,
3509 or a bag of a primitive data-type, by type-unification. XACML defines an evaluation result of
3510 "Indeterminate", which is said to be the result of an invalid expression, or an operational error
3511 occurring during the evaluation of the expression.

3512 XACML defines the following elements to be legal XACML expressions:

- 3513 • <AttributeValue>
- 3514 • <SubjectAttributeDesignator>
- 3515 • <SubjectAttributeSelector>
- 3516 • <ResourceAttributeDesignator>
- 3517 • <ActionAttributeDesignator>
- 3518 • <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.

- 3519 • <AttributeSelector>
- 3520 • <Apply>
- 3521 • <Condition>
- 3522 • <Function>

3523 A.7. Element <AttributeValue>

3524 The <AttributeValue> element SHALL represent an explicit value of a primitive type. For
3525 example:

```
3526 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-equal">
3527   <AttributeValue
3528     DataType="http://www.w3.org/2001/XMLSchema#integer">123</AttributeValue>
3529   <AttributeValue
3530     DataType="http://www.w3.org/2001/XMLSchema#integer">123</AttributeValue>
3531 </Apply>
```

3532 A.8. Elements <AttributeDesignator> and 3533 <AttributeSelector>

3534 The <AttributeDesignator> and <AttributeSelector> elements SHALL evaluate to a **bag**
3535 of a specific primitive type. The type SHALL be inferred from the function in which it appears. Each
3536 element SHALL contain a URI or XPath expression, respectively, to identify the required **attribute**
3537 values. If an operational error were to occur while finding the values, the value of the element
3538 SHALL be set to "Indeterminate". If the required **attribute** cannot be located, then the value of the
3539 element SHALL be set to an empty **bag** of the inferred primitive type.

3540 A.9. Element <Apply>

3541 XACML function calls are represented by the <Apply> element. The function to be applied is
3542 named in the `FunctionId` attribute of this element. The value of the <Apply> element SHALL be
3543 set to either a primitive data-type or a **bag** of a primitive type, whose data-type SHALL be inferred
3544 from the `FunctionId`. The arguments of a function SHALL be the values of the XACML
3545 expressions that are contained as ordered elements in an <Apply> element. The legal number of
3546 arguments within an <Apply> element SHALL depend upon the `functionId`.

3547 A.10. Element <Condition>

3548 The <Condition> element MAY appear in the <Rule> element as the premise for emitting the
3549 corresponding **effect** of the **rule**. The <Condition> element has the same structure as the
3550 <Apply> element, with the restriction that its result SHALL be of data-type
3551 "http://www.w3.org/2001/XMLSchema#boolean". The evaluation of the <Condition> element
3552 SHALL follow the same evaluation semantics as those of the <Apply> element.

3553 A.11.Element <Function>

3554 The <Function> element names a standard XACML function or an extension function in its
3555 FunctionId attribute. The <Function> element MAY be used as an argument in functions that
3556 take a function as an argument.

3557 A.12.Matching elements

3558 Matching elements appear in the <Target> element of *rules*, *policies* and *policy sets*. They are
3559 the following:

3560 <SubjectMatch>

3561 <ResourceMatch>

3562 <ActionMatch>

3563 These elements represent boolean expressions over attributes of the subject, resource, and action,
3564 respectively. A matching element contains a MatchId attribute that specifies the function to be
3565 used in performing the match evaluation, an **attribute value**, and an <AttributeDesignator>
3566 or <AttributeSelector> element that specifies the **attribute** in the **context** that is to be
3567 matched against the specified value.

3568 The MatchId attribute SHALL specify a function that compares two arguments, returning a result
3569 type of "http://www.w3.org/2001/XMLSchema#boolean". The **attribute** value specified in the
3570 matching element SHALL be supplied to the MatchId function as its first argument. An element of
3571 the **bag** returned by the <AttributeDesignator> or <AttributeSelector> element SHALL
3572 be supplied to the MatchId function as its second argument. The data-type of the **attribute** value
3573 SHALL match the data-type of the first argument expected by the MatchId function. The data-type
3574 of the <AttributeDesignator> or <AttributeSelector> element SHALL match the data-
3575 type of the second argument expected by the MatchId function.

3576 The XACML standard functions that meet the requirements for use as a MatchId attribute value
3577 are:

3578 urn:oasis:names:tc:xacml:1.0:function:-type-equal

3579 urn:oasis:names:tc:xacml:1.0:function:-type-greater-than

3580 urn:oasis:names:tc:xacml:1.0:function:-type-greater-than-or-equal

3581 urn:oasis:names:tc:xacml:1.0:function:-type-less-than

3582 urn:oasis:names:tc:xacml:1.0:function:-type-less-than-or-equal

3583 urn:oasis:names:tc:xacml:1.0:function:-type-match

3584 In addition, functions that are strictly within an extension to XACML MAY appear as a value for the
3585 MatchId attribute, and those functions MAY use data-types that are also extensions, so long as
3586 the extension function returns a boolean result and takes an **attribute** value as its first argument
3587 and an <AttributeDesignator> or <AttributeSelector> as its second argument. The
3588 function used as the value for the MatchId attribute SHOULD be easily indexable. Use of non-
3589 indexable or complex functions may prevent efficient evaluation of **decision requests**.

3590 The evaluation semantics for a matching element is as follows. If an operational error were to
3591 occur while evaluating the <AttributeDesignator> or <AttributeSelector> element, then

3592 the result of the entire expression SHALL be "Indeterminate". If the <AttributeDesignator> or
3593 <AttributeSelector> element were to evaluate to an empty **bag**, then the result of the
3594 expression SHALL be "False". Otherwise, the MatchId function SHALL be applied between the
3595 explicit **attribute** value and each element of the **bag** returned from the <AttributeDesignator>
3596 or <AttributeSelector> element. If at least one of those function applications were to evaluate
3597 to "True", then the result of the entire expression SHALL be "True". Otherwise, if at least one of the
3598 function applications results in "Indeterminate", then the result SHALL be "Indeterminate". Finally,
3599 only if all function applications evaluate to "False", the result of the entire expression SHALL be
3600 "False".

3601 It is possible to express the semantics of a **target** matching element in a **condition**. For instance,
3602 the **target** match expression that compares a "subject-name" starting with the name "John" can be
3603 expressed as follows:

```
3604 <SubjectMatch  
3605   MatchId="urn:oasis:names:tc:xacml:1.0:function:regex-string-match">  
3606   <SubjectAttributeDesignator  
3607     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3608     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3609   <AttributeValue  
3610     DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>  
3611 </SubjectMatch>
```

3612 Alternatively, the same match semantics can be expressed as an <Apply> element in a **condition**
3613 by using the "urn:oasis:names:tc:xacml:1.0:function:any-of" function, as follows:

```
3614 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">  
3615   <Function  
3616     FunctionId="urn:oasis:names:tc:xacml:1.0:function:regex-string-match"/>  
3617   <AttributeValue  
3618     DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>  
3619   <SubjectAttributeDesignator  
3620     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3621     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3622 </Apply>
```

3623

3624 This expression of the semantics is NOT normative.

3625 A.13.Arithmetic evaluation

3626 IEEE 754 [IEEE 754] specifies how to evaluate arithmetic functions in a context, which specifies
3627 defaults for precision, rounding, etc. XACML SHALL use this specification for the evaluation of all
3628 integer and double functions relying on the *Extended Default Context*, enhanced with double
3629 precision:

3630 flags - all set to 0

3631 trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the "division-by-zero" trap
3632 enabler, which SHALL be set to 1

3633 precision - is set to the designated double precision

3634 rounding - is set to round-half-even (IEEE 854 §4.1)

3635 A.14.XACML standard functions

3636 XACML specifies the following functions that are prefixed with the
3637 "urn:oasis:names:tc:xacml:1.0:function:" relative name space identifier.

3638 A14.1 Equality predicates

3639 The following functions are the *equality* functions for the various primitive types. Each function for a
3640 particular data-type follows a specified standard convention for that data-type. If an argument of
3641 one of these functions were to evaluate to "Indeterminate", then the function SHALL be set to
3642 "Indeterminate".

- 3643 • string-equal

3644 This function SHALL take two arguments of "http://www.w3.org/2001/XMLSchema#string"
3645 and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". The function
3646 SHALL return "True" if and only if the value of both of its arguments are of equal length and
3647 each string is determined to be equal byte-by-byte according to the function "integer-equal".

- 3648 • boolean-equal

3649 This function SHALL take two arguments of
3650 "http://www.w3.org/2001/XMLSchema#boolean" and SHALL return "True" if and only if both
3651 values are equal.

- 3652 • integer-equal

3653 This function SHALL take two arguments of data-type
3654 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return an
3655 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on
3656 integers according to IEEE 754 [IEEE 754].

- 3657 • double-equal

3658 This function SHALL take two arguments of data-type
3659 "http://www.w3.org/2001/XMLSchema#double" and SHALL return an
3660 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on
3661 doubles according to IEEE 754 [IEEE 754].

- 3662 • date-equal

3663 This function SHALL take two arguments of data-type
3664 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
3665 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
3666 according to the "op:date-equal" function [XF Section 8.3.11].

- 3667 • time-equal

3668 This function SHALL take two arguments of data-type
3669 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
3670 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according
3671 to the "op:time-equal" function [XF Section 8.3.14].

- 3672 • dateTime-equal

3673 This function SHALL take two arguments of data-type
3674 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an

3675 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
3676 according to the "op:dateTime-equal" function [XF Section 8.3.8].

3677 • dayTimeDuration-equal

3678 This function SHALL take two arguments of data-type "http://www.w3.org/TR/2002/WD-
3679 xquery-operators-20020816#dayTimeDuration" and SHALL return an
3680 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
3681 according to the "op:dayTimeDuration-equal" function [XF Section 8.3.5]. Note that the
3682 lexical representation of each argument MUST be converted to a value expressed in
3683 fractional seconds [XF Section 8.2.2].

3684 • yearMonthDuration-equal

3685 This function SHALL take two arguments of data-type "http://www.w3.org/TR/2002/WD-
3686 xquery-operators-20020816#yearMonthDuration" and SHALL return an
3687 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
3688 according to the "op:yearMonthDuration-equal" function [XF Section 8.3.2]. Note that the
3689 lexical representation of each argument MUST be converted to a value expressed in
3690 integer months [XF Section 8.2.1].

3691 • anyURI-equal

3692 This function SHALL take two arguments of data-type
3693 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return an
3694 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
3695 according to the "op:anyURI-equal" function [XF Section 10.2.1].

3696 • x500Name-equal

3697 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-
3698 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It
3699 shall return "True" if and only if each Relative Distinguished Name (RDN) in the two
3700 arguments matches. Two RDNs shall be said to match if and only if the result of the
3701 following operations is "True"³.

- 3702 1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory
3703 Access Protocol (v3): UTF-8 String Representation of Distinguished Names".
- 3704 2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute
3705 ValuePairs in that RDN in ascending order when compared as octet strings
3706 (described in ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").
- 3707 3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key
3708 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section
3709 4.1.2.4 "Issuer".

3710 • rfc822Name-equal

3711 This function SHALL take two arguments of data-type "urn:oasis:names:tc:xacml:1.0:data-
3712 type:rfc822Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean".
3713 This function SHALL determine whether two "urn:oasis:names:tc:xacml:1.0:data-
3714 type:rfc822Name" arguments are equal. An RFC822 name consists of a *local-part* followed
3715 by "@" followed by a *domain-part*. The *local-part* is case-sensitive, while the *domain-part*
3716 (which is usually a DNS host name) is not case-sensitive. Perform the following
3717 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.

- 3718 1. Normalize the *domain*-part of each argument to lower case
- 3719 2. Compare the expressions by applying the function
3720 “urn:oasis:names:tc:xacml:1.0:function:string-equal” to the normalized arguments.
- 3721 • hexBinary-equal
- 3722 This function SHALL take two arguments of data-type
3723 “http://www.w3.org/2001/XMLSchema#hexBinary” and SHALL return an
3724 “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL return "True" if the
3725 octet sequences represented by the value of both arguments have equal length and are
3726 equal in a conjunctive, point-wise, comparison using the
3727 “urn:oasis:names:tc:xacml:1.0:function:integer-equal”. The conversion from the string
3728 representation to an octet sequence SHALL be as specified in [XS Section 8.2.15]
- 3729 • base64Binary-equal
- 3730 This function SHALL take two arguments of data-type
3731 “http://www.w3.org/2001/XMLSchema#base64Binary” and SHALL return an
3732 “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL return "True" if the
3733 octet sequences represented by the value of both arguments have equal length and are
3734 equal in a conjunctive, point-wise, comparison using the
3735 “urn:oasis:names:tc:xacml:1.0:function:integer-equal”. The conversion from the string
3736 representation to an octet sequence SHALL be as specified in [XS Section 8.2.16]

3737 **A14.2 Arithmetic functions**

3738 All of the following functions SHALL take two arguments of the specified *data-type*, integer or
3739 double, and SHALL return an element of integer or double data-type, respectively. However, the
3740 “add” functions MAY take more than two arguments. Each function evaluation SHALL proceed as
3741 specified by their logical counterparts in IEEE 754 [IEEE 754]. In an expression that contains any
3742 of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3743 "Indeterminate". In the case of the divide functions, if the divisor is zero, then the function SHALL
3744 evaluate to "Indeterminate".

- 3745 • integer-add
- 3746 This function MAY have two or more arguments.
- 3747 • double-add
- 3748 This function MAY have two or more arguments.
- 3749 • integer-subtract
- 3750 • double-subtract
- 3751 • integer-multiply
- 3752 • double-multiply
- 3753 • integer-divide
- 3754 • double-divide
- 3755 • integer-mod

3756 The following functions SHALL take a single argument of the specified *data-type*. The round and
3757 floor functions SHALL take a single argument of data-type
3758 “http://www.w3.org/2001/XMLSchema#double” and return data-type

3759 “http://www.w3.org/2001/XMLSchema#double”. In an expression that contains any of these
3760 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3761 "Indeterminate".

3762 • integer-abs

3763 • double-abs

3764 • round

3765 • floor

3766 **A14.3 String conversion functions**

3767 The following functions convert between values of the XACML
3768 “http://www.w3.org/2001/XMLSchema#string” primitive types. In an expression that contains any of
3769 these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3770 "Indeterminate".

3771 • string-normalize-space

3772 This function SHALL take one argument of data-type
3773 “http://www.w3.org/2001/XMLSchema#string” and SHALL normalize the value by stripping
3774 off all leading and trailing whitespace characters.

3775 • string-normalize-to-lower-case

3776 This function SHALL take one argument of “http://www.w3.org/2001/XMLSchema#string”
3777 and SHALL normalize the value by converting each upper case character to its lower case
3778 equivalent.

3779 **A14.4 Numeric data-type conversion functions**

3780 The following functions convert between the XACML
3781 “http://www.w3.org/2001/XMLSchema#integer” and “http://www.w3.org/2001/XMLSchema#double”
3782 primitive types. In any expression in which the functions defined below are applied, if any argument
3783 while being evaluated results in "Indeterminate", the expression SHALL return "Indeterminate".

3784 • double-to-integer

3785 This function SHALL take one argument of data-type
3786 “http://www.w3.org/2001/XMLSchema#double” and SHALL truncate its numeric value to a
3787 whole number and return an element of data-type
3788 “http://www.w3.org/2001/XMLSchema#integer”.

3789 • integer-to-double

3790 This function SHALL take one argument of data-type
3791 “http://www.w3.org/2001/XMLSchema#integer” and SHALL promote its value to an element
3792 of data-type “http://www.w3.org/2001/XMLSchema#double” of the same numeric value.

3793 **A14.5 Logical functions**

3794 This section contains the specification for logical functions that operate on arguments of the
3795 “http://www.w3.org/2001/XMLSchema#boolean” data-type.

3796

3797 • or
3798 This function SHALL return "False" if it has no arguments and SHALL return "True" if one of
3799 its arguments evaluates to "True". The order of evaluation SHALL be from first argument to
3800 last. The evaluation SHALL stop with a result of "True" if any argument evaluates to "True",
3801 leaving the rest of the arguments unevaluated. In an expression that contains any of these
3802 functions, if ANY argument to this function evaluates to "Indeterminate", then the
3803 expression SHALL evaluate to "Indeterminate".

3804 • and
3805 This function SHALL return "True" if it has no arguments and SHALL return "False" if one of
3806 its arguments evaluates to "False". The order of evaluation SHALL be from first argument
3807 to last. The evaluation SHALL stop with a result of "False" if any argument evaluates to
3808 "False", leaving the rest of the arguments unevaluated. In an expression that contains any
3809 of these functions, if ANY argument to this function evaluates to "Indeterminate", then the
3810 expression SHALL evaluate to "Indeterminate".

3811 • n-of
3812 The first argument to this function SHALL be of data-type
3813 "http://www.w3.org/2001/XMLSchema#integer", specifying the number of the remaining
3814 arguments that MUST evaluate to "True" for the expression to be considered "True". If the
3815 first argument is 0, the result SHALL be "True". If the number of arguments after the first
3816 one is less than the value of the first argument, then the expression SHALL result in
3817 "Indeterminate". The order of evaluation SHALL be: first evaluate the integer value, then
3818 evaluate each subsequent argument. The evaluation SHALL stop and return "True" if the
3819 specified number of arguments evaluate to "True". The evaluation of arguments SHALL
3820 stop if it is determined that evaluating the remaining arguments will not satisfy the
3821 requirement. In an expression that contains any of these functions, if ANY argument to this
3822 function evaluates to "Indeterminate", then the expression SHALL evaluate to
3823 "Indeterminate".

3824 • not
3825 This function SHALL take one logical argument. If the argument evaluates to "True", then
3826 the result of the expression SHALL be "False". If the argument evaluates to "False", then
3827 the result of the expression SHALL be "True". In an expression that contains any of these
3828 functions, if ANY argument to this function evaluates to "Indeterminate", then the
3829 expression SHALL evaluate to "Indeterminate".

3830 Note: For an expression that is an application of AND, OR, or N-OF, it MAY NOT be necessary to
3831 attempt a full evaluation of each boolean argument to a truth value in order to determine whether
3832 the evaluation of the argument would result in "Indeterminate". Analysis of the argument regarding
3833 its necessary attributes, or other analysis regarding errors, such as "divide-by-zero", may render the
3834 argument error free. Such arguments occurring in the expression in a position after the evaluation is
3835 stated to stop need not be processed.

3836 **A14.6 Arithmetic comparison functions**

3837 These functions form a minimal set for comparing two numbers, yielding a boolean result. They
3838 SHALL comply with the rules governed by IEEE 754 [IEEE 754]. In an expression that contains
3839 any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3840 "Indeterminate".

3841 • integer-greater-than

3842 • integer-greater-than-or-equal

- 3843 • integer-less-than
- 3844 • integer-less-than-or-equal
- 3845 • double-greater-than
- 3846 • double-greater-than-or-equal
- 3847 • double-less-than
- 3848 • double-less-than-or-equal

3849 **A14.7 Date and time arithmetic functions**

3850 These functions perform arithmetic operations with the date and time. In an expression that
 3851 contains any of these functions, if any argument is "Indeterminate", then the expression SHALL
 3852 evaluate to "Indeterminate".

- 3853 • dateTime-add-dayTimeDuration

3854 This function SHALL take two arguments, the first is of data-type
 3855 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is of data-type
 3856 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration". It SHALL
 3857 return a result of "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL
 3858 return the value by adding the second argument to the first argument according to the
 3859 specification of adding durations to date and time [XS Appendix E].

- 3860 • dateTime-add-yearMonthDuration

3861 This function SHALL take two arguments, the first is a
 3862 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a
 3863 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It
 3864 SHALL return a result of "http://www.w3.org/2001/XMLSchema#dateTime". This function
 3865 SHALL return the value by adding the second argument to the first argument according to
 3866 the specification of adding durations to date and time [XS Appendix E].

- 3867 • dateTime-subtract-dayTimeDuration

3868 This function SHALL take two arguments, the first is a
 3869 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a
 3870 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration". It SHALL
 3871 return a result of "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument
 3872 is a positive duration, then this function SHALL return the value by adding the
 3873 corresponding negative duration, as per the specification [XS Appendix E]. If the second
 3874 argument is a negative duration, then the result SHALL be as if the function
 3875 "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration" had been applied
 3876 to the corresponding positive duration.

- 3877 • dateTime-subtract-yearMonthDuration

3878 This function SHALL take two arguments, the first is a
 3879 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a
 3880 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It
 3881 SHALL return a result of "http://www.w3.org/2001/XMLSchema#dateTime". If the second
 3882 argument is a positive duration, then this function SHALL return the value by adding the
 3883 corresponding negative duration, as per the specification [XS Appendix E]. If the second
 3884 argument is a negative duration, then the result SHALL be as if the function
 3885 "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration" had been
 3886 applied to the corresponding positive duration.

3887 • date-add-yearMonthDuration
3888 This function SHALL take two arguments, the first is a
3889 “http://www.w3.org/2001/XMLSchema#date” and the second is a
3890 “http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration”. It
3891 return a result of “http://www.w3.org/2001/XMLSchema#date”. This function SHALL return
3892 the value by adding the second argument to the first argument according to the
3893 specification of adding durations to date [XS Appendix E].

3894 • date-subtract-yearMonthDuration
3895 This function SHALL take two arguments, the first is a
3896 “http://www.w3.org/2001/XMLSchema#date” and the second is a
3897 “http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration”. It
3898 SHALL return a result of “http://www.w3.org/2001/XMLSchema#date”. If the second
3899 argument is a positive duration, then this function SHALL return the value by adding the
3900 corresponding negative duration, as per the specification [XS Appendix E]. If the second
3901 argument is a negative duration, then the result SHALL be as if the function
3902 “urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration” had been applied to
3903 the corresponding positive duration.

3904 **A14.8 Non-numeric comparison functions**

3905 These functions perform comparison operations on two arguments of non-numerical types. In an
3906 expression that contains any of these functions, if any argument is "Indeterminate", then the
3907 expression SHALL evaluate to "Indeterminate".

3908 • string-greater-than
3909 This function SHALL take two arguments of data-type
3910 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
3911 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if and only if the
3912 arguments are compared byte by byte and, after an initial prefix of corresponding bytes
3913 from both arguments that are considered equal by
3914 “urn:oasis:names:tc:xacml:1.0:function:integer-equal”, the next byte by byte comparison is
3915 such that the byte from the first argument is greater than the byte from the second
3916 argument by the use of the function “urn:oasis:names:tc:xacml:1.0:function:integer-equal”.

3917 • string-greater-than-or-equal
3918 This function SHALL take two arguments of data-type
3919 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
3920 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated
3921 with the logical function “urn:oasis:names:tc:xacml:1.0:function:or” with two arguments
3922 containing the functions “urn:oasis:names:tc:xacml:1.0:function:string-greater-than” and
3923 “urn:oasis:names:tc:xacml:1.0:function:string-equal” containing the original arguments

3924 • string-less-than
3925 This function SHALL take two arguments of data-type
3926 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
3927 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if and only if the
3928 arguments are compared byte by byte and, after an initial prefix of corresponding bytes
3929 from both arguments are considered equal by
3930 “urn:oasis:names:tc:xacml:1.0:function:integer-equal”, the next byte by byte comparison is
3931 such that the byte from the first argument is less than the byte from the second argument
3932 by the use of the function “urn:oasis:names:tc:xacml:1.0:function:integer-less-than”.

- 3933 • string-less-than-or-equal
 - 3934 This function SHALL take two arguments of data-type
 - 3935 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
 - 3936 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated
 - 3937 with the function “urn:oasis:names:tc:xacml:1.0:function:or” with two arguments containing
 - 3938 the functions “urn:oasis:names:tc:xacml:1.0:function:string-less-than” and
 - 3939 “urn:oasis:names:tc:xacml:1.0:function:string-equal” containing the original arguments.
- 3940 • time-greater-than
 - 3941 This function SHALL take two arguments of data-type
 - 3942 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an
 - 3943 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
 - 3944 argument is greater than the second argument according to the order relation specified for
 - 3945 “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].
- 3946 • time-greater-than-or-equal
 - 3947 This function SHALL take two arguments of data-type
 - 3948 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an
 - 3949 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
 - 3950 argument is greater than or equal to the second argument according to the order relation
 - 3951 specified for “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].
- 3952 • time-less-than
 - 3953 This function SHALL take two arguments of data-type
 - 3954 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an
 - 3955 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
 - 3956 argument is less than the second argument according to the order relation specified for
 - 3957 “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].
- 3958 • time-less-than-or-equal
 - 3959 This function SHALL take two arguments of data-type
 - 3960 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an
 - 3961 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
 - 3962 argument is less than or equal to the second argument according to the order relation
 - 3963 specified for “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].
- 3964 • dateTime-greater-than
 - 3965 This function SHALL take two arguments of data-type
 - 3966 “http://www.w3.org/2001/XMLSchema#dateTime” and SHALL return an
 - 3967 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
 - 3968 argument is greater than the second argument according to the order relation specified for
 - 3969 “http://www.w3.org/2001/XMLSchema#dateTime” [XS Section 3.2.7].
- 3970 • dateTime-greater-than-or-equal
 - 3971 This function SHALL take two arguments of data-type
 - 3972 “http://www.w3.org/2001/XMLSchema#dateTime” and SHALL return an
 - 3973 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
 - 3974 argument is greater than or equal to the second argument according to the order relation
 - 3975 specified for “http://www.w3.org/2001/XMLSchema#dateTime” [XS Section 3.2.7].
- 3976 • dateTime-less-than

3977 This function SHALL take two arguments of data-type
3978 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
3979 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3980 argument is less than the second argument according to the order relation specified for
3981 "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

3982

- dateTime-less-than-or-equal

3984 This function SHALL take two arguments of data-type
3985 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
3986 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3987 argument is less than or equal to the second argument according to the order relation
3988 specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

- date-greater-than

3990 This function SHALL take two arguments of data-type
3991 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
3992 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3993 argument is greater than the second argument according to the order relation specified for
3994 "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

- date-greater-than-or-equal

3996 This function SHALL take two arguments of data-type
3997 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
3998 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
3999 argument is greater than or equal to the second argument according to the order relation
4000 specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

- date-less-than

4002 This function SHALL take two arguments of data-type
4003 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4004 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
4005 argument is less than the second argument according to the order relation specified for
4006 "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

- date-less-than-or-equal

4008 This function SHALL take two arguments of data-type
4009 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4010 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
4011 argument is less than or equal to the second argument according to the order relation
4012 specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

4013 A14.9 Bag functions

4014 These functions operate on a *bag* of *type* values, where *data-type* is one of the primitive types. In
4015 an expression that contains any of these functions, if any argument is "Indeterminate", then the
4016 expression SHALL evaluate to "Indeterminate". Some additional conditions defined for each
4017 function below SHALL cause the expression to evaluate to "Indeterminate".

- type-one-and-only

4019 This function SHALL take an argument of a **bag** of *type* values and SHALL return a value
4020 of *data-type*. It SHALL return the only value in the **bag**. If the **bag** does not have one and
4021 only one value, then the expression SHALL evaluate to "Indeterminate".

4022 • *type-bag-size*

4023 This function SHALL take a **bag** of *type* values as an argument and SHALL return an
4024 "http://www.w3.org/2001/XMLSchema#integer" indicating the number of values in the **bag**.

4025

4026

4027 • *type-is-in*

4028 This function SHALL take an argument of data-type *type* as the first argument and a **bag** of
4029 *type* values as the second argument. The expression SHALL evaluate to "True" if the first
4030 argument matches by the "urn:oasis:names:tc:xacml:1.0:function:type-equal" to any value
4031 in the **bag**.

4032 • *type-bag*

4033 This function SHALL take any number of arguments of a single data-type and return a **bag**
4034 of *type* values containing the values of the arguments. An application of this function to
4035 zero arguments SHALL produce an empty **bag** of the specified data-type.

4036 **A14.10 Set functions**

4037 These functions operate on **bags** mimicking sets by eliminating duplicate elements from a **bag**. In
4038 an expression that contains any of these functions, if any argument is "Indeterminate", then the
4039 expression SHALL evaluate to "Indeterminate".

4040 • *type-intersection*

4041 This function SHALL take two arguments that are both a **bag** of *type* values. The
4042 expression SHALL return a **bag** of *type* values such that it contains only elements that are
4043 common between the two **bags**, which is determined by
4044 "urn:oasis:names:tc:xacml:1.0:function:type-equal". No duplicates as determined by
4045 "urn:oasis:names:tc:xacml:1.0:function:type-equal" SHALL exist in the result.

4046 • *type-at-least-one-member-of*

4047 This function SHALL take two arguments that are both a **bag** of *type* values. The
4048 expression SHALL evaluate to "True" if at least one element of the first argument is
4049 contained in the second argument as determined by
4050 "urn:oasis:names:tc:xacml:1.0:function:type-is-in".

4051 • *type-union*

4052 This function SHALL take two arguments that are both a **bag** of *type* values. The
4053 expression SHALL return a **bag** of *type* such that it contains all elements of both **bags**. No
4054 duplicates as determined by "urn:oasis:names:tc:xacml:1.0:function:type-equal" SHALL
4055 exist in the result.

4056 • *type-subset*

4057 This function SHALL take two arguments that are both a **bag** of *type* values. It SHALL
4058 return "True" if the first argument is a subset of the second argument. Each argument is
4059 considered to have its duplicates removed as determined by
4060 "urn:oasis:names:tc:xacml:1.0:function:type-equal" before subset calculation.

- 4061 • *type-set-equals*
- 4062 This function SHALL take two arguments that are both a *bag* of *type* values and SHALL
- 4063 return the result of applying "urn:oasis:names:tc:xacml:1.0:function:and" to the application
- 4064 of "urn:oasis:names:tc:xacml:1.0:function:type-subset" to the first and second arguments
- 4065 and the application of "urn:oasis:names:tc:xacml:1.0:function:type-subset" to the second
- 4066 and first arguments.

4067 **A14.11 Higher-order bag functions**

4068 This section describes functions in XACML that perform operations on *bags* such that functions
4069 may be applied to the *bags* in general.

4070 In this section, a general-purpose functional language called Haskell [**Haskell**] is used to formally
4071 specify the semantics of these functions. Although the English description is adequate, a formal
4072 specification of the semantics is helpful.

4073 For a quick summary, in the following Haskell notation, a function definition takes the form of
4074 clauses that are applied to patterns of structures, namely lists. The symbol "[]" denotes the empty
4075 list, whereas the expression "(x:xs)" matches against an argument of a non-empty list of which "x"
4076 represents the first element of the list, and "xs" is the rest of the list, which may be an empty list. We
4077 use the Haskell notion of a list, which is an ordered collection of elements, to model the XACML
4078 *bags* of values.

4079 A simple Haskell definition of a familiar function "urn:oasis:names:tc:xacml:1.0:function:and" that
4080 takes a list of booleans is defined as follows:

```
4081 and:: [Bool] -> Bool
```

```
4082 and [] = "True"
```

```
4083 and (x:xs) = x && (and xs)
```

4084 The first definition line denoted by a "::" formally describes the data-type of the function, which takes
4085 a list of booleans, denoted by "[Bool]", and returns a boolean, denoted by "Bool". The second
4086 definition line is a clause that states that the function "and" applied to the empty list is "True". The
4087 second definition line is a clause that states that for a non-empty list, such that the first element is
4088 "x", which is a value of data-type Bool, the function "and" applied to x SHALL be combined with,
4089 using the logical conjunction function, which is denoted by the infix symbol "&&", the result of
4090 recursively applying the function "and" to the rest of the list. Of course, an application of the "and"
4091 function is "True" if and only if the list to which it is applied is empty or every element of the list is
4092 "True". For example, the evaluation of the following Haskell expressions,

```
4093 (and []), (and ["True"]), (and ["True","True"]), (and ["True","True","False"])
```

4094 evaluate to "True", "True", "True", and "False", respectively.

4095 In an expression that contains any of these functions, if any argument is "Indeterminate", then the
4096 expression SHALL evaluate to "Indeterminate".

- 4097 • *any-of*

4098 This function applies a boolean function between a specific primitive value and a *bag* of
4099 values, and SHALL return "True" if and only if the predicate is "True" for at least one
4100 element of the *bag*.

4101 This function SHALL take three arguments. The first argument SHALL be a <Function>
4102 element that names a boolean function that takes two arguments of primitive types. The
4103 second argument SHALL be a value of a primitive data-type. The third argument SHALL

4104 be a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function
4105 named in the <Function> element is applied to the second argument and each element
4106 of the third argument (the **bag**) and the results are combined with
4107 “urn:oasis:names:tc:xacml:1.0:function:or”.

4108 In Haskell, the semantics of this operation are as follows:

```
4109 any_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
4110 any_of f a [] = "False"
4111 any_of f a (x:xs) = (f a x) || (any_of f a xs)
```

4112 In the above notation, “f” is the function name to be applied, “a” is the primitive value, and
4113 “(x:xs)” represents the first element of the list as “x” and the rest of the list as “xs”.

4114 For example, the following expression SHALL return "True":

```
4115 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
4116 <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
4117 <AttributeValue
4118 DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4119 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4120 <AttributeValue
4121 DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4122 <AttributeValue
4123 DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4124 <AttributeValue
4125 DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4126 <AttributeValue
4127 DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4128 </Apply>
4129 </Apply>
```

4130 This expression is "True" because the first argument is equal to at least one of the
4131 elements of the **bag**.

4132 • all-of

4133 This function applies a boolean function between a specific primitive value and a **bag** of
4134 values, and returns "True" if and only if the predicate is "True" for every element of the **bag**.

4135 This function SHALL take three arguments. The first argument SHALL be a <Function>
4136 element that names a boolean function that takes two arguments of primitive types. The
4137 second argument SHALL be a value of a primitive data-type. The third argument SHALL
4138 be a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function
4139 named in the <Function> element were applied to the second argument and each
4140 element of the third argument (the **bag**) and the results were combined using
4141 “urn:oasis:names:tc:xacml:1.0:function:and”.

4142 In Haskell, the semantics of this operation are as follows:

```
4143 all_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
4144 all_of f a [] = "False"
4145 all_of f a (x:xs) = (f a x) && (all_of f a xs)
```

4146 In the above notation, “f” is the function name to be applied, “a” is the primitive value, and
4147 “(x:xs)” represents the first element of the list as “x” and the rest of the list as “xs”.

4148 For example, the following expression SHALL evaluate to "True":

```

4149 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">
4150   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4151   greater"/>
4152   <AttributeValue
4153   DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4154   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4155     <AttributeValue
4156     DataType="http://www.w3.org/2001/XMLSchema#integer">9</AttributeValue>
4157     <AttributeValue
4158     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4159     <AttributeValue
4160     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4161     <AttributeValue
4162     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4163   </Apply>
4164 </Apply>

```

4165 This expression is "True" because the first argument is greater than *all* of the elements of
4166 the **bag**.

4167 • any-of-any

4168 This function applies a boolean function between each element of a **bag** of values and
4169 each element of another **bag** of values, and returns "True" if and only if the predicate is
4170 "True" for at least one comparison.

4171 This function SHALL take three arguments. The first argument SHALL be a <Function>
4172 element that names a boolean function that takes two arguments of primitive types. The
4173 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be
4174 a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function
4175 named in the <Function> element were applied between *every* element in the second
4176 argument and *every* element of the third argument (the **bag**) and the results were
4177 combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics are that the
4178 result of the expression SHALL be "True" if and only if the applied predicate is "True" for
4179 *any* comparison of elements from the two **bags**.

4180 In Haskell, taking advantage of the "any_of" function defined above, the semantics of the
4181 "any_of_any" function are as follows:

```

4182 any_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4183 any_of_any f [] ys = "False"
4184 any_of_any f (x:xs) ys = (any_of f x ys) || (any_of_any f xs ys)

```

4185 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4186 element of the list as "x" and the rest of the list as "xs".

4187 For example, the following expression SHALL evaluate to "True":


```

4188 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">
4189   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
4190   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4191     <AttributeValue
4192       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4193     <AttributeValue
4194       DataType="http://www.w3.org/2001/XMLSchema#string">Mary</AttributeValue>
4195     </Apply>
4196     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4197       <AttributeValue
4198         DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4199       <AttributeValue
4200         DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4201       <AttributeValue
4202         DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4203       <AttributeValue
4204         DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4205     </Apply>
4206 </Apply>

```

4207 This expression is "True" because at least one of the elements of the first **bag**, namely
4208 "Ringo", is equal to at least one of the string values of the second **bag**.

4209 • all-of-any

4210 This function applies a boolean function between the elements of two **bags**. The
4211 expression is "True" if and only if the predicate is "True" between each and all of the
4212 elements of the first **bag** collectively against at least one element of the second **bag**.

4213 This function SHALL take three arguments. The first argument SHALL be a <Function>
4214 element that names a boolean function that takes two arguments of primitive types. The
4215 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be
4216 a **bag** of a primitive data-type. The expression SHALL be evaluated as if function named in
4217 the <Function> element were applied between every element in the second argument
4218 and every element of the third argument (the **bag**) using
4219 "urn:oasis:names:tc:xacml:1.0:function:and". The semantics are that the result of the
4220 expression SHALL be "True" if and only if the applied predicate is "True" for each element
4221 of the first **bag** and any element of the second **bag**.

4222 In Haskell, taking advantage of the "any_of" function defined in Haskell above, the
4223 semantics of the "all_of_any" function are as follows:

```

4224               all_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4225               all_of_any f []     ys = "False"
4226               all_of_any f (x:xs) ys = (any_of f x ys) && (all_of_any f xs ys)

```

4227 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4228 element of the list as "x" and the rest of the list as "xs".

4229 For example, the following expression SHALL evaluate to "True":

```

4230 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-any">
4231   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4232   greater"/>
4233   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4234     <AttributeValue
4235     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4236     <AttributeValue
4237     DataType="http://www.w3.org/2001/XMLSchema#integer">20</AttributeValue>
4238   </Apply>
4239   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4240     <AttributeValue
4241     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4242     <AttributeValue
4243     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4244     <AttributeValue
4245     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4246     <AttributeValue
4247     DataType="http://www.w3.org/2001/XMLSchema#integer">21</AttributeValue>
4248   </Apply>
4249 </Apply>

```

4250 This expression is "True" because all of the elements of the first **bag**, each "10" and "20",
4251 are greater than at least one of the integer values "1", "3", "5", "21" of the second **bag**.

4252 • any-of-all

4253 This function applies a boolean function between the elements of two **bags**. The
4254 expression SHALL be "True" if and only if the predicate is "True" between at least one of
4255 the elements of the first **bag** collectively against all the elements of the second **bag**.

4256 This function SHALL take three arguments. The first argument SHALL be a <Function>
4257 element that names a boolean function that takes two arguments of primitive types. The
4258 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be
4259 a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function
4260 named in the <Function> element were applied between *every* element in the second
4261 argument and *every* element of the third argument (the **bag**) and the results were
4262 combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics are that the
4263 result of the expression SHALL be "True" if and only if the applied predicate is "True" for
4264 *any* element of the first **bag** compared to *all* the elements of the second **bag**.

4265 In Haskell, taking advantage of the "all_of" function defined in Haskell above, the semantics
4266 of the "any_of_all" function are as follows:

```

4267 any_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ]-> Bool
4268 any_of_all f [] ys = "False"
4269 any_of_all f (x:xs) ys = (all_of f x ys) || ( any_of_all f xs ys)

```

4270 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4271 element of the list as "x" and the rest of the list as "xs".

4272 For example, the following expression SHALL evaluate to "True":

```

4273 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-all">
4274   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4275   greater"/>
4276   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4277     <AttributeValue
4278     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4279     <AttributeValue
4280     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4281   </Apply>
4282   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4283     <AttributeValue
4284     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4285     <AttributeValue
4286     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4287     <AttributeValue
4288     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4289     <AttributeValue
4290     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4291   </Apply>
4292 </Apply>

```

4293 This expression is "True" because at least one element of the first **bag**, namely "5", is
4294 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4295 • all-of-all

4296 This function applies a boolean function between the elements of two **bags**. The
4297 expression SHALL be "True" if and only if the predicate is "True" between each and all of
4298 the elements of the first **bag** collectively against all the elements of the second **bag**.

4299 This function SHALL take three arguments. The first argument SHALL be a <Function>
4300 element that names a boolean function that takes two arguments of primitive types. The
4301 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be
4302 a **bag** of a primitive data-type. The expression is evaluated as if the function named in the
4303 <Function> element were applied between *every* element in the second argument and
4304 *every* element of the third argument (the **bag**) and the results were combined using
4305 "urn:oasis:names:tc:xacml:1.0:function:and". The semantics are that the result of the
4306 expression is "True" if and only if the applied predicate is "True" for *all* elements of the first
4307 **bag** compared to *all* the elements of the second **bag**.

4308 In Haskell, taking advantage of the "all_of" function defined in Haskell above, the semantics
4309 of the "all_of_all" function is as follows:

```

4310 all_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4311 all_of_all f [] ys = "False"
4312 all_of_all f (x:xs) ys = (all_of f x ys) && (all_of_all f xs ys)

```

4313 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4314 element of the list as "x" and the rest of the list as "xs".

4315 For example, the following expression SHALL evaluate to "True":

```

4316 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-all">
4317   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4318   greater"/>
4319   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4320     <AttributeValue
4321     DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>
4322     <AttributeValue
4323     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4324   </Apply>
4325   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4326     <AttributeValue
4327     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4328     <AttributeValue
4329     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4330     <AttributeValue
4331     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4332     <AttributeValue
4333     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4334   </Apply>
4335 </Apply>

```

4336 This expression is "True" because all elements of the first **bag**, "5" and "6", are each
4337 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4338 • map

4339 This function converts a **bag** of values to another **bag** of values.

4340 This function SHALL take two arguments. The first function SHALL be a <Function>
4341 element naming a function that takes a single argument of a primitive data-type and returns
4342 a value of a primitive data-type. The second argument SHALL be a **bag** of a primitive data-
4343 type. The expression SHALL be evaluated as if the function named in the <Function>
4344 element were applied to each element in the **bag** resulting in a **bag** of the converted value.
4345 The result SHALL be a **bag** of the primitive data-type that is the same data-type that is
4346 returned by the function named in the <Function> element.

4347 In Haskell, this function is defined as follows:

```

4348     map :: (a -> b) -> [a] -> [b]
4349     map f []     = []
4350     map f (x:xs) = (f x) : (map f xs)

```

4351 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4352 element of the list as "x" and the rest of the list as "xs".

4353 For example, the following expression,

```

4354 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:map">
4355   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
4356   normalize-to-lower-case">
4357   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4358     <AttributeValue
4359     DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>
4360     <AttributeValue
4361     DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>
4362   </Apply>
4363 </Apply>

```

4364 evaluates to a **bag** containing "hello" and "world!".

4365

A14.12 Special match functions

4366 These functions operate on various types and evaluate to
4367 "http://www.w3.org/2001/XMLSchema#boolean" based on the specified standard matching
4368 algorithm. In an expression that contains any of these functions, if any argument is "Indeterminate",
4369 then the expression SHALL evaluate to "Indeterminate".

4370 • regex-string-match

4371 This function decides a regular expression match. It SHALL take two arguments of
4372 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4373 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4374 expression and the second argument SHALL be a general string. The function
4375 specification SHALL be that of the "xf:matches" function with the arguments reversed [XF
4376 Section 6.3.15].

4377 • x500Name-match

4378 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-
4379 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It
4380 shall return "True" if and only if the first argument matches some terminal sequence of
4381 RDNs from the second argument when compared using x500Name-equal.

4382 • rfc822Name-match

4383 This function SHALL take two arguments, the first is of data-type
4384 "http://www.w3.org/2001/XMLSchema#string" and the second is of data-type
4385 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an
4386 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if
4387 the first argument matches the second argument according to the following specification.

4388 An RFC822 name consists of a local-part followed by "@" followed by domain-part. The
4389 local-part is case-sensitive, while the domain-part (which is usually a DNS name) is not
4390 case-sensitive.⁴

4391 The second argument contains a complete rfc822Name. The first argument is a complete
4392 or partial rfc822Name used to select appropriate values in the second argument as follows.

4393 In order to match a particular mailbox in the second argument, the first argument must
4394 specify the complete mail address to be matched. For example, if the first argument is
4395 "Anderson@sun.com", this matches a value in the second argument of
4396 "Anderson@sun.com" and "Anderson@SUN.COM", but not "Anne.Anderson@sun.com",
4397 "anderson@sun.com" or "Anderson@east.sun.com".

4398 In order to match any mail address at a particular domain in the second argument, the first
4399 argument must specify only a domain name (usually a DNS name). For example, if the first
4400 argument is "sun.com", this matches a value in the first argument of "Anderson@sun.com"
4401 or "Baxter@SUN.COM", but not "Anderson@east.sun.com".

4402 In order to match any mail address in a particular domain in the second argument, the first
4403 argument must specify the desired domain-part with a leading ".". For example, if the first
4404 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.

4405 "Anderson@east.sun.com" and "anne.anderson@ISRG.EAST.SUN.COM" but not
4406 "Anderson@sun.com".

4407 **A14.13 XPath-based functions**

4408 This section specifies functions that take XPath expressions for arguments. An XPath expression
4409 evaluates to a *node-set*, which is a set of XML nodes that match the expression. A node or node-
4410 set is not in the formal data-type system of XACML. All comparison or other operations on node-
4411 sets are performed in the isolation of the particular function specified. The XPath expressions in
4412 these functions are restricted to the XACML request *context*. The `<xacml-context:Request>`
4413 element is a context node for every XPath expression. The following functions are defined:

- 4414 • xpath-node-count

4415 This function SHALL take an "http://www.w3.org/2001/XMLSchema#string" as an
4416 argument, which SHALL be interpreted as an XPath expression, and evaluates to an
4417 "http://www.w3.org/2001/XMLSchema#integer". The value returned from the function
4418 SHALL be the count of the nodes within the node-set that matches the given XPath
4419 expression.

- 4420 • xpath-node-equal

4421 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,
4422 which SHALL be interpreted as XPath expressions, and SHALL return an
4423 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if any
4424 XML node from the node-set matched by the first argument equals according to the
4425 "op:node-equal" function [XF Section 13.1.6] any XML node from the node-set matched by
4426 the second argument.

- 4427 • xpath-node-match

4428 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments, which
4429 SHALL be interpreted as XPath expressions and SHALL return an
4430 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if
4431 either of the following two conditions is satisfied: (1) Any XML node from the node-set matched
4432 by the first argument is equal according to "op:node-equal" [XF Section 13.1.6] to any XML node
4433 from the node-set matched by the second argument. (2) Any attribute and element node below
4434 any XML node from the node-set matched by the first argument is equal according to "op:node-
4435 equal" [XF Section 13.1.6] to any XML node from the node-set matched by the second
4436 argument.

4437 NOTE: The first condition is equivalent to "xpath-node-equal", and guarantees that "xpath-node-
4438 equal" is a special case of "xpath-node-match".

4439 **A14.14 Extension functions and primitive types**

4440 Functions and primitive types are specified by string identifiers allowing for the introduction of
4441 functions in addition to those specified by XACML. This approach allows one to extend the XACML
4442 module with special functions and special primitive data-types.

4443 In order to preserve some integrity to the XACML evaluation strategy, the result of all function
4444 applications SHALL depend only on the values of its arguments. Global and hidden parameters
4445 SHALL NOT affect the evaluation of an expression. Functions SHALL NOT have side effects, as
4446 evaluation order cannot be guaranteed in a standard way.

4447 Appendix B. XACML identifiers (normative)

4448 This section defines standard identifiers for commonly used entities. All XACML-defined identifiers
4449 have the common base:

4450 `urn:oasis:names:tc:xacml:1.0`

4451 B.1. XACML namespaces

4452 There are currently two defined XACML namespaces.

4453 Policies are defined using this identifier.

4454 `urn:oasis:names:tc:xacml:1.0:policy`

4455 Request and response **contexts** are defined using this identifier.

4456 `urn:oasis:names:tc:xacml:1.0:context`

4457 B.2. Access subject categories

4458 This identifier indicates the system entity that initiated the **access** request. That is, the initial entity
4459 in a request chain. If **subject** category is not specified, this is the default value.

4460 `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

4461 This identifier indicates the system entity that will receive the results of the request. Used when it is
4462 distinct from the access-subject.

4463 `urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject`

4464 This identifier indicates a system entity through which the **access** request was passed. There may
4465 be more than one. No means is provided to specify the order in which they passed the message.

4466 `urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject`

4467 This identifier indicates a system entity associated with a local or remote codebase that generated
4468 the request. Corresponding **subject attributes** might include the URL from which it was loaded
4469 and/or the identity of the code-signer. There may be more than one. No means is provided to
4470 specify the order they processed the request.

4471 `urn:oasis:names:tc:xacml:1.0:subject-category:codebase`

4472 This identifier indicates a system entity associated with the computer that initiated the **access**
4473 request. An example would be an IPsec identity.

4474 `urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine`

4475 B.3. XACML functions

4476 This identifier is the base for all the identifiers in the table of functions. See Section A.1.

4477 `urn:oasis:names:tc:xacml:1.0:function`

4478 B.4. Data-types

4479 The following identifiers indicate useful data-types.

4480 X.500 distinguished name

4481 urn:oasis:names:tc:xacml:1.0:data-type:x500Name

4482 An x500Name contains an ITU-T Rec. X.520 Distinguished Name. The valid syntax for such a
 4483 name is described in IETF RFC 2253 "Lightweight Directory Access Protocol (v3): UTF-8 String
 4484 Representation of Distinguished Names".

4485 RFC822 Name

4486 urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name

4487 An rfc822Name contains an "e-mail name". The valid syntax for such a name is described in IETF
 4488 RFC 2821, Section 4.1.2, Command Argument Syntax, under the term "Mailbox".

4489 The following data-type identifiers are defined by XML Schema.

4490 http://www.w3.org/2001/XMLSchema#string
 4491 http://www.w3.org/2001/XMLSchema#boolean
 4492 http://www.w3.org/2001/XMLSchema#integer
 4493 http://www.w3.org/2001/XMLSchema#double
 4494 http://www.w3.org/2001/XMLSchema#time
 4495 http://www.w3.org/2001/XMLSchema#date
 4496 http://www.w3.org/2001/XMLSchema#dateTime
 4497 http://www.w3.org/2001/XMLSchema#anyURI
 4498 http://www.w3.org/2001/XMLSchema#hexBinary
 4499 http://www.w3.org/2001/XMLSchema#base64Binary

4500 The following data-type identifiers correspond to the dayTimeDuration and yearMonthDuration
 4501 data-types defined in [XF Sections 8.2.2 and 8.2.1, respectively].

4502 http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration
 4503 http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration

4504 B.5. Subject attributes

4505 These identifiers indicate *attributes* of a *subject*. When used, they SHALL appear within a
 4506 <Subject> element of the request *context*. They SHALL be accessed via a
 4507 <SubjectAttributeDesignator> or an <AttributeSelector> element pointing into a
 4508 <Subject> element of the request *context*.

4509 At most one of each of these attributes is associated with each subject. Each attribute associated
 4510 with authentication included within a single <Subject> element relates to the same authentication
 4511 event.

4512 This identifier indicates the name of the *subject*. The default format is
 4513 http://www.w3.org/2001/XMLSchema#string. To indicate other formats, use `DataType` attributes
 4514 listed in B.4

4515 urn:oasis:names:tc:xacml:1.0:subject:subject-id

4516 This identifier indicates the *subject* category. "access-subject" is the default.

4517 urn:oasis:names:tc:xacml:1.0:subject:category

4518 This identifier indicates the security domain of the *subject*. It identifies the administrator and policy
 4519 that manages the name-space in which the *subject* id is administered.

4520 urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier

4521 This identifier indicates a public key used to confirm the *subject's* identity.

4522 urn:oasis:names:tc:xacml:1.0:subject:key-info

4523 This identifier indicates the time at which the *subject* was authenticated.

4524 urn:oasis:names:tc:xacml:1.0:subject:authentication-time

4525 This identifier indicates the method used to authenticate the *subject*.

4526 urn:oasis:names:tc:xacml:1.0:subject:authentication-method

4527 This identifier indicates the time at which the **subject** initiated the **access** request, according to the
4528 **PEP**.

4529 urn:oasis:names:tc:xacml:1.0:subject:request-time

4530 This identifier indicates the time at which the **subject's** current session began, according to the
4531 **PEP**.

4532 urn:oasis:names:tc:xacml:1.0:subject:session-start-time

4533 The following identifiers indicate the location where authentication credentials were activated. They
4534 are intended to support the corresponding entities from the SAML authentication statement.

4535 This identifier indicates that the location is expressed as an IP address.

4536 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address

4537 This identifier indicates that the location is expressed as a DNS name.

4538 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name

4539 Where a suitable attribute is already defined in LDAP [**LDAP-1**, **LDAP-2**], the XACML identifier
4540 SHALL be formed by adding the **attribute** name to the URI of the LDAP specification. For
4541 example, the **attribute** name for the userPassword defined in the rfc2256 SHALL be:

4542 http://www.ietf.org/rfc/rfc2256.txt#userPassword

4543 B.6. Resource attributes

4544 These identifiers indicate **attributes** of the **resource**. When used, they SHALL appear within the
4545 <Resource> element of the request **context**. They SHALL be accessed via a
4546 <ResourceAttributeDesignator> or an <AttributeSelector> element pointing into the
4547 <Resource> element of the request **context**.

4548 This identifier indicates the entire URI of the **resource**.

4549 urn:oasis:names:tc:xacml:1.0:resource:resource-id

4550 A **resource attribute** used to indicate values extracted from the **resource**.

4551 urn:oasis:names:tc:xacml:1.0:resource:resource-content

4552 This identifier indicates the last (rightmost) component of the file name. For example, if the URI is:
4553 "file://home/my/status#pointer", the simple-file-name is "status".

4554 urn:oasis:names:tc:xacml:1.0:resource:simple-file-name

4555 This identifier indicates that the **resource** is specified by an XPath expression.

4556 urn:oasis:names:tc:xacml:1.0:resource:xpath

4557 This identifier indicates a UNIX file-system path.

4558 urn:oasis:names:tc:xacml:1.0:resource:ufs-path

4559 This identifier indicates the scope of the **resource**, as described in Section 7.8.

4560 urn:oasis:names:tc:xacml:1.0:resource:scope

4561 The allowed value for this attribute is of data-type http://www.w3.org/2001/XMLSchema#string, and
4562 is either "Immediate", "Children" or "Descendants".

4563 B.7. Action attributes

4564 These identifiers indicate **attributes** of the **action** being requested. When used, they SHALL
4565 appear within the <Action> element of the request **context**. They SHALL be accessed via an
4566 <ActionAttributeDesignator> or an <AttributeSelector> element pointing into the
4567 <Action> element of the request **context**.

4568 urn:oasis:names:tc:xacml:1.0:action:action-id
4569 Action namespace
4570 urn:oasis:names:tc:xacml:1.0:action:action-namespace
4571 Implied action. This is the value for action-id attribute when action is implied.
4572 urn:oasis:names:tc:xacml:1.0:action:implied-action

4573 B.8. Environment attributes

4574 These identifiers indicate *attributes* of the *environment* within which the *decision request* is to be
4575 evaluated. When used in the *decision request*, they SHALL appear in the <Environment>
4576 element of the request *context*. They SHALL be accessed via an
4577 <EnvironmentAttributeDesignator> or an <AttributeSelector> element pointing into
4578 the <Environment> element of the request *context*.

4579 This identifier indicates the current time at the *PDP*. In practice it is the time at which the request
4580 *context* was created.

4581 urn:oasis:names:tc:xacml:1.0:environment:current-time
4582 urn:oasis:names:tc:xacml:1.0:environment:current-date
4583 urn:oasis:names:tc:xacml:1.0:environment:current-dateTime

4584 B.9. Status codes

4585 The following status code identifiers are defined.

4586 This identifier indicates success.

4587 urn:oasis:names:tc:xacml:1.0:status:ok

4588 This identifier indicates that attributes necessary to make a policy decision were not available.

4589 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

4590 This identifier indicates that some attribute value contained a syntax error, such as a letter in a
4591 numeric field.

4592 urn:oasis:names:tc:xacml:1.0:status:syntax-error

4593 This identifier indicates that an error occurred during policy evaluation. An example would be
4594 division by zero.

4595 urn:oasis:names:tc:xacml:1.0:status:processing-error

4596 B.10. Combining algorithms

4597 The deny-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

4598 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

4599 The deny-overrides policy-combining algorithm has the following value for
4600 policyCombiningAlgId:

4601 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides

4602 The permit-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

4603 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides

4604 The permit-overrides policy-combining algorithm has the following value for
4605 policyCombiningAlgId:

4606 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides

4607 The first-applicable rule-combining algorithm has the following value for ruleCombiningAlgId:
4608 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable
4609 The first-applicable policy-combining algorithm has the following value for
4610 policyCombiningAlgId:
4611 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable
4612 The only-one-applicable-policy policy-combining algorithm has the following value for
4613 policyCombiningAlgId:
4614 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable
4615 The ordered-deny-overrides rule-combining algorithm has the following value for
4616 ruleCombiningAlgId:
4617 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides
4618
4619 The ordered-deny-overrides policy-combining algorithm has the following value for
4620 policyCombiningAlgId:
4621 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-overrides
4622
4623 The ordered-permit-overrides rule-combining algorithm has the following value for
4624 ruleCombiningAlgId:
4625 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-overrides
4626
4627 The ordered-permit-overrides policy-combining algorithm has the following value for
4628 policyCombiningAlgId:
4629 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-overrides

4630

Appendix C. Combining algorithms (normative)

4631

This section contains a description of the rule-combining and policy-combining algorithms specified by XACML.

4632

4633

C.1. Deny-overrides.

4634

The following specification defines the "Deny-overrides" *rule-combining algorithm* of a *policy*.

4635

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".

4636

4637

4638

4639

4640

4641

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.

4642

4643

4644

4645

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".

4646

4647

4648

The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

4649

```
Decision denyOverridesRuleCombiningAlgorithm(Rule rule[])
```

4650

```
{
```

4651

```
    Boolean atLeastOneError = false;
```

4652

```
    Boolean potentialDeny = false;
```

4653

```
    Boolean atLeastOnePermit = false;
```

4654

```
    for( i=0 ; i < lengthOf(rules) ; i++ )
```

4655

```
    {
```

4656

```
        Decision decision = evaluate(rule[i]);
```

4657

```
        if (decision == Deny)
```

4658

```
        {
```

4659

```
            return Deny;
```

4660

```
        }
```

4661

```
        if (decision == Permit)
```

4662

```
        {
```

4663

```
            atLeastOnePermit = true;
```

4664

```
            continue;
```

4665

```
        }
```

4666

```
        if (decision == NotApplicable)
```

4667

```
        {
```

4668

```
            continue;
```

4669

```
        }
```

4670

```
        if (decision == Indeterminate)
```

4671

```
        {
```

4672

```
            atLeastOneError = true;
```

4673

```
        }
```

4674

```
            if (effect(rule[i]) == Deny)
```

4675

```
            {
```

4676

```
                potentialDeny = true;
```

4677

```
            }
```

4678

```
            continue;
```

```

4679     }
4680   }
4681   if (potentialDeny)
4682   {
4683     return Indeterminate;
4684   }
4685   if (atLeastOnePermit)
4686   {
4687     return Permit;
4688   }
4689   if (atLeastOneError)
4690   {
4691     return Indeterminate;
4692   }
4693   return NotApplicable;
4694 }

```

4695 The following specification defines the “Deny-overrides” **policy-combining algorithm** of a **policy**
4696 **set**.

4697 In the entire set of **policies** in the **policy set**, if any **policy** evaluates to "Deny", then the
4698 result of the **policy** combination SHALL be "Deny". In other words, "Deny" takes
4699 precedence, regardless of the result of evaluating any of the other **policies** in the **policy**
4700 **set**. If all **policies** are found to be "NotApplicable" to the **decision request**, then the
4701 **policy set** SHALL evaluate to "NotApplicable".

4702 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is
4703 considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set**
4704 SHALL evaluate to "Deny".

4705 The following pseudo-code represents the evaluation strategy of this **policy-combining algorithm**.

```

4706 Decision denyOverridesPolicyCombiningAlgorithm(Policy policy[])
4707 {
4708   Boolean atLeastOnePermit = false;
4709   for( i=0 ; i < lengthOf(policy) ; i++ )
4710   {
4711     Decision decision = evaluate(policy[i]);
4712     if (decision == Deny)
4713     {
4714       return Deny;
4715     }
4716     if (decision == Permit)
4717     {
4718       atLeastOnePermit = true;
4719       continue;
4720     }
4721     if (decision == NotApplicable)
4722     {
4723       continue;
4724     }
4725     if (decision == Indeterminate)
4726     {
4727       return Deny;
4728     }
4729   }
4730   if (atLeastOnePermit)
4731   {
4732     return Permit;
4733   }
4734   return NotApplicable;
4735 }

```

4736 **Obligations** of the individual **policies** shall be combined as described in Section 7.11.

4737

C.2. Ordered-deny-overrides (non-normative)

4738 The following specification defines the "Ordered-deny-overrides" *rule-combining algorithm* of a
4739 *policy*.

4740 The behavior of this algorithm is identical to that of the Deny-overrides *rule-combining*
4741 *algorithm* with one exception. The order in which the collection of *rules* is evaluated SHALL
4742 match the order as listed in the *policy*.

4743 The following specification defines the "Ordered-deny-overrides" *policy-combining algorithm* of a
4744 *policy set*.

4745 The behavior of this algorithm is identical to that of the Deny-overrides *policy-combining*
4746 *algorithm* with one exception. The order in which the collection of *policies* is evaluated SHALL
4747 match the order as listed in *the policy set*.

4748

C.3. Permit-overrides

4749 The following specification defines the "Permit-overrides" *rule-combining algorithm* of a *policy*.

4750 In the entire set of *rules* in the *policy*, if any *rule* evaluates to "Permit", then the result of
4751 the *rule* combination SHALL be "Permit". If any *rule* evaluates to "Deny" and all other
4752 *rules* evaluate to "NotApplicable", then the *policy* SHALL evaluate to "Deny". In other
4753 words, "Permit" takes precedence, regardless of the result of evaluating any of the other
4754 *rules* in the *policy*. If all *rules* are found to be "NotApplicable" to the *decision request*,
4755 then the *policy* SHALL evaluate to "NotApplicable".

4756 If an error occurs while evaluating the *target* or *condition* of a *rule* that contains an *effect*
4757 of "Permit" then the evaluation SHALL continue looking for a result of "Permit". If no other
4758 *rule* evaluates to "Permit", then the *policy* SHALL evaluate to "Indeterminate", with the
4759 appropriate error status.

4760 If at least one *rule* evaluates to "Deny", all other *rules* that do not have evaluation errors
4761 evaluate to "Deny" or "NotApplicable" and all *rules* that do have evaluation errors contain
4762 an *effect* value of "Deny", then the *policy* SHALL evaluate to "Deny".

4763 The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

```
4764 Decision permitOverridesRuleCombiningAlgorithm(Rule rule[])
4765 {
4766     Boolean atLeastOneError = false;
4767     Boolean potentialPermit = false;
4768     Boolean atLeastOneDeny = false;
4769     for( i=0 ; i < lengthOf(rule) ; i++ )
4770     {
4771         Decision decision = evaluate(rule[i]);
4772         if (decision == Deny)
4773         {
4774             atLeastOneDeny = true;
4775             continue;
4776         }
4777         if (decision == Permit)
4778         {
4779             return Permit;
4780         }
4781         if (decision == NotApplicable)
4782         {
4783             continue;
```

```

4784     }
4785     if (decision == Indeterminate)
4786     {
4787         atLeastOneError = true;
4788
4789         if (effect(rule[i]) == Permit)
4790         {
4791             potentialPermit = true;
4792         }
4793         continue;
4794     }
4795 }
4796 if (potentialPermit)
4797 {
4798     return Indeterminate;
4799 }
4800 if (atLeastOneDeny)
4801 {
4802     return Deny;
4803 }
4804 if (atLeastOneError)
4805 {
4806     return Indeterminate;
4807 }
4808 return NotApplicable;
4809 }

```

4810 The following specification defines the "Permit-overrides" *policy-combining algorithm* of a *policy set*.
4811

4812 In the entire set of *policies* in the *policy set*, if any *policy* evaluates to "Permit", then the
4813 result of the *policy* combination SHALL be "Permit". In other words, "Permit" takes
4814 precedence, regardless of the result of evaluating any of the other *policies* in the *policy*
4815 *set*. If all *policies* are found to be "NotApplicable" to the *decision request*, then the
4816 *policy set* SHALL evaluate to "NotApplicable".

4817 If an error occurs while evaluating the *target* of a *policy*, a reference to a *policy* is
4818 considered invalid or the *policy* evaluation results in "Indeterminate", then the *policy set*
4819 SHALL evaluate to "Indeterminate", with the appropriate error status, provided no other
4820 *policies* evaluate to "Permit" or "Deny".

4821 The following pseudo-code represents the evaluation strategy of this *policy-combining algorithm*.

```

4822 Decision permitOverridesPolicyCombiningAlgorithm(Policy policy[])
4823 {
4824     Boolean atLeastOneError = false;
4825     Boolean atLeastOneDeny = false;
4826     for( i=0 ; i < lengthOf(policy) ; i++ )
4827     {
4828         Decision decision = evaluate(policy[i]);
4829         if (decision == Deny)
4830         {
4831             atLeastOneDeny = true;
4832             continue;
4833         }
4834         if (decision == Permit)
4835         {
4836             return Permit;
4837         }
4838         if (decision == NotApplicable)
4839         {
4840             continue;
4841         }

```

```

4842     if (decision == Indeterminate)
4843     {
4844         atLeastOneError = true;
4845         continue;
4846     }
4847 }
4848 if (atLeastOneDeny)
4849 {
4850     return Deny;
4851 }
4852 if (atLeastOneError)
4853 {
4854     return Indeterminate;
4855 }
4856 return NotApplicable;
4857 }

```

4858 **Obligations** of the individual policies shall be combined as described in Section 7.11.

4859 C.4. Ordered-permit-overrides (non-normative)

4860 The following specification defines the "Ordered-permit-overrides" **rule-combining algorithm** of a **policy**.

4862 The behavior of this algorithm is identical to that of the Permit-overrides **rule-combining algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL match the order as listed in the **policy**.

4865 The following specification defines the "Ordered-permit-overrides" **policy-combining algorithm** of a **policy set**.

4867 The behavior of this algorithm is identical to that of the Permit-overrides **policy-combining algorithm** with one exception. The order in which the collection of **policies** is evaluated SHALL match the order as listed in the **policy set**.

4870 C.5. First-applicable

4871 The following specification defines the "First-Applicable" **rule-combining algorithm** of a **policy**.

4872 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".

4879 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.

4882 The following pseudo-code represents the evaluation strategy of this **rule-combining algorithm**.

```

4883 Decision firstApplicableEffectRuleCombiningAlgorithm(Rule rule[])
4884 {
4885     for( i = 0 ; i < lengthOf(rule) ; i++ )
4886     {

```



```

4887     Decision decision = evaluate(rule[i]);
4888     if (decision == Deny)
4889     {
4890         return Deny;
4891     }
4892     if (decision == Permit)
4893     {
4894         return Permit;
4895     }
4896     if (decision == NotApplicable)
4897     {
4898         continue;
4899     }
4900     if (decision == Indeterminate)
4901     {
4902         return Indeterminate;
4903     }
4904 }
4905 return NotApplicable;
4906 }

```

4907 The following specification defines the “First-applicable” *policy-combining algorithm* of a *policy*
4908 *set*.

4909 Each *policy* is evaluated in the order that it appears in the *policy set*. For a particular
4910 *policy*, if the *target* evaluates to "True" and the *policy* evaluates to a determinate value of
4911 "Permit" or "Deny", then the evaluation SHALL halt and the *policy set* SHALL evaluate to
4912 the *effect* value of that *policy*. For a particular *policy*, if the *target* evaluate to "False", or
4913 the *policy* evaluates to "NotApplicable", then the next *policy* in the order SHALL be
4914 evaluated. If no further *policy* exists in the order, then the *policy set* SHALL evaluate to
4915 "NotApplicable".

4916 If an error were to occur when evaluating the *target*, or when evaluating a specific *policy*,
4917 the reference to the *policy* is considered invalid, or the *policy* itself evaluates to
4918 "Indeterminate", then the evaluation of the *policy-combining algorithm* shall halt, and the
4919 *policy set* shall evaluate to "Indeterminate" with an appropriate error status.

4920 The following pseudo-code represents the evaluation strategy of this *policy-combination*
4921 *algorithm*.

```

4922 Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy policy[])
4923 {
4924     for( i = 0 ; i < lengthOf(policy) ; i++ )
4925     {
4926         Decision decision = evaluate(policy[i]);
4927         if(decision == Deny)
4928         {
4929             return Deny;
4930         }
4931         if(decision == Permit)
4932         {
4933             return Permit;
4934         }
4935         if (decision == NotApplicable)
4936         {
4937             continue;
4938         }
4939         if (decision == Indeterminate)
4940         {
4941             return Indeterminate;
4942         }
4943     }
4944     return NotApplicable;

```

4945

```
}
```

4946

Obligations of the individual policies shall be combined as described in Section 7.11.

4947

C.6. Only-one-applicable

4948

The following specification defines the "Only-one-applicable" **policy-combining algorithm** of a **policy set**.

4949

4950

In the entire set of policies in the **policy set**, if no **policy** is considered applicable by virtue of their **targets**, then the result of the policy combination algorithm SHALL be "NotApplicable". If more than one policy is considered applicable by virtue of their **targets**, then the result of the policy combination algorithm SHALL be "Indeterminate".

4951

4952

4953

4954

If only one **policy** is considered applicable by evaluation of the **policy targets**, then the result of the **policy-combining algorithm** SHALL be the result of evaluating the **policy**.

4955

4956

If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set** SHALL evaluate to "Indeterminate", with the appropriate error status.

4957

4958

4959

The following pseudo-code represents the evaluation strategy of this policy combining algorithm.

4960

```
Decision onlyOneApplicablePolicyPolicyCombiningAlgorithm(Policy policy[])
```

4961

```
{
```

4962

```
  Boolean          atLeastOne      = false;
```

4963

```
  Policy           selectedPolicy = null;
```

4964

```
  ApplicableResult appResult;
```

4965

4966

```
  for ( i = 0; i < lengthOf(policy) ; i++ )
```

4967

```
  {
```

4968

```
    appResult = isApplicable(policy[i]);
```

4969

4970

```
    if ( appResult == Indeterminate )
```

4971

```
    {
```

4972

```
      return Indeterminate;
```

4973

```
    }
```

4974

```
    if( appResult == Applicable )
```

4975

```
    {
```

4976

```
      if ( atLeastOne )
```

4977

```
      {
```

4978

```
        return Indeterminate;
```

4979

```
      }
```

4980

```
      else
```

4981

```
      {
```

4982

```
        atLeastOne      = true;
```

4983

```
        selectedPolicy = policy[i];
```

4984

```
      }
```

4985

```
    }
```

4986

```
    if ( appResult == NotApplicable )
```

4987

```
    {
```

4988

```
      continue;
```

4989

```
    }
```

4990

```
  }
```

4991

```
  if ( atLeastOne )
```

4992

```
  {
```

4993

```
    return evaluate(selectedPolicy);
```

4994

```
  }
```

4995

```
  else
```

4996

```
  {
```

4997

```
    return NotApplicable;
```

4998
4999
5000

```
}  
}
```

5001 **Appendix D. Acknowledgments**

5002 The following individuals contributed to the development of the specification:

5003 Anne Anderson
5004 Bill Parducci
5005 Carlisle Adams
5006 Daniel Engovatov
5007 Don Flinn
5008 Ernesto Damiani
5009 Gerald Brose
5010 Hal Lockhart
5011 James MacLean
5012 John Merrells
5013 Ken Yagen
5014 Konstantin Beznosov
5015 Michiharu Kudo
5016 Pierangela Samarati
5017 Pirasenna Velandai Thiyagarajan
5018 Polar Humenn
5019 Satoshi Hada
5020 Sekhar Vajjhala
5021 Seth Proctor
5022 Simon Godik
5023 Steve Anderson
5024 Steve Crocker
5025 Suresh Damodaran
5026 Tim Moses
5027

5028

Appendix E. Revision history

Rev	Date	By whom	What
OS V1.0	18 Feb 2003	XACML Technical Committee	OASIS Standard

5029

5030

Appendix F. Notices

5031 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
5032 that might be claimed to pertain to the implementation or use of the technology described in this
5033 document or the extent to which any license under such rights might or might not be available;
5034 neither does it represent that it has made any effort to identify any such rights. Information on
5035 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
5036 website. Copies of claims of rights made available for publication and any assurances of licenses to
5037 be made available, or the result of an attempt made to obtain a general license or permission for
5038 the use of such proprietary rights by implementors or users of this specification, can be obtained
5039 from the OASIS Executive Director.

5040 OASIS has been notified of intellectual property rights claimed in regard to some or all of the
5041 contents of this specification. For more information consult the online list of claimed rights.

5042 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
5043 applications, or other proprietary rights which may cover technology that may be required to
5044 implement this specification. Please address the information to the OASIS Executive Director.

5045 Copyright (C) OASIS Open 2003. All Rights Reserved.

5046 This document and translations of it may be copied and furnished to others, and derivative works
5047 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
5048 published and distributed, in whole or in part, without restriction of any kind, provided that the above
5049 copyright notice and this paragraph are included on all such copies and derivative works. However,
5050 this document itself may not be modified in any way, such as by removing the copyright notice or
5051 references to OASIS, except as needed for the purpose of developing OASIS specifications, in
5052 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights
5053 document must be followed, or as required to translate it into languages other than English.

5054 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
5055 successors or assigns.

5056 This document and the information contained herein is provided on an "AS IS" basis and OASIS
5057 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
5058 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY
5059 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
5060 PARTICULAR PURPOSE.