



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

4 **Committee draft 01, 16 Sep 2004**

5 Document identifier: access_control-xacml-2.0-core-spec-cd-01

6 Location: http://docs.oasis-open.org/xacml/access_control-xacml-2.0-core-spec-cd-01.pdf

7 Editors:

8 Simon Godik, GlueCode Software

9 Tim Moses, Entrust

10 Committee members:

11 Anne Anderson, Sun Microsystems

12 Anthony Nadalin, IBM

13 Bill Parducci, GlueCode Software

14 Daniel Engovatov, BEA Systems

15 Ed Coyne, Veterans Health Administration

16 Frank Siebenlist, Argonne National Labs

17 Hal Lockhart, BEA Systems

18 Michael McIntosh, IBM

19 Michiharu Kudo, IBM

20 Polar Humenn, Self

21 Ron Jacobson, Computer Associates

22 Seth Proctor, Sun Microsystems

23 Simon Godik, GlueCode Software

24 Steve Anderson, OpenNetwork

25 Tim Moses, Entrust

26 Abstract:

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

28 Status:

29 This version of the specification is an approved Committee Draft within the OASIS Access
30 Control TC.

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

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

access_control-xacml-2.0-core-spec-cd-01

- 38 For any errata page for this specification, please refer to the Access Control TC web page
39 (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml).
- 40 Copyright (C) OASIS Open 2004. All Rights Reserved.

41	Table of contents		
42	1.	Introduction (non-normative).....	9
43	1.1.	Glossary.....	9
44	1.1.1	Preferred terms	9
45	1.1.2	Related terms.....	10
46	1.2.	Notation	11
47	1.3.	Schema organization and namespaces	11
48	2.	Background (non-normative)	12
49	2.1.	Requirements.....	12
50	2.2.	Rule and policy combining.....	13
51	2.3.	Combining algorithms.....	13
52	2.4.	Multiple subjects.....	14
53	2.5.	Policies based on subject and resource attributes.....	14
54	2.6.	Multi-valued attributes	15
55	2.7.	Policies based on resource contents	15
56	2.8.	Operators.....	15
57	2.9.	Policy distribution	16
58	2.10.	Policy indexing.....	16
59	2.11.	Abstraction layer	16
60	2.12.	Actions performed in conjunction with enforcement	17
61	3.	Models (non-normative).....	17
62	3.1.	Data-flow model	17
63	3.2.	XACML context	19
64	3.3.	Policy language model	19
65	3.3.1	Rule	21
66	3.3.2	Policy	22
67	3.3.3	Policy set.....	23
68	4.	Examples (non-normative)	23
69	4.1.	Example one.....	24
70	4.1.1	Example policy	24
71	4.1.2	Example request context	26
72	4.1.3	Example response context.....	27
73	4.2.	Example two	27
74	4.2.1	Example medical record instance	27
75	4.2.2	Example request context	29
76	4.2.3	Example plain-language rules.....	30

77	4.2.4	Example XACML rule instances.....	31
78	5.	Policy syntax (normative, with the exception of the schema fragments).....	43
79	5.1.	Element <PolicySet>	43
80	5.2.	Element <Description>	46
81	5.3.	Element <PolicySetDefaults>	46
82	5.4.	Element <XPathVersion>	46
83	5.5.	Element <Target>	46
84	5.6.	Element <Subjects>	47
85	5.7.	Element <Subject>	47
86	5.8.	Element <SubjectMatch>	48
87	5.9.	Element <Resources>	48
88	5.10.	Element <Resource>.....	49
89	5.11.	Element <ResourceMatch>	49
90	5.12.	Element <Actions>	50
91	5.13.	Element <Action>.....	50
92	5.14.	Element <ActionMatch>	50
93	5.15.	Element <Environments>	51
94	5.16.	Element <Environment>	51
95	5.17.	Element <EnvironmentMatch>.....	52
96	5.18.	Element <PolicySetIdReference>	52
97	5.19.	Element <PolicyIdReference>	53
98	5.20.	Simple type VersionType.....	53
99	5.21.	Simple type VersionMatchType	53
100	5.22.	Element <Policy>	54
101	5.23.	Element <PolicyDefaults>	56
102	5.24.	Element <CombinerParameters>	56
103	5.25.	Element <CombinerParameter>	56
104	5.26.	Element <RuleCombinerParameters>	57
105	5.27.	Element <PolicyCombinerParameters>	57
106	5.28.	Element <PolicySetCombinerParameters>	58
107	5.29.	Element <Rule>	59
108	5.30.	Simple type EffectType.....	59
109	5.31.	Element <VariableDefinition>	60
110	5.32.	Element <VariableReference>.....	60
111	5.33.	Element <Expression>	61
112	5.34.	Element <Condition>.....	61
113	5.35.	Element <Apply>.....	61

114	5.36.	Element <Function>	62
115	5.37.	Complex type AttributeDesignatorType.....	62
116	5.38.	Element <SubjectAttributeDesignator>	63
117	5.39.	Element <ResourceAttributeDesignator>	64
118	5.40.	Element <ActionAttributeDesignator>	64
119	5.41.	Element <EnvironmentAttributeDesignator>	65
120	5.42.	Element <AttributeSelector>.....	65
121	5.43.	Element <AttributeValue>.....	67
122	5.44.	Element <Obligations>	67
123	5.45.	Element <Obligation>.....	67
124	5.46.	Element <AttributeAssignment>	68
125	6.	Context syntax (normative with the exception of the schema fragments).....	69
126	6.1.	Element <Request>.....	69
127	6.2.	Element <Subject>	70
128	6.3.	Element <Resource>.....	70
129	6.4.	Element <ResourceContent>	71
130	6.5.	Element <Action>.....	71
131	6.6.	Element <Environment>	72
132	6.7.	Element <Attribute>.....	72
133	6.8.	Element <AttributeValue>.....	73
134	6.9.	Element <Response>	73
135	6.10.	Element <Result>.....	74
136	6.11.	Element <Decision>	74
137	6.12.	Element <Status>.....	75
138	6.13.	Element <StatusCode>	75
139	6.14.	Element <StatusMessage>.....	76
140	6.15.	Element <StatusDetail>.....	76
141	6.16.	Element <MissingAttributeDetail>.....	77
142	7.	Functional requirements (normative)	77
143	7.1.	Policy enforcement point	77
144	7.1.1.	Base PEP.....	78
145	7.1.2.	Deny-biased PEP	78
146	7.1.3.	Permit-biased PEP	78
147	7.2.	Attribute evaluation	78
148	7.2.1.	Structured attributes	78
149	7.2.2.	Attribute bags	79
150	7.2.3.	Multivalued attributes.....	79

151	7.2.4.	Attribute Matching.....	79
152	7.2.5.	Attribute Retrieval.....	80
153	7.2.6.	Environment Attributes	80
154	7.3.	Expression evaluation	80
155	7.4.	Arithmetic evaluation.....	81
156	7.5.	Match evaluation.....	81
157	7.6.	Target evaluation	83
158	7.7.	VariableReference Evaluation	84
159	7.8.	Condition evaluation.....	84
160	7.9.	Rule evaluation	84
161	7.10.	Policy evaluation.....	85
162	7.11.	Policy Set evaluation.....	86
163	7.12.	Hierarchical resources.....	87
164	7.13.	Authorization decision	87
165	7.14.	Obligations.....	87
166	7.15.	Exception handling.....	87
167	7.15.1.	Unsupported functionality	88
168	7.15.2.	Syntax and type errors.....	88
169	7.15.3.	Missing attributes.....	88
170	8.	XACML extensibility points (non-normative).....	88
171	8.1.	Extensible XML attribute types	89
172	8.2.	Structured attributes.....	89
173	9.	Security and privacy considerations (non-normative).....	89
174	9.1.	Threat model.....	90
175	9.1.1.	Unauthorized disclosure	90
176	9.1.2.	Message replay	90
177	9.1.3.	Message insertion	90
178	9.1.4.	Message deletion	91
179	9.1.5.	Message modification.....	91
180	9.1.6.	NotApplicable results.....	91
181	9.1.7.	Negative rules	91
182	9.2.	Safeguards	92
183	9.2.1.	Authentication.....	92
184	9.2.2.	Policy administration.....	92
185	9.2.3.	Confidentiality.....	93
186	9.2.4.	Policy integrity	93
187	9.2.5.	Policy identifiers	94

188	9.2.6.	Trust model	94
189	9.2.7.	Privacy	94
190	10.	Conformance (normative)	95
191	10.1.	Introduction.....	95
192	10.2.	Conformance tables.....	95
193	10.2.1.	Schema elements.....	95
194	10.2.2.	Identifier Prefixes.....	96
195	10.2.3.	Algorithms	96
196	10.2.4.	Status Codes.....	97
197	10.2.5.	Attributes.....	97
198	10.2.6.	Identifiers	97
199	10.2.7.	Data-types.....	98
200	10.2.8.	Functions	98
201	11.	References.....	102
202	Appendix A.	Data-types and functions (normative).....	105
203	A.1.	Introduction	105
204	A.2.	Data-types	105
205	A.3.	Functions	107
206	A.3.1	Equality predicates	107
207	A.3.2	Arithmetic functions	109
208	A.3.3	String conversion functions.....	110
209	A.3.4	Numeric data-type conversion functions.....	110
210	A.3.5	Logical functions.....	110
211	A.3.6	Numeric comparison functions.....	111
212	A.3.7	Date and time arithmetic functions	112
213	A.3.8	Non-numeric comparison functions.....	113
214	A.3.9	String functions.....	116
215	A.3.10	Bag functions.....	116
216	A.3.11	Set functions	117
217	A.3.12	Higher-order bag functions	117
218	A.3.13	Regular-expression-based functions	124
219	A.3.14	Special match functions.....	125
220	A.3.15	XPath-based functions.....	126
221	A.3.16	Extension functions and primitive types.....	126
222	Appendix B.	XACML identifiers (normative).....	127
223	B.1.	XACML namespaces.....	127
224	B.2.	Access subject categories.....	127

225	B.3. Data-types	127
226	B.4. Subject attributes	128
227	B.6. Resource attributes	129
228	B.7. Action attributes	129
229	B.8. Environment attributes	130
230	B.9. Status codes	130
231	B.10. Combining algorithms	130
232	Appendix C. Combining algorithms (normative).....	132
233	C.1. Deny-overrides	132
234	C.2. Ordered-deny-overrides.....	134
235	C.3. Permit-overrides	134
236	C.4. Ordered-permit-overrides.....	136
237	C.5. First-applicable	136
238	C.6. Only-one-applicable.....	138
239	Appendix D. Acknowledgments	140
240	Appendix E. Revision history	141
241	Appendix F. Notices	142
242		

244 1. Introduction (non-normative)

245 1.1. Glossary

246 1.1.1 Preferred terms

247 **Access** - Performing an *action*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

279 **Policy-combining algorithm** - The procedure for combining the **decision** and **obligations** from
 280 multiple **policies**

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

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

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

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

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

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

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

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

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

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

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

307 1.1.2 Related terms

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

310 For instance, the term **attribute** is used in place of the terms: group and role.
311 In place of the terms: privilege, permission, authorization, entitlement and right, we use the term
312 **rule**.
313 The term object is also in common use, but we use the term **resource** in this specification.
314 Requestors and initiators are covered by the term **subject**.

315 1.2. Notation

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

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

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

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

327 Listings of XACML schema appear like this.

328
329 [a01] Example code listings appear like this.

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

- 333 • The prefix `xacml`: stands for the XACML policy namespace.
- 334 • The prefix `xacml-context`: stands for the XACML context namespace.
- 335 • The prefix `ds`: stands for the W3C XML Signature namespace [DS].
- 336 • The prefix `xs`: stands for the W3C XML Schema namespace [XS].
- 337 • The prefix `xf`: stands for the XQuery 1.0 and XPath 2.0 Function and Operators
338 specification namespace [XF].

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

342 1.3. Schema organization and namespaces

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

344 `urn:oasis:names:tc:xacml:2.0:policy`

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

346 `urn:oasis:names:tc:xacml:2.0:context`

347

2. Background (non-normative)

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

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

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

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

2.1. Requirements

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

- 376 • To provide a method for combining individual **rules** and **policies** into a single **policy set** that
377 applies to a particular **decision request**.
- 378 • To provide a method for flexible definition of the procedure by which **rules** and **policies** are
379 combined.
- 380 • To provide a method for dealing with multiple **subjects** acting in different capacities.
- 381 • To provide a method for basing an **authorization decision** on **attributes** of the **subject** and
382 **resource**.
- 383 • To provide a method for dealing with multi-valued **attributes**.
- 384 • To provide a method for basing an **authorization decision** on the contents of an information
385 **resource**.
- 386 • To provide a set of logical and mathematical operators on **attributes** of the **subject**, **resource**
387 and **environment**.

- 388 • To provide a method for handling a distributed set of **policy** components, while abstracting the
389 method for locating, retrieving and authenticating the **policy** components.
- 390 • To provide a method for rapidly identifying the **policy** that applies to a given action, based upon
391 the values of **attributes** of the **subjects**, **resource** and **action**.
- 392 • To provide an abstraction-layer that insulates the policy-writer from the details of the application
393 environment.
- 394 • To provide a method for specifying a set of actions that must be performed in conjunction with
395 policy enforcement.

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

399 2.2. Rule and policy combining

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

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

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

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

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

419 2.3. Combining algorithms

420 XACML defines a number of combining algorithms that can be identified by a
421 `RuleCombiningAlgId` or `PolicyCombiningAlgId` attribute of the `<Policy>` or `<PolicySet>`
422 elements, respectively. The **rule-combining algorithm** defines a procedure for arriving at an
423 **authorization decision** given the individual results of evaluation of a set of **rules**. Similarly, the
424 **policy-combining algorithm** defines a procedure for arriving at an **authorization decision** given
425 the individual results of evaluation of a set of **policies**. Standard combining algorithms are defined
426 for:

- 427 • Deny-overrides (Ordered and Unordered),
- 428 • Permit-overrides (Ordered and Unordered),

- 429 • First-applicable and
 - 430 • Only-one-applicable.
- 431 In the case of the Deny-overrides algorithm, if a single <Rule> or <Policy> element is
 432 encountered that evaluates to "Deny", then, regardless of the evaluation result of the other <Rule>
 433 or <Policy> elements in the **applicable policy**, the combined result is "Deny".
- 434 Likewise, in the case of the Permit-overrides algorithm, if a single "Permit" result is encountered,
 435 then the combined result is "Permit".
- 436 In the case of the "First-applicable" combining algorithm, the combined result is the same as the
 437 result of evaluating the first <Rule>, <Policy> or <PolicySet> element in the list of **rules**
 438 whose **target** is applicable to the **decision request**.
- 439 The "Only-one-applicable" **policy-combining algorithm** only applies to **policies**. The result of this
 440 combining algorithm ensures that one and only one **policy** or **policy set** is applicable by virtue of
 441 their **targets**. If no **policy** or **policy set** applies, then the result is "NotApplicable", but if more than
 442 one **policy** or **policy set** is applicable, then the result is "Indeterminate". When exactly one **policy**
 443 or **policy set** is applicable, the result of the combining algorithm is the result of evaluating the
 444 single **applicable policy** or **policy set**.
- 445 **Policies** and **policy sets** may take parameters that modify the behaviour of the **combining**
 446 **algorithms**. However, none of the standard **combining algorithms** is affected by parameters.
- 447 Users of this specification may, if necessary, define their own combining algorithms.

448 2.4. Multiple subjects

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

455 2.5. Policies based on subject and resource attributes

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

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

467 Another common requirement is to base an **authorization decision** on some characteristic of the
 468 **resource** other than its identity. XACML provides facilities to support this approach. **Attributes** of
 469 the **resource** may be identified by the <ResourceAttributeDesignator> element. This
 470 element contains a URN that identifies the **attribute**. Alternatively, the <AttributeSelector>

471 element may contain an XPath expression over the request **context** to identify a particular
472 **resource attribute** value by its location in the **context**.

473 **2.6. Multi-valued attributes**

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

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

483 **2.7. Policies based on resource contents**

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

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

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

494 **2.8. Operators**

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

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

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

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

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

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

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

523 **2.9. Policy distribution**

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

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

535 **2.10. Policy indexing**

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

541 Two approaches are supported:

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

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

550 **2.11. Abstraction layer**

551 **PEPs** come in many forms. For instance, a **PEP** may be part of a remote-access gateway, part of
552 a Web server or part of an email user-agent, etc.. It is unrealistic to expect that all **PEPs** in an
553 enterprise do currently, or will in the future, issue **decision requests** to a **PDP** in a common format.
554 Nevertheless, a particular **policy** may have to be enforced by multiple **PEPs**. It would be inefficient

555 to force a policy writer to write the same **policy** several different ways in order to accommodate the
556 format requirements of each **PEP**. Similarly attributes may be contained in various envelope types
557 (e.g. X.509 attribute certificates, SAML attribute assertions, etc.). Therefore, there is a need for a
558 canonical form of the request and response handled by an XACML **PDP**. This canonical form is
559 called the XACML **context**. Its syntax is defined in XML schema.

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

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

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

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

573 **2.12. Actions performed in conjunction with enforcement**

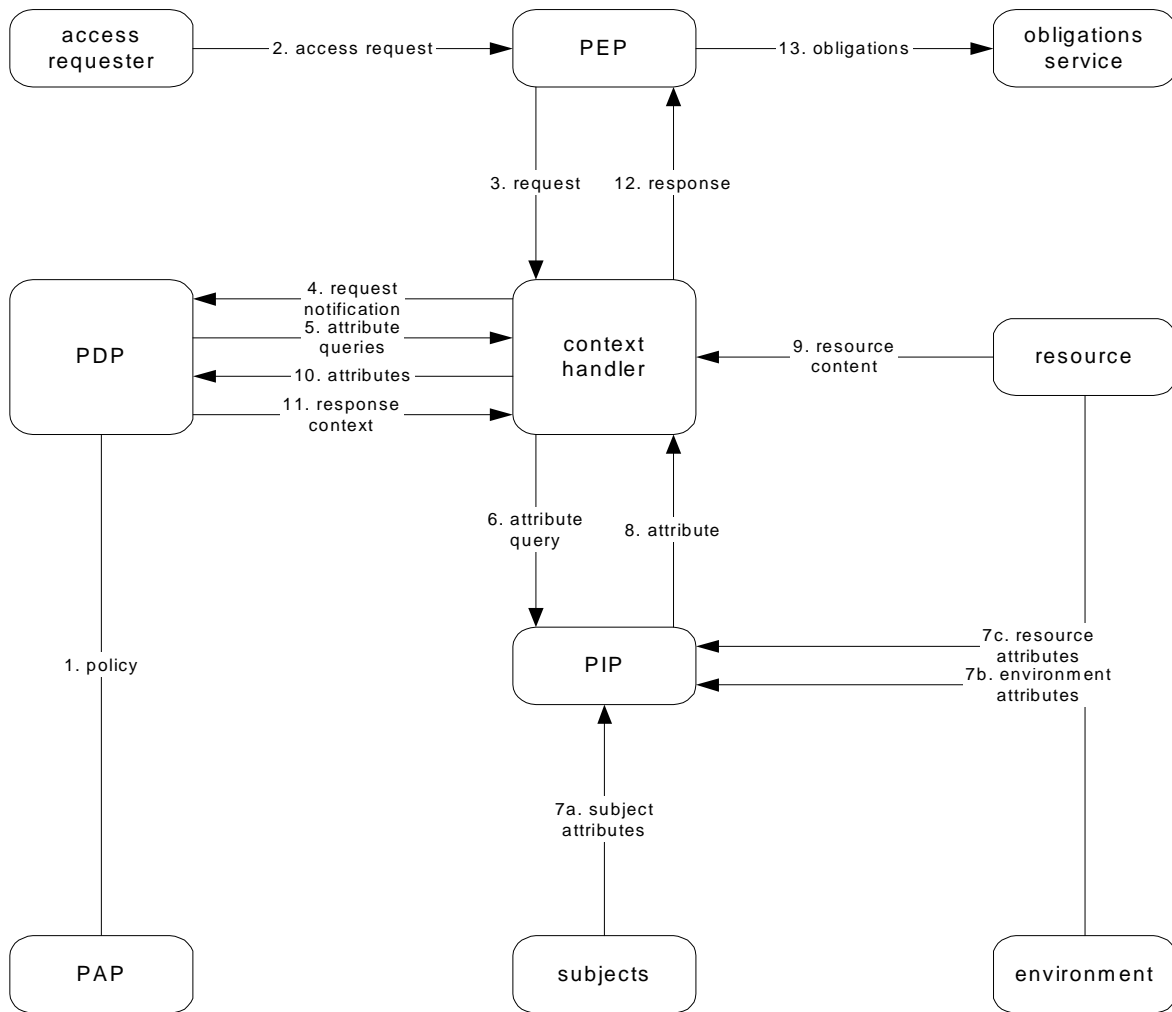
574 In many applications, policies specify actions that **MUST** be performed, either instead of, or in
575 addition to, actions that **MAY** be performed. This idea was described by Sloman [Sloman94].
576 XACML provides facilities to specify actions that **MUST** be performed in conjunction with policy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

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

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

585 **3.1. Data-flow model**

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



587

588

Figure 1 - Data-flow diagram

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

594 The model operates by the following steps.

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

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

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

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

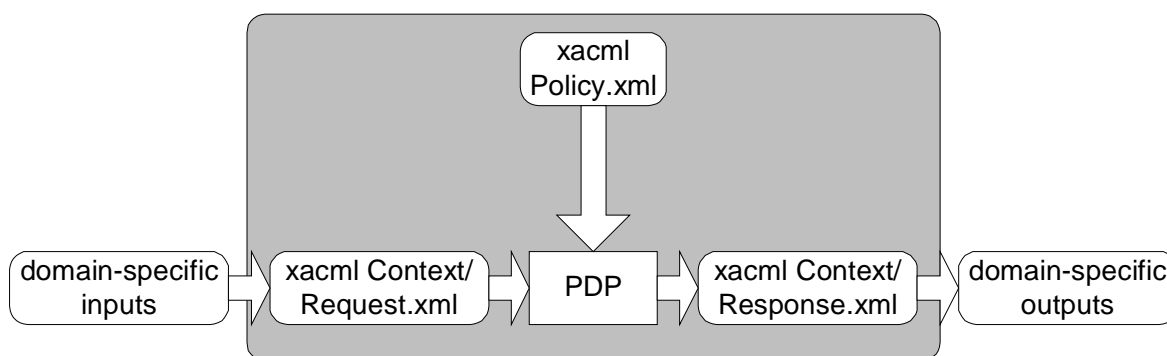
601 5. The **PDP** requests any additional **subject**, **resource**, **action** and **environment attributes** from
 602 the **context handler**.

603 6. The context handler requests the attributes from a **PIP**.

- 604 7. The **PIP** obtains the requested **attributes**.
- 605 8. The **PIP** returns the requested **attributes** to the **context handler**.
- 606 9. Optionally, the **context handler** includes the **resource** in the **context**.
- 607 10. The **context handler** sends the requested **attributes** and (optionally) the **resource** to the **PDP**.
- 608 The **PDP** evaluates the **policy**.
- 609 11. The **PDP** returns the response **context** (including the **authorization decision**) to the **context**
- 610 **handler**.
- 611 12. The **context handler** translates the response **context** to the native response format of the
- 612 **PEP**. The **context handler** returns the response to the **PEP**.
- 613 13. The **PEP** fulfills the **obligations**.
- 614 14. (Not shown) If **access** is permitted, then the **PEP** permits **access** to the **resource**; otherwise, it
- 615 denies **access**.

616 3.2. XACML context

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



629
630 **Figure 2 - XACML context**

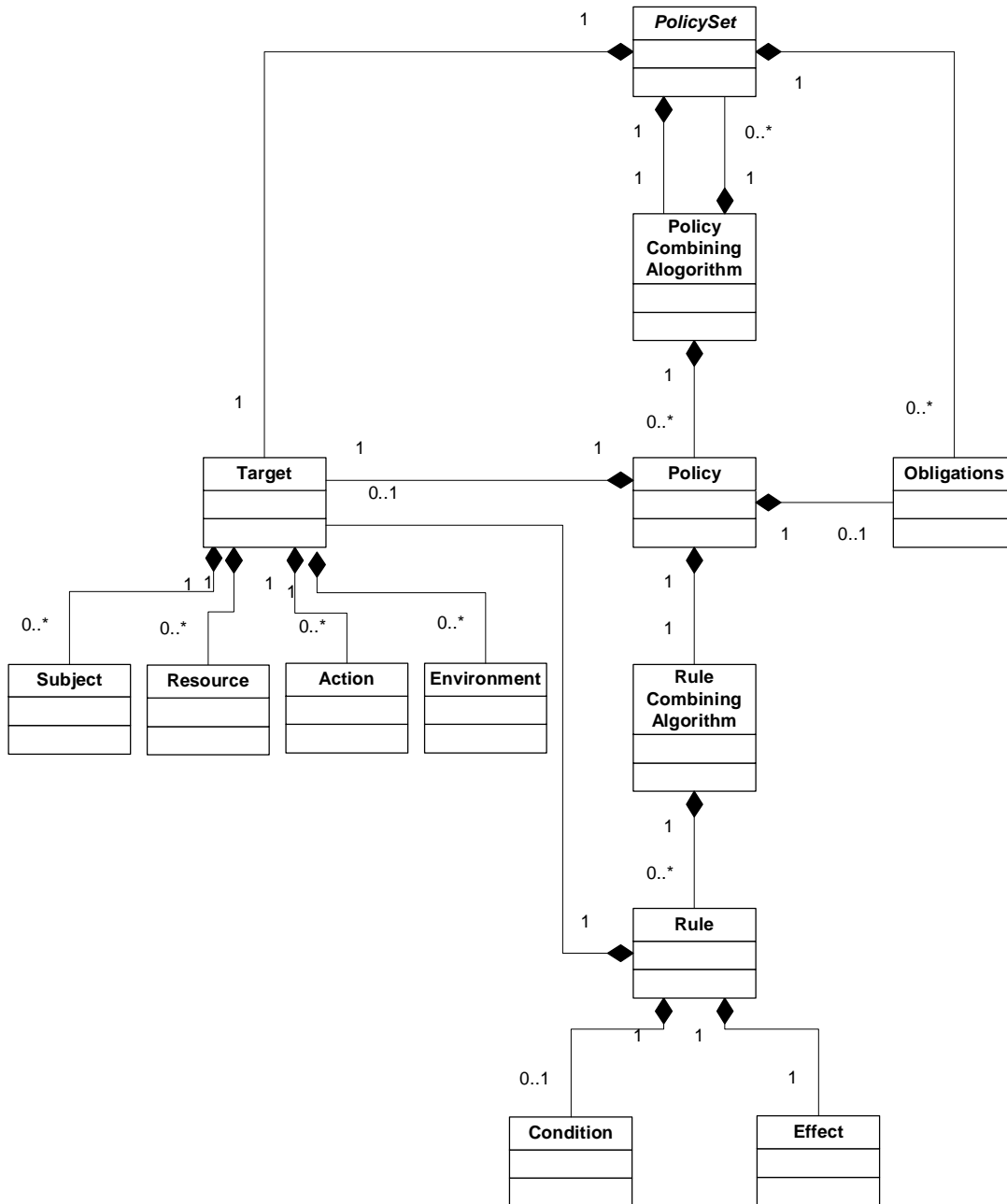
631 Note: The **PDP** is not required to operate directly on the XACML representation of a policy. It may
 632 operate directly on an alternative representation.

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

634 3.3. Policy language model

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

- 636 • **Rule**;
 - 637 • **Policy**; and
 - 638 • **Policy set**.
- 639 These are described in the following sub-sections.



640

641

Figure 3 - Policy language model

642

3.3.1 Rule

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

- 647 • a **target**,
- 648 • an **effect** and
- 649 • a **condition**.

650 These are discussed in the following sub-sections.

651

3.3.1.1. Rule target

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

- 653 • **resources**;
- 654 • **subjects**;
- 655 • **actions** and
- 656 • **environment**

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

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

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

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

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

680 In the case of **subjects**, there is no real entity that corresponds to such a node. So, names of this
681 type always refer to the set of **subjects** subordinate in the name structure to the identified node.
682 Consequently, non-leaf **subject** names should not be used in equality functions, only in match

683 functions, such as "urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match" not
684 "urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal" (see Appendix A).

685 3.3.1.2. Effect

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

688 3.3.1.3. Condition

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

691 3.3.2 Policy

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

- 694 • a **target**,
- 695 • a **rule-combining algorithm**-identifier;
- 696 • a set of **rules**; and
- 697 • **obligations**.

698 **Rules** are described above. The remaining components are described in the following sub-
699 sections.

700 3.3.2.1. Policy target

701 An XACML <PolicySet>, <Policy> or <Rule> element contains a <Target> element that
702 specifies the set of **subjects**, **resources**, **actions** and **environments** to which it applies. The
703 <Target> of a <PolicySet> or <Policy> may be declared by the writer of the <PolicySet> or
704 <Policy>, or it may be calculated from the <Target> elements of the <PolicySet>, <Policy>
705 and <Rule> elements that it contains.

706 A system entity that calculates a <Target> in this way is not defined by XACML, but there are two
707 logical methods that might be used. In one method, the <Target> element of the outer
708 <PolicySet> or <Policy> (the "outer component") is calculated as the **union** of all the
709 <Target> elements of the referenced <PolicySet>, <Policy> or <Rule> elements (the "inner
710 components"). In another method, the <Target> element of the outer component is calculated as
711 the **intersection** of all the <Target> elements of the inner components. The results of evaluation in
712 each case will be very different: in the first case, the <Target> element of the outer component
713 makes it applicable to any **decision request** that matches the <Target> element of at least one
714 inner component; in the second case, the <Target> element of the outer component makes it
715 applicable only to **decision requests** that match the <Target> elements of every inner
716 component. Note that computing the intersection of a set of <Target> elements is likely only
717 practical if the target data-model is relatively simple.

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

722 3.3.2.2. Rule-combining algorithm

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

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

729 3.3.2.3. Obligations

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

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

733 3.3.3 Policy set

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

- 735 • a **target**;
- 736 • a **policy-combining algorithm**-identifier
- 737 • a set of **policies**; and
- 738 • **obligations**.

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

741 3.3.3.1. Policy-combining algorithm

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

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

748 3.3.3.2. Obligations

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

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

753 4. Examples (non-normative)

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

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

758 4.1. Example one

759 4.1.1 Example policy

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

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

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

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

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

805 [a03] introduces the XACML Policy itself.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

852

4.1.2 Example request context

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

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

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

```
860 [a38] <?xml version="1.0" encoding="UTF-8"?>
861 [a39] <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:cd"
862 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
863 [a40] xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:cd
864 http://docs.oasis-open.org/xacml/access_control-xacml-2.0-context-schema-cd.xsd">
865 [a41] <Subject>
866 [a42] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
867 DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
868 [a43] <AttributeValue>
869 [a44] bs@simpsons.com
870 [a45] </AttributeValue>
871 [a46] </Attribute>
872 [a47] </Subject>
873 [a48] <Resource>
874 [a49] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-
875 id" DataType="http://www.w3.org/2001/XMLSchema#anyURI">
876 [a50] <AttributeValue>
877 [a51] file://example/med/record/patient/BartSimpson
878 [a52] </AttributeValue>
879 [a53] </Attribute>
880 [a54] </Resource>
881 [a55] <Action>
882 [a56] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
883 DataType="http://www.w3.org/2001/XMLSchema#string">
884 [a57] <AttributeValue>
885 [a58] read
886 [a59] </AttributeValue>
887 [a60] </Attribute>
888 [a61] </Action>
889 [a62] <Environment/>
890 [a63] </Request>
```

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

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

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

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

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

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

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

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

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

918 4.1.3 Example response context

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

```
922 [a64] <?xml version="1.0" encoding="UTF-8"?>  
923 [a65] <Response xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:cd"  
924 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
925 xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:cd  
926 http://docs.oasis-open.org/xacml/xacml-core-2.0-context-schema-cd.xsd">  
927 [a66] <Result>  
928 [a67] <Decision>NotApplicable</Decision>  
929 [a68] </Result>  
930 [a69] </Response>
```

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

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

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

937 4.2. Example two

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

941 4.2.1 Example medical record instance

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

```
945 [a70] <?xml version="1.0" encoding="UTF-8"?>  
946 [a71] <record xmlns="urn:example:med:schemas:record"
```

```

947 [a72] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
948 [a73] <patient>
949 [a74] <patientName>
950 [a75] <first>Bartholomew</first>
951 [a76] <last>Simpson</last>
952 [a77] </patientName>
953 [a78] <patientContact>
954 [a79] <street>27 Shelbyville Road</street>
955 [a80] <city>Springfield</city>
956 [a81] <state>MA</state>
957 [a82] <zip>12345</zip>
958 [a83] <phone>555.123.4567</phone>
959 [a84] <fax/>
960 [a85] <email/>
961 [a86] </patientContact>
962 [a87] <patientDoB>1992-03-21</patientDoB>
963 [a88] <patientGender>male</patientGender>
964 [a89] <patient-number>555555</patient-number>
965 [a90] </patient>
966 [a91] <parentGuardian>
967 [a92] <parentGuardianId>HS001</parentGuardianId>
968 [a93] <parentGuardianName>
969 [a94] <first>Homer</first>
970 [a95] <last>Simpson</last>
971 [a96] </parentGuardianName>
972 [a97] <parentGuardianContact>
973 [a98] <street>27 Shelbyville Road</street>
974 [a99] <city>Springfield</city>
975 [a100] <state>MA</state>
976 [a101] <zip>12345</zip>
977 [a102] <phone>555.123.4567</phone>
978 [a103] <fax/>
979 [a104] <email>homers@aol.com</email>
980 [a105] </parentGuardianContact>
981 [a106] </parentGuardian>
982 [a107] <primaryCarePhysician>
983 [a108] <physicianName>
984 [a109] <first>Julius</first>
985 [a110] <last>Hibbert</last>
986 [a111] </physicianName>
987 [a112] <physicianContact>
988 [a113] <street>1 First St</street>
989 [a114] <city>Springfield</city>
990 [a115] <state>MA</state>
991 [a116] <zip>12345</zip>
992 [a117] <phone>555.123.9012</phone>
993 [a118] <fax>555.123.9013</fax>
994 [a119] <email/>
995 [a120] </physicianContact>
996 [a121] <registrationID>ABC123</registrationID>
997 [a122] </primaryCarePhysician>
998 [a123] <insurer>
999 [a124] <name>Blue Cross</name>
1000 [a125] <street>1234 Main St</street>
1001 [a126] <city>Springfield</city>
1002 [a127] <state>MA</state>
1003 [a128] <zip>12345</zip>
1004 [a129] <phone>555.123.5678</phone>
1005 [a130] <fax>555.123.5679</fax>
1006 [a131] <email/>
1007 [a132] </insurer>
1008 [a133] <medical>
1009 [a134] <treatment>

```

```

1010 [a135] <drug>
1011 [a136] <name>methylphenidate hydrochloride</name>
1012 [a137] <dailyDosage>30mgs</dailyDosage>
1013 [a138] <startDate>1999-01-12</startDate>
1014 [a139] </drug>
1015 [a140] <comment>
1016 [a141] patient exhibits side-effects of skin coloration and carpal
1017 degeneration
1018 [a142] </comment>
1019 [a143] </treatment>
1020 [a144] <result>
1021 [a145] <test>blood pressure</test>
1022 [a146] <value>120/80</value>
1023 [a147] <date>2001-06-09</date>
1024 [a148] <performedBy>Nurse Betty</performedBy>
1025 [a149] </result>
1026 [a150] </medical>
1027 [a151] </record>

```

1028 4.2.2 Example request context

1029 The following example illustrates a request *context* to which the example *rules* may be applicable.
1030 It represents a request by the physician Julius Hibbert to read the patient date of birth in the record
1031 of Bartholomew Simpson.

```

1032 [a152] <?xml version="1.0" encoding="UTF-8"?>
1033 [a153] <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:cd"
1034 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
1035 urn:oasis:names:tc:xacml:2.0:context:schema:cd http://docs.oasis-
1036 open.org/xacml/access_control-xacml-2.0-context-schema-cd.xsd">
1037 [a154] <Subject>
1038 [a155] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject-category"
1039 DataType="http://www.w3.org/2001/XMLSchema#anyURI">
1040 [a156] <AttributeValue>urn:oasis:names:tc:xacml:1.0:subject-category:access-
1041 subject</AttributeValue>
1042 [a157] </Attribute>
1043 [a158] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
1044 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="med.example.com">
1045 [a159] <AttributeValue>CN=Julius Hibbert</AttributeValue>
1046 [a160] </Attribute>
1047 [a161] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:name-
1048 format" DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1049 Issuer="med.example.com">
1050 [a162] <AttributeValue>
1051 [a163] urn:oasis:names:tc:xacml:1.0:datatype:x500name
1052 [a164] </AttributeValue>
1053 [a165] </Attribute>
1054 [a166] <Attribute
1055 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:role"
1056 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="med.example.com">
1057 [a167] <AttributeValue>physician</AttributeValue>
1058 [a168] </Attribute>
1059 [a169] <Attribute
1060 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:physician-id"
1061 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="med.example.com">
1062 [a170] <AttributeValue>jh1234</AttributeValue>
1063 [a171] </Attribute>
1064 [a172] </Subject>
1065 [a173] <Resource>
1066 [a174] <ResourceContent>
1067 [a175] <md:record xmlns:md="urn:example:med:schemas:record"
1068 xsi:schemaLocation="urn:example:med:schemas:record
1069 http://www.med.example.com/schemas/record.xsd">

```

```

1070 [a176] <md:patient>
1071 [a177] <md:patientDoB>1992-03-21</md:patientDoB>
1072 [a178] <md:patient-number>555555</md:patient-number>
1073 [a179] </md:patient>
1074 [a180] </md:record>
1075 [a181] </ResourceContent>
1076 [a182] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-
1077 id" DataType="http://www.w3.org/2001/XMLSchema#string">
1078 [a183] <AttributeValue>
1079 [a184] //med.example.com/records/bart-simpson.xml#
1080 [a185] xmlns(md=:Resource/ResourceContent/xpointer
1081 [a186] (/md:record/md:patient/md:patientDoB)
1082 [a187] </AttributeValue>
1083 [a188] </Attribute>
1084 [a189] </Resource>
1085 [a190] <Action>
1086 [a191] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1087 DataType="http://www.w3.org/2001/XMLSchema#string">
1088 [a192] <AttributeValue>read</AttributeValue>
1089 [a193] </Attribute>
1090 [a194] </Action>
1091 [a195] <Environment/>
1092 [a196] </Request>

```

1093 [a152] - [a153] Standard namespace declarations.

1094 [a154] - [a172] **Subject** attributes are placed in the <Subject> element of the <Request>
1095 element. Each **attribute** consists of the **attribute** meta-data and the **attribute** value. There is only
1096 one subject involved in this **request**.

1097 [a155] - [a157] Each <Subject> element has a SubjectCategory attribute. The value of this
1098 attribute describes the role that the related **subject** plays in making the **decision request**. The
1099 value of "access-subject" denotes the identity for which the request was issued.

1100 [a158] - [a160] **Subject** subject-id **attribute**.

1101 [a161] - [a165] The format of the subject-id.

1102 [a166] - [a168] **Subject** role **attribute**.

1103 [a169] - [a171] **Subject** physician-id **attribute**.

1104 [a173] - [a189] **Resource attributes** are placed in the <Resource> element of the <Request>
1105 element. Each **attribute** consists of **attribute** meta-data and an **attribute** value.

1106 [a174] - [a181] **Resource** content. The XML resource instance, access to all or part of which may
1107 be requested, is placed here.

1108 [a182] - [a188] The identifier of the **Resource** instance for which access is requested, which is an
1109 XPath expression into the <ResourceContent> element that selects the data to be accessed.

1110 [a190] - [a194] **Action attributes** are placed in the <Action> element of the <Request> element.

1111 [a192] **Action** identifier.

1112 [a195] The empty <Environment> element.

1113 4.2.3 Example plain-language rules

1114 The following plain-language rules are to be enforced:

- 1115 Rule 1: A person, identified by his or her patient number, may read any record for which he
1116 or she is the designated patient.
- 1117 Rule 2: A person may read any record for which he or she is the designated parent or
1118 guardian, and for which the patient is under 16 years of age.
- 1119 Rule 3: A physician may write to any medical element for which he or she is the designated
1120 primary care physician, provided an email is sent to the patient.
- 1121 Rule 4: An administrator shall not be permitted to read or write to medical elements of a
1122 patient record.
- 1123 These **rules** may be written by different **PAPs** operating independently, or by a single **PAP**.

1124 4.2.4 Example XACML rule instances

1125 4.2.4.1. Rule 1

1126 Rule 1 illustrates a simple **rule** with a single <Condition> element. It also illustrates the use of
1127 the <VariableDefinition> element to define a function that may be used throughout the
1128 **policy**. The following XACML <Rule> instance expresses Rule 1:

```
1129 [a197] <?xml version="1.0" encoding="UTF-8"?>
1130 [a198] <Policy
1131 [a199] xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd" xmlns:xacml-
1132 context="urn:oasis:names:tc:xacml:2.0:context:schema:cd"
1133 [a200] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
1134 urn:oasis:names:tc:xacml:2.0:policy:schema:cd http://docs.oasis-
1135 open.org/xacml/access_control-xacml-2.0-context-schema-cd.xsd"
1136 [a201] xmlns:md="http://www.med.example.com/schemas/record.xsd"
1137 [a202] PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:1"
1138 [a203] RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1139 algorithm:deny-overrides">
1140 [a204] <PolicyDefaults>
1141 [a205] <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-
1142 19991116</XPathVersion>
1143 [a206] </PolicyDefaults>
1144 [a207] <Target/>
1145 [a208] <VariableDefinition VariableId="17590034">
1146 [a209] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1147 [a210] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1148 and-only">
1149 [a211] <SubjectAttributeDesignator
1150 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:patient-number"
1151 [a212] <DataTypes="http://www.w3.org/2001/XMLSchema#string"/>
1152 [a213] </Apply>
1153 [a214] <Apply
1154 [a215] FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1155 [a216] <AttributeSelector
1156 [a217] RequestContextPath="//xacml-context:Resource/xacml-
1157 context:ResourceContent/md:record/md:patient/md:patient-number/text()"
1158 [a218] <DataTypes="http://www.w3.org/2001/XMLSchema#string"/>
1159 [a219] </Apply>
1160 [a220] </Apply>
1161 [a221] </VariableDefinition>
1162 [a222] <Rule
1163 [a223] RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:1"
1164 [a224] Effect="Permit">
1165 [a225] <Description>
1166 [a226] A person may read any medical record in the
1167 [a227] http://www.med.example.com/schemas/record.xsd namespace
```

```

1168 [a228]     for which he or she is the designated patient
1169 [a229] </Description>
1170 [a230] <Target>
1171 [a231]   <Resources>
1172 [a232]     <Resource>
1173 [a233]       <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1174 equal">
1175 [a234]         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1176 [a235]           urn:example:med:schemas:record
1177 [a236]         </AttributeValue>
1178 [a237]         <ResourceAttributeDesignator AttributeId=
1179 [a238]           "urn:oasis:names:tc:xacml:2.0:resource:target -namespace"
1180 [a239]           DataType="http://www.w3.org/2001/XMLSchema#string"/>
1181 [a240]         </ResourceMatch>
1182 [a241]       <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-
1183 node-match">
1184 [a242]         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1185 [a243]           /md:record
1186 [a244]         </AttributeValue>
1187 [a245]         <ResourceAttributeDesignator
1188 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
1189 [a246]           DataType="http://www.w3.org/2001/XMLSchema#string"/>
1190 [a247]         </ResourceMatch>
1191 [a248]       </Resource>
1192 [a249]     </Resources>
1193 [a250]   <Actions>
1194 [a251]     <Action>
1195 [a252]       <ActionMatch
1196 [a253]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1197 [a254]           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1198 [a255]             read
1199 [a256]           </AttributeValue>
1200 [a257]           <ActionAttributeDesignator
1201 [a258]             AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1202 [a259]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1203 [a260]           </ActionMatch>
1204 [a261]         </Action>
1205 [a262]       </Actions>
1206 [a263]     </Target>
1207 [a264]   <Condition>
1208 [a265]     <VariableReference VariableId="17590034"/>
1209 [a266]   </Condition>
1210 [a267] </Rule>
1211 [a268] </Policy>

```

1212 [a199] - [a201]. XML namespace declarations.

1213 [a205] XPath expressions in the *policy* are to be interpreted according to the 1.0 version of the
1214 XPath specification.

1215 [a208] - [a221] A <VariableDefinition> element. It defines a function that evaluates the truth
1216 of the statement: the patient-number *subject attribute* is equal to the patient-number in the
1217 *resource*.

1218 [a209] The FunctionId attribute names the function to be used for comparison. In this case,
1219 comparison is done with the "urn:oasis:names:tc:xacml:1.0:function:string-equal" function; this
1220 function takes two arguments of type "http://www.w3.org/2001/XMLSchema#string".

1221 [a210] The first argument of the variable definition is a function specified by the FunctionId
1222 attribute. Since urn:oasis:names:tc:xacml:1.0:function:string-equal takes
1223 arguments of type "http://www.w3.org/2001/XMLSchema#string" and
1224 SubjectAttributeDesignator selects a *bag* of type

1225 "http://www.w3.org/2001/XMLSchema#string", "urn:oasis:names:tc:xacml:1.0:function:string-one-
1226 and-only" is used. This function guarantees that its argument evaluates to a **bag** containing exactly
1227 one value.

1228 [a211] The `SubjectAttributeDesignator` selects a **bag** of values for the `patient-number`
1229 **subject attribute** in the request **context**.

1230 [a215] The second argument of the variable definition is a function specified by the `FunctionId`
1231 attribute. Since "urn:oasis:names:tc:xacml:1.0:function:string-equal" takes arguments of type
1232 "http://www.w3.org/2001/XMLSchema#string" and the `AttributeSelector` selects a **bag** of type
1233 "http://www.w3.org/2001/XMLSchema#string", "urn:oasis:names:tc:xacml:1.0:function:string-one-
1234 and-only" is used. This function guarantees that its argument evaluates to a **bag** containing exactly
1235 one value.

1236 [a216] The `<AttributeSelector>` element selects a **bag** of values from the request **context**
1237 using a free-form XPath expression. In this case, it selects the value of the `patient-number` in
1238 the **resource**. Note that the namespace prefixes in the XPath expression are resolved with the
1239 standard XML namespace declarations.

1240 [a223] **Rule** identifier.

1241 [a224]. **Rule effect** declaration. When a **rule** evaluates to 'True' it emits the value of the `Effect`
1242 attribute. This value is then combined with the `Effect` values of other **rules** according to the **rule-**
1243 **combining algorithm**.

1244 [a225] - [a229] Free form description of the **rule**.

1245 [a230] - [a263]. A **rule target** defines a set of **decision requests** that the **rule** is intended to
1246 evaluate. In this example, the `<Subjects>` and `<Environments>` elements are omitted.

1247 [a231] - [a249] The `<Resources>` element contains a **disjunctive sequence** of `<Resource>`
1248 elements. In this example, there is just one.

1249 [a232] - [a248] The `<Resource>` element encloses the **conjunctive sequence** of
1250 `ResourceMatch` elements. In this example, there are two.

1251 [a233] - [a240] The first `<ResourceMatch>` element compares its first and second child elements
1252 according to the matching function. A match is positive if the value of the first argument matches
1253 any of the values selected by the second argument. This match compares the target namespace of
1254 the requested document with the value of "urn:example:med:schemas:record".

1255 [a233] The `MatchId` attribute names the matching function.

1256 [a235] Literal attribute value to match.

1257 [a237] - [a239] The `<ResourceAttributeDesignator>` element selects the target namespace
1258 from the resource contained in the request **context**. The **attribute** name is specified by the
1259 `AttributeId`.

1260 [a241] - [a247] The second `<ResourceMatch>` element. This match compares the results of two
1261 XPath expressions. The second XPath expression is the location path to the requested XML
1262 element and the first XPath expression is the literal value `"/md:record"`. The "xpath-node-match"
1263 function evaluates to "True" if the requested XML element is below the `"/md:record"` element.

1264 [a250] - [a262] The `<Actions>` element contains a **disjunctive sequence** of `<Action>` elements.
1265 In this case, there is just one `<Action>` element.

1266 [a251] - [a261] The `<Action>` element contains a **conjunctive sequence** of `<ActionMatch>`
1267 elements. In this case, there is just one `<ActionMatch>` element.

1268 [a252] - [a260] The <ActionMatch> element compares its first and second child elements
1269 according to the matching function. The match is positive if the value of the first argument matches
1270 any of the values selected by the second argument. In this case, the value of the action-id
1271 action attribute in the request **context** is compared with the literal value "read".

1272 [a264] - [a266] The <Condition> element. A **condition** must evaluate to "True" for the **rule** to be
1273 applicable. This **condition** contains a reference to a variable definition defined elsewhere in the
1274 **policy**.

1275 4.2.4.2. Rule 2

1276 Rule 2 illustrates the use of a mathematical function, i.e. the <Apply> element with functionId
1277 "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" to calculate the date of the
1278 patient's sixteenth birthday. It also illustrates the use of **predicate** expressions, with the
1279 functionId "urn:oasis:names:tc:xacml:1.0:function:and". This example has one function
1280 embedded in the <Condition> element and another one referenced in a
1281 <VariableDefinition> element.

```
1282 [a269] <?xml version="1.0" encoding="UTF-8" ?>
1283 [a270] <Policy
1284 [a271] xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd" xmlns:xacml-
1285 context="urn:oasis:names:tc:xacml:2.0:context:schema:cd"
1286 [a272] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1287 xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:cd
1288 http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-cd.xsd"
1289 [a273] xmlns:xf="http://www.w3.org/TR/2002/WD-xquery-operators-20020816/#"
1290 [a274] xmlns:md="http://www.med.example.com/schemas/record.xsd"
1291 [a275] PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:2"
1292 RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
1293 overrides">
1294 [a276] <PolicyDefaults>
1295 [a277] <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-
1296 19991116</XPathVersion>
1297 [a278] </PolicyDefaults>
1298 [a279] <Target/>
1299 [a280] <VariableDefinition VariableId="17590035">
1300 [a281] <Apply FunctionId="urn:oasis:names:tc:xacml:2.0:function:date-less-or-
1301 equal">
1302 [a282] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-
1303 only">
1304 [a283] <EnvironmentAttributeDesignator
1305 [a284] AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date"
1306 [a285] DataType="http://www.w3.org/2001/XMLSchema#date"/>
1307 [a286] </Apply>
1308 [a287] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-
1309 yearMonthDuration">
1310 [a288] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-
1311 only">
1312 [a289] <AttributeSelector RequestContextPath=
1313 [a290] "//md:record/md:patient/md:patientDoB/text()"
1314 [a291] DataType="http://www.w3.org/2001/XMLSchema#date"/>
1315 [a292] </Apply>
1316 [a293] <AttributeValue
1317 [a294] DataType="http://www.w3.org/TR/2002/WD-xquery-operators-
1318 20020816#yearMonthDuration">
1319 [a295] <xf:dt-yearMonthDuration>
1320 [a296] P16Y
1321 [a297] </xf:dt-yearMonthDuration>
1322 [a298] </AttributeValue>
1323 [a299] </Apply>
1324 [a300] </Apply>
```

```

1325 [a301] </VariableDefinition>
1326 [a302] <Rule
1327 [a303] RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:2"
1328 [a304] Effect="Permit">
1329 [a305] <Description>
1330 [a306]   A person may read any medical record in the
1331 [a307]   http://www.med.example.com/records.xsd namespace
1332 [a308]   for which he or she is the designated parent or guardian,
1333 [a309]   and for which the patient is under 16 years of age
1334 [a310] </Description>
1335 [a311] <Target>
1336 [a312] <Resources>
1337 [a313] <Resource>
1338 [a314] <ResourceMatch
1339 [a315]   MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1340 [a316] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1341 [a317]   http://www.med.example.com/schemas/record.xsd
1342 [a318] </AttributeValue>
1343 [a319] <ResourceAttributeDesignator AttributeId=
1344 "urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1345 [a320]   DataType="http://www.w3.org/2001/XMLSchema#string"/>
1346 [a321] </ResourceMatch>
1347 [a322] <ResourceMatch
1348 [a323]   MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1349 [a324] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1350 [a325]   /md:record
1351 [a326] </AttributeValue>
1352 [a327] <ResourceAttributeDesignator
1353 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
1354 [a328]   DataType="http://www.w3.org/2001/XMLSchema#string"/>
1355 [a329] </ResourceMatch>
1356 [a330] </Resource>
1357 [a331] </Resources>
1358 [a332] <Actions>
1359 [a333] <Action>
1360 [a334] <ActionMatch
1361 [a335]   MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1362 [a336] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1363 [a337]   read
1364 [a338] </AttributeValue>
1365 [a339] <ActionAttributeDesignator
1366 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1367 [a340]   DataType="http://www.w3.org/2001/XMLSchema#string"/>
1368 [a341] </ActionMatch>
1369 [a342] </Action>
1370 [a343] </Actions>
1371 [a344] </Target>
1372 [a345] <Condition>
1373 [a346] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
1374 [a347] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1375 [a348] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1376 and-only">
1377 [a349] <SubjectAttributeDesignator
1378 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:
1379 [a350] parent-guardian-id"
1380 [a351]   DataType="http://www.w3.org/2001/XMLSchema#string"/>
1381 [a352] </Apply>
1382 [a353] <Apply
1383 [a354]   FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-
1384 only">
1385 [a355] <AttributeSelector
1386 [a356]   RequestContextPath="//xacml-context:Resource/xacml-
1387 context:ResourceContent/md:record/md:parentGuardian/md:parentGuardianId/text()"

```

1388 [a357] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1389 [a358] </Apply>
1390 [a359] </Apply>
1391 [a360] <VariableReference VariableId="17590035"/>
1392 [a361] </Apply>
1393 [a362] </Condition>
1394 [a363] </Rule>
1395 [a364] </Policy>

1396 [a280] - [a301] The <VariableDefinition> element contains part of the **condition** (i.e. is the
1397 patient under 16 years of age?). The patient is under 16 years of age if the current date is less than
1398 the date computed by adding 16 to the patient's date of birth.

1399 [a281] - [a300] "urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal" is used to compute the
1400 difference of two date arguments.

1401 [a282] - [a286] The first date argument uses "urn:oasis:names:tc:xacml:1.0:function:date-one-and-
1402 only" to ensure that the **bag** of values selected by its argument contains exactly one value of type
1403 "http://www.w3.org/2001/XMLSchema#date".

1404 [a284] The current date is evaluated by selecting the
1405 "urn:oasis:names:tc:xacml:1.0:environment:current-date" **environment attribute**.

1406 [a287] - [a299] The second date argument uses "urn:oasis:names:tc:xacml:1.0:function:date-add-
1407 yearMonthDuration" to compute the date of the patient's sixteenth birthday by adding 16 years to
1408 the patient's date of birth. The first of its arguments is of type
1409 "http://www.w3.org/2001/XMLSchema#date" and the second is of type
1410 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration".

1411 [a289] The <AttributeSelector> element selects the patient's date of birth by taking the XPath
1412 expression over the **resource** content.

1413 [a293] - [a298] Year Month Duration of 16 years.

1414 [a311] - [a344] **Rule** declaration and **rule target**. See Rule 1 in Section 4.2.4.1 for the detailed
1415 explanation of these elements.

1416 [a345] - [a362] The <Condition> element. The **condition** must evaluate to "True" for the **rule** to
1417 be applicable. This **condition** evaluates the truth of the statement: the requestor is the designated
1418 parent or guardian and the patient is under 16 years of age. It contains one embedded <Apply>
1419 element and one referenced <VariableDefinition> element.

1420 [a346] The **condition** uses the "urn:oasis:names:tc:xacml:1.0:function:and" function. This is a
1421 Boolean function that takes one or more Boolean arguments (2 in this case) and performs the
1422 logical "AND" operation to compute the truth value of the expression.

1423 [a347] - [a359] The first part of the **condition** is evaluated (i.e. is the requestor the designated
1424 parent or guardian?). The function is "urn:oasis:names:tc:xacml:1.0:function:string-equal" and it
1425 takes two arguments of type "http://www.w3.org/2001/XMLSchema#string".

1426 [a348] designates the first argument. Since "urn:oasis:names:tc:xacml:1.0:function:string-equal"
1427 takes arguments of type "http://www.w3.org/2001/XMLSchema#string",
1428 "urn:oasis:names:tc:xacml:1.0:function:string-one-and-only" is used to ensure that the **subject**
1429 **attribute** "urn:oasis:names:tc:xacml:2.0:example:attribute:parent-guardian-id" in the request
1430 **context** contains exactly one value.

1431 [a353] designates the second argument. The value of the **subject attribute**
1432 "urn:oasis:names:tc:xacml:2.0:example:attribute:parent-guardian-id" is selected from the request
1433 **context** using the <SubjectAttributeDesignator> element.

1434 [a354] As above, the “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used to ensure
1435 that the **bag** of values selected by it’s argument contains exactly one value of type
1436 “http://www.w3.org/2001/XMLSchema#string”.

1437 [a355] The second argument selects the value of the <md:parentGuardianId> element from the
1438 **resource** content using the <AttributeSelector> element. This element contains a free-form
1439 XPath expression, pointing into the request **context**. Note that all namespace prefixes in the XPath
1440 expression are resolved with standard namespace declarations. The AttributeSelector
1441 evaluates to the **bag** of values of type “http://www.w3.org/2001/XMLSchema#string”.

1442 [a360] references the <VariableDefinition> element, where the second part of the **condition**
1443 is defined.

1444 4.2.4.3. Rule 3

1445 Rule 3 illustrates the use of an **obligation**. The XACML <Rule> element syntax does not include
1446 an element suitable for carrying an **obligation**, therefore Rule 3 has to be formatted as a
1447 <Policy> element.

```
1448 [a365] <?xml version="1.0" encoding="UTF-8"?>
1449 [a366] <Policy
1450 [a367]   xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd" xmlns:xacml-
1451 [a368]   context="urn:oasis:names:tc:xacml:2.0:context:schema:cd"
1452 [a369]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1453 [a369]   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:cd
1454 [a370] http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-cd.xsd"
1455 [a370]   xmlns:md="http://www.med.example.com/schemas/record.xsd"
1456 [a371]   PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:3"
1457 [a372]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1458 [a373]   algorithm:deny-overrides">
1459 [a373]   <Description>
1460 [a374]     Policy for any medical record in the
1461 [a375]     http://www.med.example.com/schemas/record.xsd namespace
1462 [a376]   </Description>
1463 [a377]   <PolicyDefaults>
1464 [a378]     <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-
1465 [a379]     19991116</XPathVersion>
1466 [a379]   </PolicyDefaults>
1467 [a380]   <Target>
1468 [a381]     <Resources>
1469 [a382]       <Resource>
1470 [a383]         <ResourceMatch
1471 [a384]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1472 [a385]             <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1473 [a386]               urn:example:med:schemas:record
1474 [a387]             </AttributeValue>
1475 [a388]             <ResourceAttributeDesignator AttributeId=
1476 [a389]               "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1477 [a390]               DataType="http://www.w3.org/2001/XMLSchema#string" />
1478 [a391]             </ResourceMatch>
1479 [a392]           </Resource>
1480 [a393]         </Resources>
1481 [a394]       </Target>
1482 [a395]     <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:3"
1483 [a396]       Effect="Permit">
1484 [a397]       <Description>
1485 [a398]         A physician may write any medical element in a record
1486 [a399]         for which he or she is the designated primary care
1487 [a400]         physician, provided an email is sent to the patient
1488 [a401]       </Description>
1489 [a402]     </Rule>
1489 [a402]   </Target>
```

```

1490 [a403] <Subjects>
1491 [a404] <Subject>
1492 [a405] <SubjectMatch
1493 [a406] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1494 [a407] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1495 [a408] physician
1496 [a409] </AttributeValue>
1497 [a410] <SubjectAttributeDesignator AttributeId=
1498 "urn:oasis:names:tc:xacml:2.0:example:attribute:role"
1499 [a411] DataType="http://www.w3.org/2001/XMLSchema#string" />
1500 [a412] </SubjectMatch>
1501 [a413] </Subject>
1502 [a414] </Subjects>
1503 [a415] <Resources>
1504 [a416] <Resource>
1505 [a417] <ResourceMatch
1506 [a418] MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1507 [a419] <AttributeValue
1508 [a420] DataType="http://www.w3.org/2001/XMLSchema#string">
1509 [a421] /md:record/md:medical
1510 [a422] </AttributeValue>
1511 [a423] <ResourceAttributeDesignator
1512 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
1513 [a424] DataType="http://www.w3.org/2001/XMLSchema#string" />
1514 [a425] </ResourceMatch>
1515 [a426] </Resource>
1516 [a427] </Resources>
1517 [a428] <Actions>
1518 [a429] <Action>
1519 [a430] <ActionMatch
1520 [a431] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1521 [a432] <AttributeValue
1522 [a433] DataType="http://www.w3.org/2001/XMLSchema#string">
1523 [a434] write
1524 [a435] </AttributeValue>
1525 [a436] <ActionAttributeDesignator
1526 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1527 [a437] DataType="http://www.w3.org/2001/XMLSchema#string" />
1528 [a438] </ActionMatch>
1529 [a439] </Action>
1530 [a440] </Actions>
1531 [a441] </Target>
1532 [a442] <Condition>
1533 [a443] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1534 [a444] <Apply
1535 [a445] FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1536 [a446] <SubjectAttributeDesignator
1537 [a447] AttributeId="urn:oasis:names:tc:xacml:2.0:example:
1538 attribute:physician-id"
1539 [a448] DataType="http://www.w3.org/2001/XMLSchema#string" />
1540 [a449] </Apply>
1541 [a450] <Apply
1542 [a451] FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1543 [a452] <AttributeSelector RequestContextPath=
1544 [a453] "//xacml-context:Resource/xacml-
1545 context:ResourceContent/md:record/md:primaryCarePhysician/md:registrationID/text(
1546 )"
1547 [a454] DataType="http://www.w3.org/2001/XMLSchema#string" />
1548 [a455] </Apply>
1549 [a456] </Apply>
1550 [a457] </Condition>
1551 [a458] </Rule>
1552 [a459] <Obligations>

```

```

1553 [a460] <Obligation
1554 ObligationId="urn:oasis:names:tc:xacml:example:obligation:email"
1555 [a461] FulfillOn="Permit">
1556 [a462] <AttributeAssignment
1557 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:mailto"
1558 [a463] DataType="http://www.w3.org/2001/XMLSchema#string">
1559 [a464] &lt;AttributeSelector RequestContextPath=
1560 [a465] "/md:/record/md:patient/md:patientContact/md:email"
1561 [a466] DataType="http://www.w3.org/2001/XMLSchema#string"/ &gt; ;
1562 [a467] </AttributeAssignment>
1563 [a468] <AttributeAssignment
1564 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
1565 [a469] DataType="http://www.w3.org/2001/XMLSchema#string">
1566 [a470] Your medical record has been accessed by:
1567 [a471] </AttributeAssignment>
1568 [a472] <AttributeAssignment
1569 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
1570 [a473] DataType="http://www.w3.org/2001/XMLSchema#string">
1571 [a474] &lt;SubjectAttributeDesignator
1572 AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
1573 [a475] DataType="http://www.w3.org/2001/XMLSchema#string"/ &gt; ;
1574 [a476] </AttributeAssignment>
1575 [a477] </Obligation>
1576 [a478] </Obligations>
1577 [a479] </Policy>

```

1578 [a366] - [a372] The <Policy> element includes standard namespace declarations as well as policy
1579 specific parameters, such as PolicyId and RuleCombiningAlgId.

1580 [a371] **Policy** identifier. This parameter allows the **policy** to be referenced by a **policy set**.

1581 [a372] The **Rule combining algorithm** identifies the algorithm for combining the outcomes of **rule**
1582 evaluation.

1583 [a373] - [a376] Free-form description of the **policy**.

1584 [a379] - [a394] **Policy target**. The **policy target** defines a set of applicable decision requests. The
1585 structure of the <Target> element in the <Policy> is identical to the structure of the <Target>
1586 element in the <Rule>. In this case, the **policy target** is the set of all XML resources that conform
1587 to the namespace "urn:example:med:schemas:record".

1588 [a395] The only <Rule> element included in this <Policy>. Two parameters are specified in the
1589 **rule** header: RuleId and Effect.

1590 [a402] - [a441] The **rule target** further constrains the **policy target**.

1591 [a405] - [a412] The <SubjectMatch> element targets the **rule** at **subjects** whose
1592 "urn:oasis:names:tc:xacml:2.0:example:attribute:role" **subject attribute** is equal to "physician".

1593 [a417] - [a425] The <ResourceMatch> element targets the **rule** at **resources** that match the
1594 XPath expression "/md:record/md:medical".

1595 [a430] - [a438] The <ActionMatch> element targets the **rule** at **actions** whose
1596 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** is equal to "write".

1597 [a442] - [a457] The <Condition> element. For the **rule** to be applicable to the **decision request**,
1598 the **condition** must evaluate to "True". This **condition** compares the value of the
1599 "urn:oasis:names:tc:xacml:2.0:example:attribute:physician-id" **subject attribute** with the value of
1600 the <registrationId> element in the medical record that is being accessed.

1601 [a459] - [a478] The <Obligations> element. **Obligations** are a set of operations that must be
1602 performed by the **PEP** in conjunction with an **authorization decision**. An **obligation** may be

1603 associated with a "Permit" or "Deny" **authorization decision**. The element contains a single
1604 **obligation**.

1605 [a460] - [a477] The <Obligation> element consists of the ObligationId attribute, the
1606 **authorization decision** value for which it must be fulfilled, and a set of **attribute** assignments. The
1607 **PDP** does not resolve the attribute assignments. This is the job of the **PEP**.

1608 [a460] The ObligationId attribute identifies the **obligation**. In this case, the **PEP** is required to
1609 send email.

1610 [a461] The FulfillOn attribute defines the **authorization decision** value for which this
1611 **obligation** must be fulfilled. In this case, when access is permitted.

1612 [a462] - [a467] The first parameter indicates where the **PEP** will find the email address in the
1613 resource.

1614 [a468] - [a471] The second parameter contains literal text for the email body.

1615 [a472] - [a476] The third parameter indicates where the **PEP** will find further text for the email body
1616 in the resource.

1617 4.2.4.4. Rule 4

1618 Rule 4 illustrates the use of the "Deny" Effect value, and a <Rule> with no <Condition>
1619 element.

```

1620 [a480] <?xml version="1.0" encoding="UTF-8"?>
1621 [a481] <Policy
1622 [a482] xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd"
1623 [a483] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1624 [a484] xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:cd
1625 [a484] http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-cd.xsd"
1626 [a484] xmlns:md="http://www.med.example.com/schemas/record.xsd"
1627 [a485] PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:4"
1628 [a486] RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1629 algorithm:deny-overrides">
1630 [a487] <PolicyDefaults>
1631 [a488] <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-
1632 [a488] 19991116</XPathVersion>
1633 [a489] </PolicyDefaults>
1634 [a490] <Target/>
1635 [a491] <Rule
1636 [a492] RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:4"
1637 [a493] Effect="Deny">
1638 [a494] <Description>
1639 [a495] An Administrator shall not be permitted to read or write
1640 [a496] medical elements of a patient record in the
1641 [a497] http://www.med.example.com/records.xsd namespace.
1642 [a498] </Description>
1643 [a499] <Target>
1644 [a500] <Subjects>
1645 [a501] <Subject>
1646 [a502] <SubjectMatch
1647 [a503] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1648 [a504] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1649 [a505] administrator
1650 [a506] </AttributeValue>
1651 [a507] <SubjectAttributeDesignator AttributeId=
1652 [a508] "urn:oasis:names:tc:xacml:2.0:example:attribute:role"
1653 [a509] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1654 [a510] </SubjectMatch>
1655 [a511] </Subject>

```



```

1656 [a512] </Subjects>
1657 [a513] <Resources>
1658 [a514] <Resource>
1659 [a515] <ResourceMatch
1660 [a516] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1661 [a517] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1662 [a518] urn:example:med:schemas:record
1663 [a519] </AttributeValue>
1664 [a520] <ResourceAttributeDesignator
1665 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1666 [a521] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1667 [a522] </ResourceMatch>
1668 [a523] <ResourceMatch
1669 [a524] MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1670 [a525] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1671 [a526] /md:record/md:medical
1672 [a527] </AttributeValue>
1673 [a528] <ResourceAttributeDesignator
1674 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
1675 [a529] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1676 [a530] </ResourceMatch>
1677 [a531] </Resource>
1678 [a532] </Resources>
1679 [a533] <Actions>
1680 [a534] <Action>
1681 [a535] <ActionMatch
1682 [a536] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1683 [a537] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1684 [a538] read
1685 [a539] </AttributeValue>
1686 [a540] <ActionAttributeDesignator
1687 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1688 [a541] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1689 [a542] </ActionMatch>
1690 [a543] </Action>
1691 [a544] <Action>
1692 [a545] <ActionMatch
1693 [a546] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1694 [a547] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1695 [a548] write
1696 [a549] </AttributeValue>
1697 [a550] <ActionAttributeDesignator
1698 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1699 [a551] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1700 [a552] </ActionMatch>
1701 [a553] </Action>
1702 [a554] </Actions>
1703 [a555] </Target>
1704 [a556] </Rule>
1705 [a557] </Policy>

```

1706 [a492] - [a493] The <Rule> element declaration.

1707 [a493] **Rule** Effect. Every **rule** that evaluates to “True” emits the **rule effect** as its value. This
1708 **rule** Effect is “Deny” meaning that according to this **rule**, access must be denied when it
1709 evaluates to “True”.

1710 [a494] - [a498] Free form description of the **rule**.

1711 [a499] - [a555] **Rule target**. The **Rule target** defines the set of **decision requests** that are
1712 applicable to the **rule**.

1713 [a502] - [a510] The <SubjectMatch> element targets the **rule** at **subjects** whose
 1714 "urn:oasis:names:tc:xacml:2.0:example:attribute:role" **subject attribute** is equal to
 1715 "administrator".

1716 [a513] - [a532] The <Resources> element contains one <Resource> element, which (in turn)
 1717 contains two <ResourceMatch> elements. The **target** matches if the **resource** identified by the
 1718 request **context** matches both **resource** match criteria.

1719 [a558] [a515]-[a522] The first <ResourceMatch> element targets the **rule** at
 1720 **resources** whose "urn:oasis:names:tc:xacml:2.0:resource:target-namespace" **resource**
 1721 **attribute** is equal to "urn:example:med:schemas:record".

1722 [a523] - [a530] The second <ResourceMatch> element targets the **rule** at XML elements that
 1723 match the XPath expression "/md:record/md:medical".

1724 [a533] - [a554] The <Actions> element contains two <Action> elements, each of which contains
 1725 one <ActionMatch> element. The **target** matches if the **action** identified in the request **context**
 1726 matches either of the **action** match criteria.

1727 [a535] - [a552] The <ActionMatch> elements target the **rule** at **actions** whose
 1728 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** is equal to "read" or "write".

1729 This **rule** does not have a <Condition> element.

4.2.4.5. Example PolicySet

1731 This section uses the examples of the previous sections to illustrate the process of combining
 1732 **policies**. The policy governing read access to medical elements of a record is formed from each of
 1733 the four **rules** described in Section 4.2.3. In plain language, the combined rule is:

- 1734 • Either the requestor is the patient; or
- 1735 • the requestor is the parent or guardian and the patient is under 16; or
- 1736 • the requestor is the primary care physician and a notification is sent to the patient; and
- 1737 • the requestor is not an administrator.

1738 The following **policy set** illustrates the combined **policies**. **Policy 3** is included by reference and
 1739 **policy 2** is explicitly included.

```
[a559] <?xml version="1.0" encoding="UTF-8"?>
[a560] <PolicySet
[a561]   xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd"
[a562]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:cd
http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-cd.xsd"
[a563]   PolicySetId=
[a564]     "urn:oasis:names:tc:xacml:2.0:example:policysetid:1"
[a565]   PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
[a566]   policy-combining-algorithm:deny-overrides">
[a567]     <Description>
[a568]       Example policy set.
[a569]     </Description>
[a570]     <Target>
[a571]       <Resources>
[a572]         <Resource>
[a573]           <ResourceMatch
[a574]             MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[a575]               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
[a576]                 urn:example:med:schema:records
[a577]               </AttributeValue>
```

```

[a578]         <ResourceAttributeDesignator AttributeId=
[a579]           "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
[a580]           DataType="http://www.w3.org/2001/XMLSchema#string" />
[a581]         </ResourceMatch>
[a582]       </Resource>
[a583]     </Resources>
[a584]   </Target>
[a585] <PolicyIdReference>
[a586]   urn:oasis:names:tc:xacml:2.0:example:policyid:3
[a587] </PolicyIdReference>
[a588] <Policy>
[a589]   PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:2"
[a590]   RuleCombiningAlgId=
[a591] "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
[a592]   <Target />
[a593]   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:1"
[a594]     Effect="Permit">
[a595]   </Rule>
[a596]   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:2"
[a597]     Effect="Permit">
[a598]   </Rule>
[a599]   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:4"
[a600]     Effect="Deny">
[a601]   </Rule>
[a602] </Policy>
[a603] </PolicySet>

```

1740

1741 [a560] - [a566] The <PolicySet> element declaration. Standard XML namespace declarations
1742 are included.

1743 [a563] The PolicySetId attribute is used for identifying this **policy set** for possible inclusion in
1744 another **policy set**.

1745 [a565] The **policy combining algorithm** identifier. **Policies** and **policy sets** in this **policy set** are
1746 combined according to the specified **policy combining algorithm** when the **authorization**
1747 **decision** is computed.

1748 [a567] - [a569] Free form description of the **policy set**.

1749 [a570] - [a584] The **policy set** <Target> element defines the set of **decision requests** that are
1750 applicable to this <PolicySet> element.

1751 [a585] PolicyIdReference includes a **policy** by id.

1752 [a589] Policy 2 is explicitly included in this **policy set**. The **rules** in Policy 2 are omitted for
1753 clarity.

1754 5. Policy syntax (normative, with the exception of 1755 the schema fragments)

1756 5.1. Element <PolicySet>

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

1760 <PolicySetIdReference> element. **Policies** MAY be included in an enclosing <PolicySet>
1761 element either directly using the <Policy> element or indirectly using the
1762 <PolicyIdReference> element.

1763 A <PolicySet> element MAY be evaluated, in which case the evaluation procedure defined in
1764 Section 7.11 SHALL be used.

1765 If a <PolicySet> element contains references to other **policy sets** or **policies** in the form of
1766 URLs, then these references MAY be resolvable.

1767 **Policy sets** and **policies** included in a <PolicySet> element MUST be combined using the
1768 algorithm identified by the PolicyCombiningAlgId attribute. <PolicySet> is treated exactly
1769 like a <Policy> in all **policy combining algorithms**.

1770 The <Target> element defines the applicability of the <PolicySet> element to a set of **decision**
1771 **requests**. If the <Target> element within the <PolicySet> element matches the **request**
1772 **context**, then the <PolicySet> element MAY be used by the **PDP** in making its **authorization**
1773 **decision**. See Section 7.11.

1774 The <Obligations> element contains a set of **obligations** that MUST be fulfilled by the **PEP** in
1775 conjunction with the **authorization decision**. If the **PEP** does not understand, or cannot fulfill, any
1776 of the **obligations**, then it MUST act as if the **PDP** had returned a "Deny" **authorization decision**
1777 value. See Section 7.14.

```
1778 <xs:element name="PolicySet" type="xacml:PolicySetType"/>  
1779 <xs:complexType name="PolicySetType">  
1780   <xs:sequence>  
1781     <xs:element ref="xacml:Description" minOccurs="0"/>  
1782     <xs:element ref="xacml:PolicySetDefaults" minOccurs="0"/>  
1783     <xs:element ref="xacml:Target"/>  
1784     <xs:choice minOccurs="0" maxOccurs="unbounded">  
1785       <xs:element ref="xacml:PolicySet"/>  
1786       <xs:element ref="xacml:Policy"/>  
1787       <xs:element ref="xacml:PolicySetIdReference"/>  
1788       <xs:element ref="xacml:PolicyIdReference"/>  
1789       <xs:element ref="xacml:CombinerParameters"/>  
1790       <xs:element ref="xacml:PolicyCombinerParameters"/>  
1791       <xs:element ref="xacml:PolicySetCombinerParameters"/>  
1792     </xs:choice>  
1793     <xs:element ref="xacml:Obligations" minOccurs="0"/>  
1794   </xs:sequence>  
1795   <xs:attribute name="PolicySetId" type="xs:anyURI" use="required"/>  
1796   <xs:attribute name="Version" type="xacml:VersionType" default="1.0"/>  
1797   <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI" use="required"/>  
1798 </xs:complexType>
```

1799 The <PolicySet> element is of **PolicySetType** complex type.

1800 The <PolicySet> element contains the following attributes and elements:

1801 PolicySetId [Required]

1802 **Policy set** identifier. It is the responsibility of the **PAP** to ensure that no two **policies**
1803 visible to the **PDP** have the same identifier. This MAY be achieved by following a
1804 predefined URN or URI scheme. If the **policy set** identifier is in the form of a URL, then it
1805 MAY be resolvable.

1806 Version [Default 1.0]

1807 The version number of the **PolicySet**.

1808 PolicyCombiningAlgId [Required]

1809 The identifier of the **policy-combining algorithm** by which the <PolicySet>,
1810 <CombinerParameters>, <PolicyCombinerParameters> and
1811 <PolicySetCombinerParameters> components MUST be combined. Standard
1812 **policy-combining algorithms** are listed in Appendix C. Standard **policy-combining**
1813 **algorithm** identifiers are listed in Section B.10.

1814 <Description> [Optional]

1815 A free-form description of the **policy set**.

1816 <PolicySetDefaults> [Optional]

1817 A set of default values applicable to the **policy set**. The scope of the
1818 <PolicySetDefaults> element SHALL be the enclosing **policy set**.

1819 <Target> [Required]

1820 The <Target> element defines the applicability of a **policy set** to a set of **decision**
1821 **requests**.

1822 The <Target> element MAY be declared by the creator of the <PolicySet> or it MAY be
1823 computed from the <Target> elements of the referenced <Policy> elements, either as
1824 an intersection or as a union.

1825 <PolicySet> [Any Number]

1826 A **policy set** that is included in this **policy set**.

1827 <Policy> [Any Number]

1828 A **policy** that is included in this **policy set**.

1829 <PolicySetIdReference> [Any Number]

1830 A reference to a **policy set** that MUST be included in this **policy set**. If
1831 <PolicySetIdReference> is a URL, then it MAY be resolvable.

1832 <PolicyIdReference> [Any Number]

1833 A reference to a **policy** that MUST be included in this **policy set**. If the
1834 <PolicyIdReference> is a URL, then it MAY be resolvable.

1835 <Obligations> [Optional]

1836 Contains the set of <Obligation> elements. See Section 7.14 for a description of how
1837 the set of **obligations** to be returned by the **PDP** shall be determined.

1838 <CombinerParameters> [Optional]

1839 Contains a sequence of <CombinerParameter> elements.

1840 <PolicyCombinerParameters> [Optional]

1841 Contains a sequence of <CombinerParameter> elements that are associated with a
1842 particular <Policy> or <PolicyIdReference> element within the <PolicySet>.

1843 <PolicySetCombinerParameters> [Optional]

1844 Contains a sequence of <CombinerParameter> elements that are associated with a
1845 particular <PolicySet> or <PolicySetIdReference> element within the
1846 <PolicySet>.

1847 **5.2. Element <Description>**

1848 The <Description> element contains a free-form description of the <PolicySet>, <Policy>
1849 or <Rule> element. The <Description> element is of **xs:string** simple type.

```
1850 <xs:element name="Description" type="xs:string"/>
```

1851 **5.3. Element <PolicySetDefaults>**

1852 The <PolicySetDefaults> element SHALL specify default values that apply to the
1853 <PolicySet> element.

```
1854 <xs:element name="PolicySetDefaults" type="xacml:DefaultsType"/>  
1855 <xs:complexType name="DefaultsType">  
1856 <xs:sequence>  
1857 <xs:choice>  
1858 <xs:element ref="xacml:XPathVersion" minOccurs="0"/>  
1859 </xs:choice>  
1860 </xs:sequence>  
1861 </xs:complexType>
```

1862 <PolicySetDefaults> element is of **DefaultsType** complex type.

1863 The <PolicySetDefaults> element contains the following elements:

1864 <XPathVersion> [Optional]

1865 Default XPath version.

1866 **5.4. Element <XPathVersion>**

1867 The <XPathVersion> element SHALL specify the version of the XPath specification to be used by
1868 <AttributeSelector> elements and XPath-based functions in the **policy set** or **policy**.

```
1869 <xs:element name="XPathVersion" type="xs:anyURI"/>
```

1870 The URI for the XPath 1.0 specification is "http://www.w3.org/TR/1999/Rec-xpath-19991116". The
1871 <XPathVersion> element is REQUIRED if the XACML enclosing **policy set** or **policy** contains
1872 <AttributeSelector> elements or XPath-based functions.

1873 **5.5. Element <Target>**

1874 The <Target> element identifies the set of **decision requests** that the parent element is intended
1875 to evaluate. The <Target> element SHALL appear as a child of a <PolicySet> and <Policy>
1876 element and MAY appear as a child of a <Rule> element. It contains definitions for **subjects**,
1877 **resources**, **actions** and **environments**.

1878 The <Target> element SHALL contain a **conjunctive sequence** of <Subjects>, <Resources>
1879 <Actions> and <Environments> elements. For the parent of the <Target> element to be
1880 applicable to the **decision request**, there MUST be at least one positive match between each
1881 section of the <Target> element and the corresponding section of the <xacml-
1882 context:Request> element.

```
1883 <xs:element name="Target" type="xacml:TargetType"/>  
1884 <xs:complexType name="TargetType">  
1885 <xs:sequence>  
1886 <xs:element ref="xacml:Subjects" minOccurs="0"/>  
1887 <xs:element ref="xacml:Resources" minOccurs="0"/>  
1888 <xs:element ref="xacml:Actions" minOccurs="0"/>  
1889 <xs:element ref="xacml:Environments" minOccurs="0"/>
```

1890 `</xs:sequence>`
 1891 `</xs:complexType>`

1892 The `<Target>` element is of **TargetType** complex type.

1893 The `<Target>` element contains the following elements:

1894 `<Subjects>` [Optional]

1895 Matching specification for the **subject attributes** in the **context**. If this element is missing,
 1896 then the **target** SHALL match all **subjects**.

1897 `<Resources>` [Optional]

1898 Matching specification for the **resource attributes** in the **context**. If this element is
 1899 missing, then the **target** SHALL match all **resources**.

1900 `<Actions>` [Optional]

1901 Matching specification for the **action attributes** in the **context**. If this element is missing,
 1902 then the **target** SHALL match all **actions**.

1903 `<Environments>` [Optional]

1904 Matching specification for the **environment attributes** in the **context**. If this element is
 1905 missing, then the **target** SHALL match all **environments**.

5.6. Element `<Subjects>`

1906 The `<Subjects>` element SHALL contain a **disjunctive sequence** of `<Subject>` elements.

```
1908 <xs:element name="Subjects" type="xacml:SubjectsType" />
1909 <xs:complexType name="SubjectsType">
1910   <xs:sequence>
1911     <xs:element ref="xacml:Subject" maxOccurs="unbounded" />
1912   </xs:sequence>
1913 </xs:complexType>
```

1914 The `<Subjects>` element is of **SubjectsType** complex type.

1915 The `<Subjects>` element contains the following elements:

1916 `<Subject>` [One to Many, Required]

1917 See Section 5.7.

5.7. Element `<Subject>`

1918 The `<Subject>` element SHALL contain a **conjunctive sequence** of `<SubjectMatch>`
 1919 elements.

```
1921 <xs:element name="Subject" type="xacml:SubjectType" />
1922 <xs:complexType name="SubjectType">
1923   <xs:sequence>
1924     <xs:element ref="xacml:SubjectMatch" maxOccurs="unbounded" />
1925   </xs:sequence>
1926 </xs:complexType>
```

1927 The `<Subject>` element is of **SubjectType** complex type.

1928 The `<Subject>` element contains the following elements:

1929 `<SubjectMatch>` [One to Many]

1930 A **conjunctive sequence** of individual matches of the **subject attributes** in the request
1931 **context** and the embedded **attribute** values. See Section 5.8.

1932 **5.8. Element <SubjectMatch>**

1933 The <SubjectMatch> element SHALL identify a set of **subject**-related entities by matching
1934 **attribute** values in a <xacml-context:Subject> element of the request **context** with the
1935 embedded **attribute** value.

```
1936 <xs:element name="SubjectMatch" type="xacml:SubjectMatchType"/>  
1937 <xs:complexType name="SubjectMatchType">  
1938   <xs:sequence>  
1939     <xs:element ref="xacml:AttributeValue"/>  
1940     <xs:choice>  
1941       <xs:element ref="xacml:SubjectAttributeDesignator"/>  
1942       <xs:element ref="xacml:AttributeSelector"/>  
1943     </xs:choice>  
1944   </xs:sequence>  
1945   <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>  
1946 </xs:complexType>
```

1947 The <SubjectMatch> element is of **SubjectMatchType** complex type.

1948 The <SubjectMatch> element contains the following attributes and elements:

1949 MatchId [Required]

1950 Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI** with
1951 legal values documented in Section 7.5.

1952 <xacml:AttributeValue> [Required]

1953 Embedded attribute value.

1954 <SubjectAttributeDesignator> [Required choice]

1955 MAY be used to identify one or more **attribute** values in a <Subject> element of the
1956 request **context**.

1957 <AttributeSelector> [Required choice]

1958 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
1959 expression SHOULD resolve to an **attribute** in a <Subject> element of the request
1960 **context**.

1961 **5.9. Element <Resources>**

1962 The <Resources> element SHALL contain a **disjunctive sequence** of <Resource> elements.

```
1963 <xs:element name="Resources" type="xacml:ResourcesType"/>  
1964 <xs:complexType name="ResourcesType">  
1965   <xs:sequence>  
1966     <xs:element ref="xacml:Resource" maxOccurs="unbounded"/>  
1967   </xs:sequence>  
1968 </xs:complexType>
```

1969 The <Resources> element is of **ResourcesType** complex type.

1970 The <Resources> element contains the following elements:

1971 <Resource> [One to Many, Required]

1972 See Section 5.10.

1973 **5.10. Element <Resource>**

1974 The <Resource> element SHALL contain a **conjunctive sequence** of <ResourceMatch>
1975 elements.

```
1976 <xs:element name="Resource" type="xacml:ResourceType" />  
1977 <xs:complexType name="ResourceType">  
1978 <xs:sequence>  
1979 <xs:element ref="xacml:ResourceMatch" maxOccurs="unbounded" />  
1980 </xs:sequence>  
1981 </xs:complexType>
```

1982 The <Resource> element is of **ResourceType** complex type.

1983 The <Resource> element contains the following elements:

1984 <ResourceMatch> [One to Many]

1985 A **conjunctive sequence** of individual matches of the **resource attributes** in the request
1986 **context** and the embedded **attribute** values. See Section 5.11.

1987 **5.11. Element <ResourceMatch>**

1988 The <ResourceMatch> element SHALL identify a set of **resource**-related entities by matching
1989 **attribute** values in the <xacml-context:Resource> element of the request **context** with the
1990 embedded **attribute** value.

```
1991 <xs:element name="ResourceMatch" type="xacml:ResourceMatchType" />  
1992 <xs:complexType name="ResourceMatchType">  
1993 <xs:sequence>  
1994 <xs:element ref="xacml:AttributeValue" />  
1995 <xs:choice>  
1996 <xs:element ref="xacml:ResourceAttributeDesignator" />  
1997 <xs:element ref="xacml:AttributeSelector" />  
1998 </xs:choice>  
1999 </xs:sequence>  
2000 <xs:attribute name="MatchId" type="xs:anyURI" use="required" />  
2001 </xs:complexType>
```

2002 The <ResourceMatch> element is of **ResourceMatchType** complex type.

2003 The <ResourceMatch> element contains the following attributes and elements:

2004 MatchId [Required]

2005 Specifies a matching function. Values of this attribute MUST be of type **xs:anyURI**, with
2006 legal values documented in Section 7.5.

2007 <xacml:AttributeValue> [Required]

2008 Embedded attribute value.

2009 <ResourceAttributeDesignator> [Required Choice]

2010 MAY be used to identify one or more **attribute** values in the <Resource> element of the
2011 request **context**.

2012 <AttributeSelector> [Required Choice]

2013 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
2014 expression SHOULD resolve to an **attribute** in the <Resource> element of the request
2015 **context**.

2016 **5.12. Element <Actions>**

2017 The <Actions> element SHALL contain a **disjunctive sequence** of <Action> elements.

```
2018 <xs:element name="Actions" type="xacml:ActionTypes" />  
2019 <xs:complexType name="ActionTypes">  
2020 <xs:sequence>  
2021 <xs:element ref="xacml:Action" maxOccurs="unbounded" />  
2022 </xs:sequence>  
2023 </xs:complexType>
```

2024 The <Actions> element is of **ActionTypes** complex type.

2025 The <Actions> element contains the following elements:

2026 <Action> [One to Many, Required]

2027 See Section 5.13.

2028 **5.13. Element <Action>**

2029 The <Action> element SHALL contain a **conjunctive sequence** of <ActionMatch> elements.

```
2030 <xs:element name="Action" type="xacml:ActionType" />  
2031 <xs:complexType name="ActionType">  
2032 <xs:sequence>  
2033 <xs:element ref="xacml:ActionMatch" maxOccurs="unbounded" />  
2034 </xs:sequence>  
2035 </xs:complexType>
```

2036 The <Action> element is of **ActionType** complex type.

2037 The <Action> element contains the following elements:

2038 <ActionMatch> [One to Many]

2039 A **conjunctive sequence** of individual matches of the **action attributes** in the request
2040 **context** and the embedded **attribute** values. See Section 5.14.

2041 **5.14. Element <ActionMatch>**

2042 The <ActionMatch> element SHALL identify a set of **action**-related entities by matching **attribute**
2043 values in the <xacml-context:Action> element of the request **context** with the embedded
2044 **attribute** value.

```
2045 <xs:element name="ActionMatch" type="xacml:ActionMatchType" />  
2046 <xs:complexType name="ActionMatchType">  
2047 <xs:sequence>  
2048 <xs:element ref="xacml:AttributeValue" />  
2049 <xs:choice>  
2050 <xs:element ref="xacml:ActionAttributeDesignator" />  
2051 <xs:element ref="xacml:AttributeSelector" />  
2052 </xs:choice>  
2053 </xs:sequence>  
2054 <xs:attribute name="MatchId" type="xs:anyURI" use="required" />  
2055 </xs:complexType>
```

2056 The <ActionMatch> element is of **ActionMatchType** complex type.

2057 The <ActionMatch> element contains the following attributes and elements:

2058 MatchId [Required]

2059 Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI**, with
2060 legal values documented in Section 7.5.

2061 <xacml:AttributeValue> [Required]

2062 Embedded attribute value.

2063 <ActionAttributeDesignator> [Required Choice]

2064 MAY be used to identify one or more **attribute** values in the <Action> element of the
2065 request **context**.

2066 <AttributeSelector> [Required Choice]

2067 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
2068 expression SHOULD resolve to an **attribute** in the <Action> element of the **context**.

2069 **5.15. Element <Environments>**

2070 The <Environments> element SHALL contain a **disjunctive sequence** of <Environment>
2071 elements.

```
2072 <xs:element name="Environments" type="xacml:EnvironmentsType" />
2073 <xs:complexType name="EnvironmentsType">
2074   <xs:sequence>
2075     <xs:element ref="xacml:Environment" maxOccurs="unbounded" />
2076   </xs:sequence>
2077 </xs:complexType>
```

2078 The <Environments> element is of **EnvironmentsType** complex type.

2079 The <Environments> element contains the following elements:

2080 <Environment> [One to Many, Required]

2081 See Section 5.16.

2082 **5.16. Element <Environment>**

2083 The <Environment> element SHALL contain a **conjunctive sequence** of
2084 <EnvironmentMatch> elements.

```
2085 <xs:element name="Environment" type="xacml:EnvironmentType" />
2086 <xs:complexType name="EnvironmentType">
2087   <xs:sequence>
2088     <xs:element ref="xacml:EnvironmentMatch" maxOccurs="unbounded" />
2089   </xs:sequence>
2090 </xs:complexType>
```

2091 The <Environment> element is of **EnvironmentType** complex type.

2092 The <Environment> element contains the following elements:

2093 <EnvironmentMatch> [One to Many]

2094 A **conjunctive sequence** of individual matches of the **environment** attributes in the
2095 request **context** and the embedded **attribute** values.

2096 **5.17. Element <EnvironmentMatch>**

2097 The <EnvironmentMatch> element SHALL identify an environment by matching **attribute** values
2098 in the <xacml-context:Environment> element of the request **context** with the embedded
2099 **attribute** value.

```
2100 <xs:element name="EnvironmentMatch" type="xacml:EnvironmentMatchType" />  
2101 <xs:complexType name="EnvironmentMatchType">  
2102   <xs:sequence>  
2103     <xs:element ref="xacml:AttributeValue" />  
2104     <xs:choice>  
2105       <xs:element ref="xacml:EnvironmentAttributeDesignator" />  
2106       <xs:element ref="xacml:AttributeSelector" />  
2107     </xs:choice>  
2108   </xs:sequence>  
2109   <xs:attribute name="MatchId" type="xs:anyURI" use="required" />  
2110 </xs:complexType>
```

2111 The <EnvironmentMatch> element is of **EnvironmentMatchType** complex type.

2112 The <EnvironmentMatch> element contains the following attributes and elements:

2113 MatchId [Required]

2114 Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI**, with
2115 legal values documented in Section A.3.

2116 <xacml:AttributeValue> [Required]

2117 Embedded attribute value.

2118 <EnvironmentAttributeDesignator> [Required Choice]

2119 MAY be used to identify one or more **attribute** values in the <Environment> element of
2120 the request **context**.

2121 <AttributeSelector> [Required Choice]

2122 MAY be used to identify one or more **attribute** values in the request **context**. The XPath
2123 expression SHOULD resolve to an **attribute** in the <Environment> element of the
2124 request **context**.

2125 **5.18. Element <PolicySetIdReference>**

2126 The <PolicySetIdReference> element SHALL be used to reference a <PolicySet> element
2127 by id. If <PolicySetIdReference> is a URL, then it MAY be resolvable to the <PolicySet>
2128 element. However, the mechanism for resolving a **policy set** reference to the corresponding
2129 **policy set** is outside the scope of this specification.

```
2130 <xs:element name="PolicySetIdReference" type="xacml:IdReferenceType" />  
2131 <xs:complexType name="IdReferenceType">  
2132   <xs:simpleContent>  
2133     <xs:extension base="xs:anyURI">  
2134       <xs:attribute name="xacml:Version" type="xacml:VersionMatchType"  
2135       use="optional" />  
2136       <xs:attribute name="xacml:EarliestVersion" type="xacml:VersionMatchType"  
2137       use="optional" />
```

```

2138     <xs:attribute name="xacml:LatestVersion" type="xacml:VersionMatchType "
2139 use="optional" />
2140   </xs:extension>
2141   </xs:simpleContent>
2142 </xs:complexType>

```

2143 Element <PolicySetIdReference> is of **xacml:IdReferenceType** complex type.

2144 **IdReferenceType** extends the **xs:anyURI** type with the following attributes:

2145 Version [Optional]

2146 Specifies a matching expression for the version of the **policy set** referenced.

2147 EarliestVersion [Optional]

2148 Specifies a matching expression for the earliest acceptable version of the **policy set** referenced.

2150 LatestVersion [Optional]

2151 Specifies a matching expression for the latest acceptable version of the **policy set** referenced.

2153 The matching operation is defined in Section 5.21. Any combination of these attributes MAY be
 2154 present in a <PolicySetIdReference>. The referenced **policy set** MUST match all
 2155 expressions. If none of these attributes is present, then any version of the **policy set** is acceptable.
 2156 In the case that more than one matching version can be obtained, then the most recent one
 2157 SHOULD be used.

2158 5.19. Element <PolicyIdReference>

2159 The <xacml:PolicyIdReference> element SHALL be used to reference a <Policy> element
 2160 by id. If <PolicyIdReference> is a URL, then it MAY be resolvable to the <Policy> element.
 2161 However, the mechanism for resolving a **policy** reference to the corresponding **policy** is outside
 2162 the scope of this specification.

```

2163 <xs:element name="PolicyIdReference" type="xacml:IdReferenceType" />

```

2164 Element <PolicyIdReference> is of **xacml:IdReferenceType** complex type (see Section 5.18) .

2165 5.20. Simple type VersionType

2166 Elements of this type SHALL contain the version number of the **policy** or **policy set**.

```

2167 <xs:simpleType name="VersionType">
2168   <xs:restriction base="xs:string">
2169     <xs:pattern value="(\d+\.)*\d+"/>
2170   </xs:restriction>
2171 </xs:simpleType>

```

2172 The version number is expressed as a sequence of decimal numbers, each separated by a period
 2173 (.). 'd+' represents a sequence of one or more decimal digits.

2174 5.21. Simple type VersionMatchType

2175 Elements of this type SHALL contain a restricted regular expression matching a version number
 2176 (see Section 5.20). The expression SHALL match versions of a referenced **policy** or **policy set**
 2177 that are acceptable for inclusion in the referencing **policy** or **policy set**.

```

2178 <xs:simpleType name="VersionMatchType">

```

```

2179 <xs:restriction base="xs:string">
2180 <xs:pattern value="((\d+|\*)\.)*(\d+|\*|\+)" />
2181 </xs:restriction>
2182 </xs:simpleType>

```

2183 A version match is '.'-separated, like a version string. A number represents a direct numeric match.
 2184 A '*' means that any single number is valid. A '+' means that any number, and any subsequent
 2185 numbers, are valid. In this manner, the following four patterns would all match the version string
 2186 '1.2.3': '1.2.3', '1.*.3', '1.2.*' and '1.+'

2187 **5.22. Element <Policy>**

2188 The <Policy> element is the smallest entity that SHALL be presented to the **PDP** for evaluation.

2189 A <Policy> element MAY be evaluated, in which case the evaluation procedure defined in
 2190 Section 7.10 SHALL be used.

2191 The main components of this element are the <Target>, <Rule>, <CombinerParameters>,
 2192 <RuleCombinerParameters> and <Obligations> elements and the RuleCombiningAlgId
 2193 attribute.

2194 The <Target> element defines the applicability of the <Policy> element to a set of **decision**
 2195 **requests**. If the <Target> element within the <Policy> element matches the **request context**,
 2196 then the <Policy> element MAY be used by the **PDP** in making its **authorization decision**. See
 2197 Section 7.10.

2198 The <Policy> element includes a sequence of choices between <VariableDefinition> and
 2199 <Rule> elements.

2200 **Rules** included in the <Policy> element MUST be combined by the algorithm specified by the
 2201 RuleCombiningAlgId attribute.

2202 The <Obligations> element contains a set of **obligations** that MUST be fulfilled by the **PEP** in
 2203 conjunction with the **authorization decision**.

```

2204 <xs:element name="Policy" type="xacml:PolicyType"/>
2205 <xs:complexType name="PolicyType">
2206 <xs:sequence>
2207 <xs:element ref="xacml:Description" minOccurs="0"/>
2208 <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>
2209 <xs:element ref="xacml:CombinerParameters" minOccurs="0"/>
2210 <xs:element ref="xacml:Target"/>
2211 <xs:choice maxOccurs="unbounded">
2212 <xs:element ref="xacml:CombinerParameters" minOccurs="0"/>
2213 <xs:element ref="xacml:RuleCombinerParameters" minOccurs="0"/>
2214 <xs:element ref="xacml:VariableDefinition"/>
2215 <xs:element ref="xacml:Rule"/>
2216 </xs:choice>
2217 <xs:element ref="xacml:Obligations" minOccurs="0"/>
2218 </xs:sequence>
2219 <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
2220 <xs:attribute name="Version" type="xacml:VersionType" default="1.0"/>
2221 <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>
2222 </xs:complexType>

```

2223 The <Policy> element is of **PolicyType** complex type.

2224 The <Policy> element contains the following attributes and elements:

2225 PolicyId [Required]

2226 **Policy** identifier. It is the responsibility of the **PAP** to ensure that no two **policies** visible to
2227 the **PDP** have the same identifier. This MAY be achieved by following a predefined URN or
2228 URI scheme. If the **policy** identifier is in the form of a URL, then it MAY be resolvable.

2229 Version [Default 1.0]

2230 The version number of the **Policy**.

2231 RuleCombiningAlgId [Required]

2232 The identifier of the **rule-combining algorithm** by which the <Policy>,
2233 <CombinerParameters> and <RuleCombinerParameters> components MUST be
2234 combined. Standard **rule-combining algorithms** are listed in Appendix C. Standard **rule-**
2235 **combining algorithm** identifiers are listed in Section B.10.

2236 <Description> [Optional]

2237 A free-form description of the **policy**. See Section 5.2.

2238 <PolicyDefaults> [Optional]

2239 Defines a set of default values applicable to the **policy**. The scope of the
2240 <PolicyDefaults> element SHALL be the enclosing **policy**.

2241 <CombinerParameters> [Optional]

2242 A sequence of parameters to be used by the **rule-combining algorithm**.

2243 <RuleCombinerParameters> [Optional]

2244 A sequence of parameters to be used by the **rule-combining algorithm**.

2245 <Target> [Required]

2246 The <Target> element defines the applicability of a <Policy> to a set of **decision requests**.

2247 The <Target> element MAY be declared by the creator of the <Policy> element, or it
2248 MAY be computed from the <Target> elements of the referenced <Rule> elements either
2249 as an intersection or as a union.

2250 <VariableDefinition> [Any Number]

2251 Common function definitions that can be referenced from anywhere in a **rule** where an
2252 expression can be found.

2253 <Rule> [Any Number]

2254 A sequence of **rules** that MUST be combined according to the RuleCombiningAlgId
2255 attribute. **Rules** whose <Target> elements match the **decision request** MUST be
2256 considered. **Rules** whose <Target> elements do not match the **decision request** SHALL
2257 be ignored.

2258 <Obligations> [Optional]

2259 A **conjunctive sequence** of **obligations** that MUST be fulfilled by the **PEP** in conjunction
2260 with the **authorization decision**. See Section 7.14 for a description of how the set of
2261 **obligations** to be returned by the **PDP** SHALL be determined.

2262 **5.23. Element <PolicyDefaults>**

2263 The <PolicyDefaults> element SHALL specify default values that apply to the <Policy>
2264 element.

```
2265 <xs:element name="PolicyDefaults" type="xacml:DefaultsType"/>
2266 <xs:complexType name="DefaultsType">
2267   <xs:sequence>
2268     <xs:choice>
2269       <xs:element ref="xacml:XPathVersion" minOccurs="0"/>
2270     </xs:choice>
2271   </xs:sequence>
2272 </xs:complexType>
```

2273 <PolicyDefaults> element is of **DefaultsType** complex type.

2274 The <PolicyDefaults> element contains the following elements:

2275 <XPathVersion> [Optional]

2276 Default XPath version.

2277 **5.24. Element <CombinerParameters>**

2278 The <CombinerParameters> element conveys parameters for a *policy-* or *rule-combining*
2279 *algorithm*.

2280 If multiple <CombinerParameters> elements occur within the same *policy* or *policy set*, they
2281 SHALL be considered equal to one <CombinerParameters> element containing the
2282 concatenation of all the sequences of <CombinerParameters> contained in all the aforementioned
2283 <CombinerParameters> elements, such that the order of occurrence of the
2284 <CominberParameters> elements is preserved in the concatenation of the
2285 <CombinerParameter> elements.

2286 Note that none of the *combining algorithms* specified in XACML 2.0 is parameterized.

```
2287 <xs:element name="CombinerParameters" type="xacml:CombinerParametersType"/>
2288 <xs:complexType name="CombinerParametersType">
2289   <xs:sequence>
2290     <xs:element ref="xacml:CombinerParameter" minOccurs="0"
2291     maxOccurs="unbounded"/>
2292   </xs:sequence>
2293 </xs:complexType>
```

2294 The <CombinerParameters> element is of **CombinerParametersType** complex type.

2295 The <CombinerParameters> element contains the following elements:

2296 <CombinerParameter> [Any Number]

2297 A single parameter. See Section 5.25.

2298 Support for the <CombinerParameters> element is optional.

2299 **5.25. Element <CombinerParameter>**

2300 The <CombinerParameter> element conveys a single parameter for a *policy-* or *rule-*
2301 *combining algorithm*.

```
2302 <xs:element name="CombinerParameter" type="xacml:CombinerParameterType"/>
2303 <xs:complexType name="CombinerParameterType">
```



```

2304 <xs:sequence>
2305   <xs:element ref="xacml:AttributeValue" />
2306 </xs:sequence>
2307 <xs:attribute name="ParameterName" type="xs:string" use="required" />
2308 </xs:complexType>

```

2309 The <CombinerParameter> element is of **CombinerParameterType** complex type.

2310 The <CombinerParameter> element contains the following attribute:

2311 ParameterName [Required]

2312 The identifier of the parameter.

2313 AttributeValue [Required]

2314 The value of the parameter.

2315 Support for the <CombinerParameter> element is optional.

2316 **5.26. Element <RuleCombinerParameters>**

2317 The <RuleCombinerParameters> element conveys *parameters* associated with a particular
2318 *rule* within a *policy* for a *rule-combining algorithm*.

2319 Each <RuleCombinerParameters> element MUST be associated with a *rule* contained within
2320 the same *policy*. If multiple <RuleCombinerParameters> elements reference the same *rule*,
2321 they SHALL be considered equal to one <RuleCombinerParameters> element containing the
2322 concatenation of all the sequences of <CombinerParameters> contained in all the aforementioned
2323 <RuleCombinerParameters> elements, such that the order of occurrence of the
2324 <RuleCominberParameters> elements is preserved in the concatenation of the
2325 <CombinerParameter> elements.

2326 Note that none of the *rule-combining algorithms* specified in XACML 2.0 is parameterized.

```

2327 <xs:element name="RuleCombinerParameters"
2328   type="xacml:RuleCombinerParametersType" />
2329 <xs:complexType name="RuleCombinerParametersType">
2330   <xs:complexContent>
2331     <xs:extension base="xacml:CombinerParametersType">
2332       <xs:attribute name="RuleIdRef" type="xs:string" use="required" />
2333     </xs:extension>
2334   </xs:complexContent>
2335 </xs:complexType>

```

2336 The <RuleCombinerParameters> element contains the following elements:

2337 RuleIdRef [Required]

2338 The identifier of the <Rule> contained in the *policy*.

2339 Support for the <RuleCombinerParameters> element is optional, only if support for *combiner*
2340 *parameters* is optional.

2341 **5.27. Element <PolicyCombinerParameters>**

2342 The <PolicyCombinerParameters> element conveys *parameters* associated with a particular
2343 *policy* within a *policy set* for a *policy-combining algorithm*.

2344 Each <PolicyCombinerParameters> element MUST be associated with a **policy** contained
2345 within the same **policy set**. If multiple <PolicyCombinerParameters> elements reference the
2346 same **policy**, they SHALL be considered equal to one <PolicyCombinerParameters> element
2347 containing the concatenation of all the sequences of <CombinerParameters> contained in all the
2348 aforementioned <PolicyCombinerParameters> elements, such that the order of occurrence of
2349 the <PolicyCominberParameters> elements is preserved in the concatenation of the
2350 <CombinerParameter> elements.

2351 Note that none of the **policy-combining algorithms** specified in XACML 2.0 is parameterized.

```
2352 <xs:element name="PolicyCombinerParameters"  
2353 type="xacml:PolicyCombinerParametersType"/>  
2354 <xs:complexType name="PolicyCombinerParametersType">  
2355   <xs:complexContent>  
2356     <xs:extension base="xacml:CombinerParametersType">  
2357       <xs:attribute name="PolicyIdRef" type="xs:anyURI" use="required"/>  
2358     </xs:extension>  
2359   </xs:complexContent>  
2360 </xs:complexType>
```

2361 The <PolicyCombinerParameters> element is of **PolicyCombinerParametersType** complex
2362 type.

2363 The <PolicyCombinerParameters> element contains the following elements:

2364 PolicyIdRef [Required]

2365 The identifier of a <Policy> or the value of a <PolicyIdReference> contained in the
2366 **policy set**.

2367 Support for the <PolicyCombinerParameters> element is optional, only if support for
2368 **combiner parameters** is optional.

2369 **5.28. Element <PolicySetCombinerParameters>**

2370 The <PolicySetCombinerParameters> element conveys **parameters** associated with a
2371 particular **policy set** within a **policy set** for a **policy-combining algorithm**.

2372 Each <PolicySetCombinerParameters> element MUST be associated with a **policy set**
2373 contained within the same **policy set**. If multiple <PolicySetCombinerParameters> elements
2374 reference the same **policy set**, they SHALL be considered equal to one
2375 <PolicySetCombinerParameters> element containing the concatenation of all the sequences
2376 of <CombinerParameters> contained in all the aforementioned
2377 <PolicySetCombinerParameters> elements, such that the order of occurrence of the
2378 <PolicySetCominberParameters> elements is preserved in the concatenation of the
2379 <CombinerParameter> elements.

2380 Note that none of the **policy-combining algorithms** specified in XACML 2.0 is parameterized.

```
2381 <xs:element name="PolicySetCombinerParameters"  
2382 type="xacml:PolicySetCombinerParametersType"/>  
2383 <xs:complexType name="PolicySetCombinerParametersType">  
2384   <xs:complexContent>  
2385     <xs:extension base="xacml:CombinerParametersType">  
2386       <xs:attribute name="PolicySetIdRef" type="xs:anyURI" use="required"/>  
2387     </xs:extension>  
2388   </xs:complexContent>  
2389 </xs:complexType>
```

2390 The <PolicySetCombinerParameters> element is of **PolicySetCombinerParametersType**
2391 complex type.

2392 The <PolicySetCombinerParameters> element contains the following elements:
2393 PolicySetIdRef [Required]
2394 The identifier of a <PolicySet> or the value of a <PolicySetIdReference> contained
2395 in the **policy set**.
2396 Support for the <PolicySetCombinerParameters> element is optional, only if support for
2397 **combiner parameters** is optional.

2398 **5.29. Element <Rule>**

2399 The <Rule> element SHALL define the individual **rules** in the **policy**. The main components of
2400 this element are the <Target> and <Condition> elements and the Effect attribute.

2401 A <Rule> element MAY be evaluated, in which case the evaluation procedure defined in Section
2402 7.9 SHALL be used.

```
2403 <xs:element name="Rule" type="xacml:RuleType" />  
2404 <xs:complexType name="RuleType">  
2405   <xs:sequence>  
2406     <xs:element ref="xacml:Description" minOccurs="0" />  
2407     <xs:element ref="xacml:Target" minOccurs="0" />  
2408     <xs:element ref="xacml:Condition" minOccurs="0" />  
2409   </xs:sequence>  
2410   <xs:attribute name="RuleId" type="xs:string" use="required" />  
2411   <xs:attribute name="Effect" type="xacml:EffectType" use="required" />  
2412 </xs:complexType>
```

2413 The <Rule> element is of **RuleType** complex type.

2414 The <Rule> element contains the following attributes and elements:

2415 RuleId [Required]

2416 A string identifying this **rule**.

2417 Effect [Required]

2418 **Rule effect.** The value of this attribute is either “Permit” or “Deny”.

2419 <Description> [Optional]

2420 A free-form description of the **rule**.

2421 <Target> [Optional]

2422 Identifies the set of **decision requests** that the <Rule> element is intended to evaluate. If
2423 this element is omitted, then the **target** for the <Rule> SHALL be defined by the
2424 <Target> element of the enclosing <Policy> element. See Section 7.6 for details.

2425 <Condition> [Optional]

2426 A **predicate** that MUST be satisfied for the **rule** to be assigned its Effect value.

2427 **5.30. Simple type EffectType**

2428 The **EffectType** simple type defines the values allowed for the Effect attribute of the <Rule>
2429 element and for the FulfillOn attribute of the <Obligation> element.

```
2430 <xs:simpleType name="EffectType">
```

```

2431 <xs:restriction base="xs:string">
2432   <xs:enumeration value="Permit" />
2433   <xs:enumeration value="Deny" />
2434 </xs:restriction>
2435 </xs:simpleType>

```

2436 **5.31. Element <VariableDefinition>**

2437 The <VariableDefinition> element SHALL be used to define a value that can be referenced
2438 by a <VariableReference> element. The name supplied for its VariableId attribute SHALL
2439 NOT occur in the VariableId attribute of any other <VariableDefinition> element within the
2440 encompassing **policy**. The <VariableDefinition> element MAY contain undefined
2441 <VariableReference> element, but if it does, a corresponding <VariableDefinition> element
2442 MUST be defined later in the encompassing **policy**. <VariableDefinition> elements MAY be
2443 grouped together or MAY be placed close to the reference in the encompassing **policy**. There
2444 MAY be zero or more references to each <VariableDefinition> element.

```

2445 <xs:element name="VariableDefinition" type="xacml:VariableDefinitionType" />
2446 <xs:complexType name="VariableDefinitionType">
2447   <xs:sequence>
2448     <xs:element ref="xacml:Expression" />
2449   </xs:sequence>
2450   <xs:attribute name="VariableId" type="xs:string" use="required" />
2451 </xs:complexType>

```

2452 The <VariableDefinition> element is of **VariableDefinitionType** complex type. The
2453 <VariableDefinition> element has the following elements and attributes:

2454 <Expression> [Required]

2455 Any element of **ExpressionType** complex type.

2456 VariableId [Required]

2457 The name of the variable definition.

2458 **5.32. Element <VariableReference>**

2459 The <VariableReference> element is used to reference a value defined within the same
2460 encompassing <Policy> element. The <VariableReference> element SHALL refer to the
2461 <VariableDefinition> element by string equality on the value of their respective VariableId
2462 attributes. There SHALL exist one and only one <VariableDefinition> within the same
2463 encompassing <Policy> element to which the <VariableReference> refers. There MAY be
2464 zero or more <VariableReference> elements that refer to the same <VariableDefinition>
2465 element.

```

2466 <xs:element name="VariableReference" type="xacml:VariableReferenceType"
2467 substitutionGroup="xacml:Expression" />
2468 <xs:complexType name="VariableReferenceType">
2469   <xs:complexContent>
2470     <xs:extension base="xacml:ExpressionType">
2471       <xs:attribute name="VariableId" type="xs:string" use="required" />
2472     </xs:extension>
2473   </xs:complexContent>
2474 </xs:complexType>

```

2475 The <VariableReference> element is of the **VariableReferenceType** complex type, which is of
2476 the **ExpressionType** complex type and is a member of the <Expression> element substitution
2477 group. The <VariableReference> element MAY appear any place where an <Expression>
2478 element occurs in the schema.

2479 The <VariableReference> element has the following attributes:

2480 VariableId [Required]

2481 The name used to refer to the value defined in a <VariableDefinition> element.

2482 **5.33. Element <Expression>**

2483 The <Expression> element is not used directly in a *policy*. The <Expression> element
2484 signifies that an element that extends the **ExpressionType** and is a member of the
2485 <Expression> element substitution group SHALL appear in its place.

```
2486 <xs:element name="Expression" type="xacml:ExpressionType" abstract="true"/>  
2487 <xs:complexType name="ExpressionType" abstract="true"/>
```

2488 The following elements are in the <Expression> element substitution group:

2489 <Apply>, <AttributeSelector>, <AttributeValue>, <Function>,
2490 <VariableReference>, <ActionAttributeDesignator>,
2491 <ResourceAttributeDesignator>, <SubjectAttributeDesignator> and
2492 <EnvironmentAttributeDesignator>.

2493 **5.34. Element <Condition>**

2494 The <Condition> element is a Boolean function over *subject*, *resource*, *action* and
2495 *environment attributes* or functions of *attributes*.

```
2496 <xs:element name="Condition" type="xacml:ConditionType"/>  
2497 <xs:complexType name="ConditionType">  
2498   <xs:sequence>  
2499     <xs:element ref="xacml:Expression"/>  
2500   </xs:sequence>  
2501 </xs:complexType>
```

2502 The <Condition> contains one <Expression> element, with the restriction that the
2503 <Expression> return data-type MUST be "http://www.w3.org/2001/XMLSchema#boolean".
2504 Evaluation of the <Condition> element is described in Section 7.8.

2505 **5.35. Element <Apply>**

2506 The <Apply> element denotes application of a function to its arguments, thus encoding a function
2507 call. The <Apply> element can be applied to any combination of the members of the
2508 <Expression> element substitution group. See Section 5.33.

```
2509 <xs:element name="Apply" type="xacml:ApplyType"  
2510 substitutionGroup="xacml:Expression"/>  
2511 <xs:complexType name="ApplyType">  
2512   <xs:complexContent>  
2513     <xs:extension base="xacml:ExpressionType">  
2514       <xs:sequence>  
2515         <xs:element ref="xacml:Expression" minOccurs="0"  
2516 maxOccurs="unbounded"/>  
2517       </xs:sequence>  
2518       <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>  
2519     </xs:extension>  
2520   </xs:complexContent>  
2521 </xs:complexType>
```

2522 The <Apply> element is of **ApplyType** complex type.

2523 The <Apply> element contains the following attributes and elements:

2524 FunctionId [Required]

2525 The identifier of the function to be applied to the arguments. XACML-defined functions are
2526 described in Appendix A.

2527 <Expression> [Optional]

2528 Arguments to the function, which may include other functions.

2529 **5.36. Element <Function>**

2530 The <Function> element SHALL be used to name a function as an argument to the function
2531 defined by the parent <Apply> element. In the case where the parent <Apply> element is a
2532 higher-order **bag** function, the named function is applied to every element of the **bag** or **bags**
2533 identified in the other arguments of the parent element. The higher-order **bag** functions are
2534 described in Section A3A.3.12.

```
2535 <xs:element name="Function" type="xacml:FunctionType"  
2536 substitutionGroup="xacml:Expression"/>  
2537 <xs:complexType name="FunctionType">  
2538   <xs:complexContent>  
2539     <xs:extension base="xacml:ExpressionType">  
2540       <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>  
2541     </xs:extension>  
2542   </xs:complexContent>  
2543 </xs:complexType>
```

2544 The Function element is of **FunctionType** complex type.

2545 The Function element contains the following attributes:

2546 FunctionId [Required]

2547 The identifier of the function.

2548 **5.37. Complex type AttributeDesignatorType**

2549 The **AttributeDesignatorType** complex type is the type for elements that identify **attributes** by
2550 name. It contains the information required to match **attributes** in the request **context**. See Section
2551 7.2.4.

2552 It also contains information to control behaviour in the event that no matching **attribute** is present in
2553 the **context**.

2554 Elements of this type SHALL NOT alter the match semantics of **named attributes**, but MAY narrow
2555 the search space.

```
2556 <xs:complexType name="AttributeDesignatorType">  
2557   <xs:complexContent>  
2558     <xs:extension base="xacml:ExpressionType">  
2559       <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2560       <xs:attribute name="DataType" type="xs:anyURI" use="required"/>  
2561       <xs:attribute name="Issuer" type="xs:string" use="optional"/>  
2562       <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"  
2563 default="false"/>  
2564     </xs:extension>  
2565   </xs:complexContent>  
2566 </xs:complexType>
```

2567 A **named attribute** SHALL match an **attribute** if the values of their respective `AttributeId`,
2568 `DataType` and `Issuer` attributes match. The **attribute** designator's `AttributeId` MUST match,
2569 by URI equality, the `AttributeId` of the **attribute**. The **attribute** designator's `DataType` MUST
2570 match, by URI equality, the `DataType` of the same **attribute**.

2571 If the `Issuer` attribute is present in the **attribute** designator, then it MUST match, using the
2572 "urn:oasis:names:tc:xacml:1.0:function:string-equal" function, the `Issuer` of the same **attribute**. If
2573 the `Issuer` is not present in the **attribute** designator, then the matching of the **attribute** to the
2574 **named attribute** SHALL be governed by `AttributeId` and `DataType` attributes alone.

2575 The `<AttributeDesignatorType>` contains the following attributes:

2576 `AttributeId` [Required]

2577 This attribute SHALL specify the `AttributeId` with which to match the **attribute**.

2578 `DataType` [Required]

2579 The bag returned by the `<AttributeDesignator>` element SHALL contain values of this
2580 data-type.

2581 `Issuer` [Optional]

2582 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **attribute**.

2583 `MustBePresent` [Optional]

2584 This attribute governs whether the element returns "Indeterminate" or an empty **bag** in the
2585 event the **named attribute** is absent from the request **context**. See Section 7.2.5. Also
2586 see Sections 7.15.2 and 7.15.3.

2587 **5.38. Element `<SubjectAttributeDesignator>`**

2588 The `<SubjectAttributeDesignator>` element retrieves a **bag** of values for a **named**
2589 categorized **subject attribute** from the request **context**. A **subject attribute** is an **attribute**
2590 contained within a `<Subject>` element of the request **context**. A categorized **subject** is a **subject**
2591 that is identified by a particular **subject-category** attribute. A **named categorized subject attribute**
2592 is a **named subject attribute** for a particular **categorized subject**.

2593 The `<SubjectAttributeDesignator>` element SHALL return a **bag** containing all the **subject**
2594 **attribute** values that are matched by the **named categorized subject attribute**. In the event that
2595 no matching attribute is present in the context, the `MustBePresent` attribute governs whether this
2596 element returns an empty **bag** or "Indeterminate". See Section 7.2.5.

2597 The `SubjectAttributeDesignatorType` extends the match semantics of the
2598 `AttributeDesignatorType` (See Section 5.37) such that it narrows the **attribute** search space to
2599 the specific **categorized subject** such that the value of this element's `SubjectCategory` attribute
2600 matches, by URI equality, the value of the request **context's** `<Subject>` element's
2601 `SubjectCategory` attribute.

2602 If the request context contains multiple **subjects** with the same `SubjectCategory` XML attribute,
2603 then they SHALL be treated as if they were one **categorized subject**.

2604 The `<SubjectAttributeDesignator>` MAY appear in the `<SubjectMatch>` element and
2605 MAY be passed to the `<Apply>` element as an argument.

```

2606 <xs:element name="SubjectAttributeDesignator"
2607 type="xacml:SubjectAttributeDesignatorType"
2608 substitutionGroup="xacml:Expression"/>
2609 <xs:complexType name="SubjectAttributeDesignatorType">
2610   <xs:complexContent>
2611     <xs:extension base="xacml:AttributeDesignatorType">
2612       <xs:attribute name="SubjectCategory" type="xs:anyURI" use="optional"
2613 default="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"/>
2614     </xs:extension>
2615   </xs:complexContent>
2616 </xs:complexType>

```

2617 The <SubjectAttributeDesignator> element is of type **SubjectAttributeDesignatorType**.
2618 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType**
2619 complex type with a SubjectCategory attribute.

2620 SubjectCategory [Optional]

2621 This attribute SHALL specify the *categorized subject* from which to match *named subject*
2622 *attributes*. If SubjectCategory is not present, then its default value of
2623 "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" SHALL be used. Standard
2624 values of the SubjectCategory are listed in Section B.2.

2625 **5.39. Element <ResourceAttributeDesignator>**

2626 The <ResourceAttributeDesignator> element retrieves a **bag** of values for a *named*
2627 **resource attribute** from the request **context**. A **resource attribute** is an **attribute** contained
2628 within the <Resource> element of the request **context**. A *named resource attribute* is a **named**
2629 **attribute** that matches a **resource attribute**. A *named resource attribute* SHALL be considered
2630 *present* if there is at least one **resource attribute** that matches the criteria set out below. A
2631 **resource attribute** value is an **attribute** value that is contained within a **resource attribute**.

2632 The <ResourceAttributeDesignator> element SHALL return a **bag** containing all the
2633 **resource attribute** values that are matched by the *named resource attribute*. In the event that no
2634 matching attribute is present in the context, the MustBePresent attribute governs whether this
2635 element returns an empty **bag** or "Indeterminate". See Section 7.2.5.

2636 A *named resource attribute* SHALL match a **resource attribute** as per the match semantics
2637 specified in the **AttributeDesignatorType** complex type. See Section 5.37.

2638 The <ResourceAttributeDesignator> MAY appear in the <ResourceMatch> element and
2639 MAY be passed to the <Apply> element as an argument.

```

2640 <xs:element name="ResourceAttributeDesignator"
2641 type="xacml:AttributeDesignatorType" substitutionGroup="xacml:Expression"/>

```

2642 The <ResourceAttributeDesignator> element is of the **AttributeDesignatorType** complex
2643 type.

2644 **5.40. Element <ActionAttributeDesignator>**

2645 The <ActionAttributeDesignator> element retrieves a **bag** of values for a *named action*
2646 **attribute** from the request **context**. An **action attribute** is an **attribute** contained within the
2647 <Action> element of the request **context**. A *named action attribute* has specific criteria
2648 (described below) with which to match an **action attribute**. A *named action attribute* SHALL be
2649 considered *present*, if there is at least one **action attribute** that matches the criteria. An **action**
2650 **attribute value** is an **attribute value** that is contained within an **action attribute**.

2651 The <ActionAttributeDesignator> element SHALL return a **bag** of all the **action attribute**
2652 values that are matched by the **named action attribute**. In the event that no matching attribute is
2653 present in the context, the `MustBePresent` attribute governs whether this element returns an
2654 empty **bag** or “Indeterminate”. See Section 7.2.5.

2655 A **named action attribute** SHALL match an **action attribute** as per the match semantics specified
2656 in the **AttributeDesignatorType** complex type. See Section 5.37.

2657 The <ActionAttributeDesignator> MAY appear in the <ActionMatch> element and MAY
2658 be passed to the <Apply> element as an argument.

```
2659 <xs:element name="ActionAttributeDesignator" type="xacml:AttributeDesignatorType"  
2660 substitutionGroup="xacml:Expression" />
```

2661 The <ActionAttributeDesignator> element is of the **AttributeDesignatorType** complex
2662 type.

2663 **5.41. Element <EnvironmentAttributeDesignator>**

2664 The <EnvironmentAttributeDesignator> element retrieves a **bag** of values for a **named**
2665 **environment attribute** from the request **context**. An **environment attribute** is an **attribute**
2666 contained within the <Environment> element of request **context**. A **named environment**
2667 **attribute** has specific criteria (described below) with which to match an **environment attribute**. A
2668 **named environment attribute** SHALL be considered *present*, if there is at least one **environment**
2669 **attribute** that matches the criteria. An **environment attribute value** is an **attribute** value that is
2670 contained within an **environment attribute**.

2671 The <EnvironmentAttributeDesignator> element SHALL evaluate to a **bag** of all the
2672 **environment attribute** values that are matched by the **named environment attribute**. In the
2673 event that no matching attribute is present in the context, the `MustBePresent` attribute governs
2674 whether this element returns an empty **bag** or “Indeterminate”. See Section 7.2.5.

2675 A **named environment attribute** SHALL match an **environment attribute** as per the match
2676 semantics specified in the **AttributeDesignatorType** complex type. See Section 5.37.

2677 The <EnvironmentAttributeDesignator> MAY be passed to the <Apply> element as an
2678 argument.

```
2679 <xs:element name="EnvironmentAttributeDesignator"  
2680 type="xacml:AttributeDesignatorType" substitutionGroup="xacml:Expression" />
```

2681 The <EnvironmentAttributeDesignator> element is of the **AttributeDesignatorType**
2682 complex type.

2683 **5.42. Element <AttributeSelector>**

2684 The <AttributeSelector> element identifies attributes by their location in the request **context**.
2685 Support for the <AttributeSelector> element is OPTIONAL.

2686 The <AttributeSelector> element's `RequestContextPath` XML attribute SHALL contain a
2687 legal XPath expression whose context node is the <xacml-context:Request> element. The
2688 **AttributeSelector** element SHALL evaluate to a **bag** of values whose data-type is specified by
2689 the element's `DataType` attribute. If the `DataType` specified in the **AttributeSelector** is a
2690 primitive data type defined in [XF] or [XS], then the value returned by the XPath expression SHALL
2691 be converted to the `DataType` specified in the <AttributeSelector> using the constructor
2692 function below [XF Section 4] that corresponds to the `DataType`. If an error results from using the
2693 constructor function, then the value of the <AttributeSelector> SHALL be "Indeterminate".
2694

2695 xs:string()
 2696 xs:boolean()
 2697 xs:integer()
 2698 xs:double()
 2699 xs:dateTime()
 2700 xs:date()
 2701 xs:time()
 2702 xs:hexBinary()
 2703 xs:base64Binary()
 2704 xs:anyURI()
 2705 xf:yearMonthDuration()
 2706 xf:dayTimeDuration()
 2707

2708 If the `Data Type` specified in the `AttributeSelector` is not one of the preceding primitive
 2709 `Data Types`, then the `AttributeSelector` SHALL return a **bag** of instances of the specified
 2710 `Data Type`. If an error occurs when converting the values returned by the XPath expression to the
 2711 specified `Data Type`, then the result of the `AttributeSelector` SHALL be "Indeterminate".
 2712

2713 Each node selected by the specified XPath expression MUST be either a text node, an attribute
 2714 node, a processing instruction node or a comment node. The string representation of the value of
 2715 each node MUST be converted to an **attribute** value of the specified data-type, and the result of
 2716 the `AttributeSelector` is the **bag** of the **attribute** values generated from all the selected
 2717 nodes.
 2718

2719 If the node selected by the specified XPath expression is not one of those listed above (i.e. a text
 2720 node, an attribute node, a processing instruction node or a comment node), then the result of the
 2721 enclosing **policy** SHALL be "Indeterminate" with a `Status Code` value of
 2722 "urn:oasis:names:tc:xacml:1.0:status:syntax-error".
 2723

```
2724 <xs:element name="AttributeSelector" type="xacml:AttributeSelectorType"
2725 substitutionGroup="xacml:Expression" />
2726 <xs:complexType name="AttributeSelectorType">
2727   <xs:complexContent>
2728     <xs:extension base="xacml:ExpressionType">
2729       <xs:attribute name="RequestContextPath" type="xs:string" use="required"/>
2730       <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
2731       <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"
2732 default="false"/>
2733     </xs:extension>
2734   </xs:complexContent>
2735 </xs:complexType>
```

2736 The `<AttributeSelector>` element is of **AttributeSelectorType** complex type.

2737 The `<AttributeSelector>` element has the following attributes:

2738 RequestContextPath [Required]

2739 An XPath expression whose context node is the `<xacml-context:Request>` element.
 2740 There SHALL be no restriction on the XPath syntax. See also Section 5.4.

2741 DataType [Required]

2742 The **bag** returned by the `<AttributeSelector>` element SHALL contain values of this
 2743 data-type.

2744 MustBePresent [Optional]

2745 This attribute governs whether the element returns "Indeterminate" or an empty **bag** in the
2746 event the XPath expression selects no node. See Section 7.2.5. Also see Sections 7.15.2
2747 and 7.15.3.

2748 **5.43. Element <AttributeValue>**

2749 The <xacml:AttributeValue> element SHALL contain a literal **attribute** value.

```
2750 <xs:element name="AttributeValue" type="xacml:AttributeValueType"  
2751 substitutionGroup="xacml:Expression" />  
2752 <xs:complexType name="AttributeValueType" mixed="true">  
2753 <xs:complexContent>  
2754 <xs:extension base="xacml:ExpressionType">  
2755 <xs:sequence>  
2756 <xs:any namespace="##any" processContents="lax" minOccurs="0"  
2757 maxOccurs="unbounded" />  
2758 </xs:sequence>  
2759 <xs:attribute name="DataType" type="xs:anyURI" use="required" />  
2760 <xs:anyAttribute namespace="##any" processContents="lax" />  
2761 </xs:extension>  
2762 </xs:complexContent>  
2763 </xs:complexType>
```

2764 The <xacml:AttributeValue> element is of **AttributeValueType** complex type.

2765 The <xacml:AttributeValue> element has the following attributes:

2766 **DataType** [Required]

2767 The data-type of the **attribute** value.

2768 **5.44. Element <Obligations>**

2769 The <Obligations> element SHALL contain a set of <Obligation> elements.

2770 Support for the <Obligations> element is OPTIONAL.

```
2771 <xs:element name="Obligations" type="xacml:ObligationsType" />  
2772 <xs:complexType name="ObligationsType">  
2773 <xs:sequence>  
2774 <xs:element ref="xacml:Obligation" maxOccurs="unbounded" />  
2775 </xs:sequence>  
2776 </xs:complexType>
```

2777 The <Obligations> element is of **ObligationsType** complexType.

2778 The <Obligations> element contains the following element:

2779 <Obligation> [One to Many]

2780 A sequence of **obligations**. See Section 5.45.

2781 **5.45. Element <Obligation>**

2782 The <Obligation> element SHALL contain an identifier for the **obligation** and a set of **attributes**
2783 that form arguments of the action defined by the **obligation**. The FulfillOn attribute SHALL
2784 indicate the **effect** for which this **obligation** must be fulfilled by the **PEP**.

```
2785 <xs:element name="Obligation" type="xacml:ObligationType" />  
2786 <xs:complexType name="ObligationType">  
2787 <xs:sequence>
```

```

2788     <xs:element ref="xacml:AttributeAssignment" minOccurs="0"
2789 maxOccurs="unbounded" />
2790   </xs:sequence>
2791   <xs:attribute name="ObligationId" type="xs:anyURI" use="required" />
2792   <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required" />
2793 </xs:complexType>

```

2794 The <Obligation> element is of **ObligationType** complexType. See Section 7.14 for a
 2795 description of how the set of **obligations** to be returned by the **PDP** is determined.

2796 The <Obligation> element contains the following elements and attributes:

2797 ObligationId [Required]

2798 **Obligation** identifier. The value of the **obligation** identifier SHALL be interpreted by the
 2799 **PEP**.

2800 FulfillOn [Required]

2801 The **effect** for which this **obligation** must be fulfilled by the **PEP**.

2802 <AttributeAssignment> [Optional]

2803 **Obligation** arguments assignment. The values of the **obligation** arguments SHALL be
 2804 interpreted by the **PEP**.

2805 **5.46. Element <AttributeAssignment>**

2806 The <AttributeAssignment> element is used for including arguments in **obligations**. It SHALL
 2807 contain an **AttributeId** and the corresponding **attribute** value, by extending the
 2808 **AttributeValueType** type definition. The <AttributeAssignment> element MAY be used in
 2809 any way that is consistent with the schema syntax, which is a sequence of <xs:any> elements.
 2810 The value specified SHALL be understood by the **PEP**, but it is not further specified by XACML.
 2811 See Section 7.14. Section 4.2.4.3 provides a number of examples of arguments included in
 2812 **obligations**.

```

2813 <xs:element name="AttributeAssignment" type="xacml:AttributeAssignmentType" />
2814 <xs:complexType name="AttributeAssignmentType" mixed="true">
2815   <xs:complexContent>
2816     <xs:extension base="xacml:AttributeValueType">
2817       <xs:attribute name="AttributeId" type="xs:anyURI" use="required" />
2818     </xs:extension>
2819   </xs:complexContent>
2820 </xs:complexType>

```

2821 The <AttributeAssignment> element is of **AttributeAssignmentType** complex type.

2822 The <AttributeAssignment> element contains the following attributes:

2823 AttributeId [Required]

2824 The **attribute** Identifier.

2825

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

2826

2827

6.1. Element <Request>

2828

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

2829

2830

2831

2832

2833

2834

2835

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

2836

2837

2838

2839

2840

```
<xs:element name="Request" type="xacml-context:RequestType" />
```

2841

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

2842

```
  <xs:sequence>
```

2843

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

2844

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

2845

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

2846

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

2847

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

2848

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

2849

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

2850

The <Request> element contains the following elements:

2851

<Subject> [One to Many]

2852

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

2853

2854

2855

2856

2857

2858

2859

2860

2861

<Resource> [One to Many]

2862

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

2863

2864

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

- 2865 <Action> [Required]
- 2866 Specifies the requested **action** to be performed on the **resource** by listing a set of
2867 <Attribute> elements associated with the **action**.
- 2868 <Environment> [Required]
- 2869 Contains a set of <Attribute> elements for the **environment**.

2870 6.2. Element <Subject>

2871 The <Subject> element specifies a **subject** by listing a sequence of <Attribute> elements
2872 associated with the **subject**.

```
2873 <xs:element name="Subject" type="xacml-context:SubjectType" />
2874 <xs:complexType name="SubjectType">
2875   <xs:sequence>
2876     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2877     maxOccurs="unbounded" />
2878   </xs:sequence>
2879   <xs:attribute name="SubjectCategory" type="xs:anyURI"
2880   default="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" />
2881 </xs:complexType>
```

2882 The <Subject> element is of **SubjectType** complex type.

2883 The <Subject> element contains the following elements and attributes:

2884 SubjectCategory [Optional]

2885 This attribute indicates the role that the parent <Subject> played in the formation of the
2886 **access** request. If this attribute is not present in a given <Subject> element, then the
2887 default value of "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" SHALL be
2888 used, indicating that the parent <Subject> element represents the entity ultimately
2889 responsible for initiating the **access** request.

2890 If more than one <Subject> element contains a "urn:oasis:names:tc:xacml:2.0:subject-
2891 category" attribute with the same value, then the PDP SHALL treat the contents of those
2892 elements as if they were contained in the same <Subject> element.

2893 <Attribute> [Any Number]

2894 A sequence of **attributes** that apply to the subject.

2895 Typically, a <Subject> element will contain an <Attribute> with an AttributeId of
2896 "urn:oasis:names:tc:xacml:1.0:subject:subject-id", containing the identity of the **subject**.

2897 A <Subject> element MAY contain additional <Attribute> elements.

2898 6.3. Element <Resource>

2899 The <Resource> element specifies information about the **resource** to which **access** is requested,
2900 by listing a sequence of <Attribute> elements associated with the **resource**. It MAY include the
2901 **resource** content.

```
2902 <xs:element name="Resource" type="xacml-context:ResourceType" />
2903 <xs:complexType name="ResourceType">
2904   <xs:sequence>
2905     <xs:element ref="xacml-context:ResourceContent" minOccurs="0" />
```

```

2906     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2907 maxOccurs="unbounded" />
2908     </xs:sequence>
2909 </xs:complexType>

```

2910 The <Resource> element is of **ResourceType** complex type.

2911 The <Resource> element contains the following elements:

2912 <ResourceContent> [Optional]

2913 The **resource** content.

2914 <Attribute> [Any Number]

2915 A sequence of **resource attributes**.

2916 The <Resource> element MAY contain one or more <Attribute> elements with an
2917 AttributeId of "urn:oasis:names:tc:xacml:2.0:resource:resource-id". Each such
2918 <Attribute> SHALL be an absolute and fully-resolved representation of the identity of
2919 the single **resource** to which access is being requested. If there is more than one such
2920 absolute and fully-resolved representation, and if any <Attribute> with this
2921 AttributeId is specified, then an <Attribute> for each such distinct representation of
2922 the **resource** identity SHALL be specified. All such <Attribute> elements SHALL refer
2923 to the same single **resource** instance. A Profile for a particular **resource** MAY specify a
2924 single normative representation for instances of the **resource**; in this case, any
2925 <Attribute> with this AttributeId SHALL use only this one representation.

2926 A <Resource> element MAY contain additional <Attribute> elements.

2927 **6.4. Element <ResourceContent>**

2928 The <ResourceContent> element is a notional placeholder for the content of the **resource**. If an
2929 XACML **policy** references the contents of the **resource** by means of an <AttributeSelector>
2930 element, then the <ResourceContent> element MUST be included in the
2931 RequestContextPath string.

```

2932 <xs:complexType name="ResourceContentType" mixed="true">
2933   <xs:sequence>
2934     <xs:any namespace="##any" processContents="lax" minOccurs="0"
2935 maxOccurs="unbounded" />
2936   </xs:sequence>
2937   <xs:anyAttribute namespace="##any" processContents="lax" />
2938 </xs:complexType>

```

2939 The <ResourceContent> element is of **ResourceContentType** complex type.

2940 The <ResourceContent> element allows arbitrary elements and attributes.

2941 **6.5. Element <Action>**

2942 The <Action> element specifies the requested **action** on the **resource**, by listing a set of
2943 <Attribute> elements associated with the **action**.

```

2944 <xs:element name="Action" type="xacml-context:ActionType" />
2945 <xs:complexType name="ActionType">
2946   <xs:sequence>
2947     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2948 maxOccurs="unbounded" />
2949   </xs:sequence>

```

2950 `</xs:complexType>`

2951 The `<Action>` element is of **ActionType** complex type.

2952 The `<Action>` element contains the following elements:

2953 `<Attribute>` [Any Number]

2954 List of **attributes** of the **action** to be performed on the **resource**.

2955 **6.6. Element `<Environment>`**

2956 The `<Environment>` element contains a set of **attributes** of the **environment**.

```
2957 <xs:element name="Environment" type="xacml-context:EnvironmentType" />
2958 <xs:complexType name="EnvironmentType">
2959   <xs:sequence>
2960     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2961     maxOccurs="unbounded" />
2962   </xs:sequence>
2963 </xs:complexType>
```

2964 The `<Environment>` element is of **EnvironmentType** complex type.

2965 The `<Environment>` element contains the following elements:

2966 `<Attribute>` [Any Number]

2967 A list of **environment attributes**. Environment **attributes** are **attributes** that are not
2968 associated with either the **resource**, the **action** or any of the **subjects** of the **access**
2969 request.

2970 **6.7. Element `<Attribute>`**

2971 The `<Attribute>` element is the central abstraction of the request **context**. It contains **attribute**
2972 meta-data and one or more **attribute** values. The **attribute** meta-data comprises the **attribute**
2973 identifier and the **attribute** issuer. `<AttributeDesignator>` and `<AttributeSelector>`
2974 elements in the **policy** MAY refer to **attributes** by means of this meta-data.

```
2975 <xs:element name="Attribute" type="xacml-context:AttributeType" />
2976 <xs:complexType name="AttributeType">
2977   <xs:sequence>
2978     <xs:element ref="xacml-context:AttributeValue" maxOccurs="unbounded" />
2979   </xs:sequence>
2980   <xs:attribute name="AttributeId" type="xs:anyURI" use="required" />
2981   <xs:attribute name="DataType" type="xs:anyURI" use="required" />
2982   <xs:attribute name="Issuer" type="xs:string" use="optional" />
2983 </xs:complexType>
```

2984 The `<Attribute>` element is of **AttributeType** complex type.

2985 The `<Attribute>` element contains the following attributes and elements:

2986 `AttributeId` [Required]

2987 The **Attribute** identifier. A number of identifiers are reserved by XACML to denote
2988 commonly used **attributes**. See Appendix B.

2989 `DataType` [Required]

2990 The data-type of the contents of the `<xacml-context:AttributeValue>` element.
2991 This SHALL be either a primitive type defined by the XACML 2.0 specification or a type

2992 (primitive or structured) defined in a namespace declared in the `<xacml-context>`
 2993 element.

2994 Issuer [Optional]

2995 The **Attribute** issuer. For example, this attribute value MAY be an x500Name that binds to
 2996 a public key, or it may be some other identifier exchanged out-of-band by issuing and
 2997 relying parties.

2998 `<xacml-context:AttributeValue>` [One to Many]

2999 One or more **attribute** values. Each **attribute** value MAY have contents that are empty,
 3000 occur once or occur multiple times.

6.8. Element `<AttributeValue>`

3001

3002 The `<xacml-context:AttributeValue>` element contains the value of an **attribute**.

```
3003 <xs:element name="AttributeValue" type="xacml-context:AttributeValueType" />
3004 <xs:complexType name="AttributeValueType" mixed="true">
3005   <xs:sequence>
3006     <xs:any namespace="##any" processContents="lax" minOccurs="0"
3007     maxOccurs="unbounded" />
3008   </xs:sequence>
3009   <xs:anyAttribute namespace="##any" processContents="lax" />
3010 </xs:complexType>
```

3011 The `<xacml-context:AttributeValue>` element is of **AttributeValueType** complex type.

3012 The data-type of the `<xacml-context:AttributeValue>` SHALL be specified by using the
 3013 `DataType` attribute of the parent `<Attribute>` element.

6.9. Element `<Response>`

3015 The `<Response>` element is a top-level element in the XACML **context** schema. The
 3016 `<Response>` element is an abstraction layer used by the **policy** language. Any proprietary
 3017 system using the XACML specification MUST transform an XACML **context** `<Response>` element
 3018 into the form of its **authorization decision**.

3019 The `<Response>` element encapsulates the **authorization decision** produced by the **PDP**. It includes
 3020 a sequence of one or more results, with one `<Result>` element per requested **resource**. Multiple
 3021 results MAY be returned by some implementations, in particular those that support the XACML
 3022 Profile for Requests for Multiple Resources [MULT]. Support for multiple results is OPTIONAL.

```
3023 <xs:element name="Response" type="xacml-context:ResponseType" />
3024 <xs:complexType name="ResponseType">
3025   <xs:sequence>
3026     <xs:element ref="xacml-context:Result" maxOccurs="unbounded" />
3027   </xs:sequence>
3028 </xs:complexType>
```

3029 The `<Response>` element is of **ResponseType** complex type.

3030 The `<Response>` element contains the following elements:

3031 `<Result>` [One to Many]

3032 An authorization decision result. See Section 6.10.

3033 6.10. Element <Result>

3034 The <Result> element represents an **authorization decision** result for the **resource** specified by
3035 the ResourceId **attribute**. It MAY include a set of **obligations** that MUST be fulfilled by the **PEP**.
3036 If the **PEP** does not understand or cannot fulfill an **obligation**, then it MUST act as if the **PDP** had
3037 denied **access** to the requested **resource**.

3038

```
3039 <xs:complexType name="ResultType">  
3040   <xs:sequence>  
3041     <xs:element ref="xacml-context:Decision"/>  
3042     <xs:element ref="xacml-context:Status" minOccurs="0"/>  
3043     <xs:element ref="xacml:Obligations" minOccurs="0"/>  
3044   </xs:sequence>  
3045   <xs:attribute name="ResourceId" type="xs:string" use="optional"/>  
3046 </xs:complexType>
```

3047 The <Result> element is of **ResultType** complex type.

3048 The <Result> element contains the following attributes and elements:

3049 ResourceId [Optional]

3050 The identifier of the requested **resource**. If this attribute is omitted, then the **resource**
3051 identity is that specified by the "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
3052 **resource attribute** in the corresponding <Request> element.

3053 <Decision> [Required]

3054 The **authorization decision**: "Permit", "Deny", "Indeterminate" or "NotApplicable".

3055 <Status> [Optional]

3056 Indicates whether errors occurred during evaluation of the **decision request**, and
3057 optionally, information about those errors. If the <Response> element contains <Result>
3058 elements whose <Status> elements are all identical, and the <Response> element is
3059 contained in a protocol wrapper that can convey status information, then the common
3060 status information MAY be placed in the protocol wrapper and this <Status> element
3061 MAY be omitted from all <Result> elements.

3062 <Obligations> [Optional]

3063 A list of **obligations** that MUST be fulfilled by the **PEP**. If the **PEP** does not understand or
3064 cannot fulfill an **obligation**, then it MUST act as if the **PDP** had denied **access** to the
3065 requested **resource**. See Section 7.14 for a description of how the set of **obligations** to
3066 be returned by the PDP is determined.

3067 6.11. Element <Decision>

3068 The <Decision> element contains the result of **policy** evaluation.

```
3069 <xs:element name="Decision" type="xacml-context:DecisionType"/>  
3070 <xs:simpleType name="DecisionType">  
3071   <xs:restriction base="xs:string">  
3072     <xs:enumeration value="Permit"/>  
3073     <xs:enumeration value="Deny"/>  
3074     <xs:enumeration value="Indeterminate"/>  
3075     <xs:enumeration value="NotApplicable"/>  
3076   </xs:restriction>  
3077 </xs:simpleType>
```

- 3078 The <Decision> element is of **DecisionType** simple type.
- 3079 The values of the <Decision> element have the following meanings:
- 3080 "Permit": the requested **access** is permitted.
- 3081 "Deny": the requested **access** is denied.
- 3082 "Indeterminate": the PDP is unable to evaluate the requested **access**. Reasons for such
 3083 inability include: missing **attributes**, network errors while retrieving **policies**, division by
 3084 zero during **policy** evaluation, syntax errors in the **decision request** or in the **policy**, etc..
- 3085 "NotApplicable": the **PDP** does not have any **policy** that applies to this **decision request**.

3086 **6.12. Element <Status>**

3087 The <Status> element represents the status of the **authorization decision** result.

```
3088 <xs:element name="Status" type="xacml-context:StatusType" />
3089 <xs:complexType name="StatusType">
3090   <xs:sequence>
3091     <xs:element ref="xacml-context:StatusCode" />
3092     <xs:element ref="xacml-context:StatusMessage" minOccurs="0" />
3093     <xs:element ref="xacml-context:StatusDetail" minOccurs="0" />
3094   </xs:sequence>
3095 </xs:complexType>
```

3096 The <Status> element is of **StatusType** complex type.

3097 The <Status> element contains the following elements:

3098 <StatusCode> [Required]

3099 Status code.

3100 <StatusMessage> [Optional]

3101 A status message describing the status code.

3102 <StatusDetail> [Optional]

3103 Additional status information.

3104 **6.13. Element <StatusCode>**

3105 The <StatusCode> element contains a major status code value and an optional sequence of
 3106 minor status codes.

```
3107 <xs:element name="StatusCode" type="xacml-context:StatusCodeType" />
3108 <xs:complexType name="StatusCodeType">
3109   <xs:sequence>
3110     <xs:element ref="xacml-context:StatusCode" minOccurs="0" />
3111   </xs:sequence>
3112   <xs:attribute name="Value" type="xs:anyURI" use="required" />
3113 </xs:complexType>
```

3114 The <StatusCode> element is of **StatusCodeType** complex type.

3115 The <StatusCode> element contains the following attributes and elements:

3116 Value [Required]

3117 See Section B.9 for a list of values.

3118 <StatusCode> [Any Number]

3119 Minor status code. This status code qualifies its parent status code.

3120 **6.14. Element <StatusMessage>**

3121 The <StatusMessage> element is a free-form description of the status code.

```
3122 <xs:element name="StatusMessage" type="xs:string" />
```

3123 The <StatusMessage> element is of **xs:string** type.

3124 **6.15. Element <StatusDetail>**

3125 The <StatusDetail> element qualifies the <Status> element with additional information.

```
3126 <xs:element name="StatusDetail" type="xacml-context:StatusDetailType" />
```

```
3127 <xs:complexType name="StatusDetailType">
```

```
3128 <xs:sequence>
```

```
3129 <xs:any namespace="##any" processContents="lax" minOccurs="0"
```

```
3130 maxOccurs="unbounded" />
```

```
3131 </xs:sequence>
```

```
3132 </xs:complexType>
```

3133 The <StatusDetail> element is of **StatusDetailType** complex type.

3134 The <StatusDetail> element allows arbitrary XML content.

3135 Inclusion of a <StatusDetail> element is optional. However, if a **PDP** returns one of the
3136 following XACML-defined <StatusCode> values and includes a <StatusDetail> element, then
3137 the following rules apply.

3138 urn:oasis:names:tc:xacml:1.0:status:ok

3139 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “ok” status value.

3140 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

3141 A **PDP** MAY choose not to return any <StatusDetail> information or MAY choose to return a
3142 <StatusDetail> element containing one or more <xacml-context:
3143 MissingAttributeDetail> elements.

3144 urn:oasis:names:tc:xacml:1.0:status:syntax-error

3145 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “syntax-error” status
3146 value. A syntax error may represent either a problem with the **policy** being used or with the
3147 request **context**. The **PDP** MAY return a <StatusMessage> describing the problem.

3148 urn:oasis:names:tc:xacml:1.0:status:processing-error

3149 A **PDP** MUST NOT return <StatusDetail> element in conjunction with the “processing-error”
3150 status value. This status code indicates an internal problem in the **PDP**. For security reasons, the
3151 **PDP** MAY choose to return no further information to the **PEP**. In the case of a divide-by-zero error
3152 or other computational error, the **PDP** MAY return a <StatusMessage> describing the nature of
3153 the error.

3154

6.16. Element <MissingAttributeDetail>

3155 The <MissingAttributeDetail> element conveys information about **attributes** required for
3156 **policy** evaluation that were missing from the request **context**.

```
3157 <xs:element name="MissingAttributeDetail" type="xacml -  
3158 context:MissingAttributeDetailType" />  
3159 <xs:complexType name="MissingAttributeDetailType">  
3160   <xs:sequence>  
3161     <xs:element ref="xacml-context:AttributeValue" minOccurs="0"  
3162     maxOccurs="unbounded" />  
3163   </xs:sequence>  
3164   <xs:attribute name="AttributeId" type="xs:anyURI" use="required" />  
3165   <xs:attribute name="DataType" type="xs:anyURI" use="required" />  
3166   <xs:attribute name="Issuer" type="xs:string" use="optional" />  
3167 </xs:complexType>
```

3168 The <MissingAttributeDetail> element is of **MissingAttributeDetailType** complex type.

3169 The <MissingAttributeDetail> element contains the following attributes and elements:

3170 AttributeValue [Optional]

3171 The required value of the missing **attribute**.

3172 <AttributeId> [Required]

3173 The identifier of the missing **attribute**.

3174 <DataType> [Required]

3175 The data-type of the missing **attribute**.

3176 Issuer [Optional]

3177 This attribute, if supplied, SHALL specify the required **Issuer** of the missing **attribute**.

3178 If the PDP includes <xacml-context:AttributeValue> elements in the <MissingAttributeDetail>
3179 element, then this indicates the acceptable values for that attribute. If no <xacml-
3180 context:AttributeValue> elements are included, then this indicates the names of attributes that the
3181 PDP failed to resolve during its evaluation. The list of attributes may be partial or complete. There
3182 is no guarantee by the PDP that supplying the missing values or attributes will be sufficient to
3183 satisfy the policy.

3184 7. Functional requirements (normative)

3185 This section specifies certain functional requirements that are not directly associated with the
3186 production or consumption of a particular XACML element.

3187 7.1. Policy enforcement point

3188 This section describes the requirements for the **PEP**.

3189 An application functions in the role of the **PEP** if it guards access to a set of **resources** and asks
3190 the **PDP** for an **authorization decision**. The **PEP** MUST abide by the **authorization decision** as
3191 described in one of the following sub-sections

3192

7.1.1. Base PEP

3193 If the **decision** is "Permit", then the **PEP** SHALL permit **access**. If **obligations** accompany the
3194 **decision**, then the **PEP** SHALL permit **access** only if it understands and it can and will discharge
3195 those **obligations**.

3196 If the **decision** is "Deny", then the **PEP** SHALL deny **access**. If **obligations** accompany the
3197 **decision**, then the **PEP** shall deny **access** only if it understands, and it can and will discharge
3198 those **obligations**.

3199 If the **decision** is "Not Applicable", then the **PEP's** behavior is undefined.

3200 If the **decision** is "Indeterminate", then the **PEP's** behavior is undefined.

3201

7.1.2. Deny-biased PEP

3202 If the **decision** is "Permit", then the **PEP** SHALL permit **access**. If **obligations** accompany the
3203 **decision**, then the **PEP** SHALL permit **access** only if it understands and it can and will discharge
3204 those **obligations**.

3205 All other **decisions** SHALL result in the denial of **access**.

3206 Note: other actions, e.g. consultation of additional **PDPs**, reformulation/resubmission of the
3207 **decision request**, etc., are not prohibited.

3208

7.1.3. Permit-biased PEP

3209 If the **decision** is "Deny", then the **PEP** SHALL deny **access**. If **obligations** accompany the
3210 **decision**, then the **PEP** shall deny **access** only if it understands, and it can and will discharge
3211 those **obligations**.

3212 All other **decisions** SHALL result in the permission of **access**.

3213 Note: other actions, e.g. consultation of additional **PDPs**, reformulation/resubmission of the
3214 **decision request**, etc., are not prohibited.

3215

7.2. Attribute evaluation

3216 **Attributes** are represented in the request **context** by the **context handler**, regardless of whether
3217 or not they appeared in the original **decision request**, and are referred to in the **policy** by **subject**,
3218 **resource**, **action** and **environment attribute** designators and **attribute** selectors. A **named**
3219 **attribute** is the term used for the criteria that the specific **subject**, **resource**, **action** and
3220 **environment attribute** designators and selectors use to refer to particular **attributes** in the
3221 **subject**, **resource**, **action** and **environment** elements of the request **context**, respectively.

3222

7.2.1. Structured attributes

3223 <xacml:AttributeValue> and <xacml-context:AttributeValue> elements MAY contain
3224 an instance of a structured XML data-type, for example <ds:KeyInfo>. XACML 2.0 supports
3225 several ways for comparing the contents of such elements.

3226 1. In some cases, such elements MAY be compared using one of the XACML string functions,
3227 such as "regex-string-match", described below. This requires that the element be given
3228 the data-type "<http://www.w3.org/2001/XMLSchema#string>". For example, a structured
3229 data-type that is actually a ds:KeyInfo/KeyName would appear in the Context as:

```
3230 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
3231   &lt;ds:KeyName&gt;jhibbert-key&lt;/ds:KeyName&gt;
3232 </AttributeValue>
```

3233 In general, this method will not be adequate unless the structured data-type is quite simple.

3234 2. An `<AttributeSelector>` element MAY be used to select the contents of a leaf sub-
3235 element of the structured data-type by means of an XPath expression. That value MAY
3236 then be compared using one of the supported XACML functions appropriate for its primitive
3237 data-type. This method requires support by the **PDP** for the optional XPath expressions
3238 feature.

3239 3. An `<AttributeSelector>` element MAY be used to select any node in the structured
3240 data-type by means of an XPath expression. This node MAY then be compared using one
3241 of the XPath-based functions described in Section A.3. This method requires support by
3242 the **PDP** for the optional XPath expressions and XPath functions features.

3243 See also Section 8.2.

3244 7.2.2. Attribute bags

3245 XACML defines implicit collections of its data-types. XACML refers to a collection of values that are
3246 of a single data-type as a **bag**. **Bags** of data-types are needed because selections of nodes from
3247 an XML **resource** or XACML request **context** may return more than one value.

3248 The `<AttributeSelector>` element uses an XPath expression to specify the selection of data
3249 from an XML **resource**. The result of an XPath expression is termed a *node-set*, which contains all
3250 the leaf nodes from the XML **resource** that match the predicate in the XPath expression. Based on
3251 the various indexing functions provided in the XPath specification, it SHALL be implied that a
3252 resultant node-set is the collection of the matching nodes. XACML also defines the
3253 `<AttributeDesignator>` element to have the same matching methodology for **attributes** in the
3254 XACML request **context**.

3255 The values in a **bag** are not ordered, and some of the values may be duplicates. There SHALL be
3256 no notion of a **bag** containing **bags**, or a **bag** containing values of differing types. I.e. a **bag** in
3257 XACML SHALL contain only values that are of the same data-type.

3258 7.2.3. Multivalued attributes

3259 If a single `<Attribute>` element in a request **context** contains multiple `<xacml-
3260 context:AttributeValue>` child elements, then the **bag** of values resulting from evaluation of
3261 the `<Attribute>` element MUST be identical to the **bag** of values that results from evaluating a
3262 **context** in which each `<xacml-context:AttributeValue>` element appears in a separate
3263 `<Attribute>` element, each carrying identical meta-data.

3264 7.2.4. Attribute Matching

3265 A **named attribute** includes specific criteria with which to match **attributes** in the **context**. An
3266 **attribute** specifies an `AttributeId` and `DataType`, and a **named attribute** also specifies the
3267 Issuer. A **named attribute** SHALL match an **attribute** if the values of their respective
3268 `AttributeId`, `DataType` and optional `Issuer` attributes match within their particular element -
3269 **subject**, **resource**, **action** or **environment** - of the **context**. The `AttributeId` of the **named**
3270 **attribute** MUST match, by URI equality, the `AttributeId` of the corresponding **context attribute**.
3271 The `DataType` of the **named attribute** MUST match, by URI equality, the `DataType` of the
3272 corresponding **context attribute**. If `Issuer` is supplied in the **named attribute**, then it MUST

3273 match, using the `urn:oasis:names:tc:xacml:1.0:function:string-equal` function, the
3274 Issuer of the corresponding **context attribute**. If Issuer is not supplied in the **named attribute**,
3275 then the matching of the **context attribute** to the **named attribute** SHALL be governed by
3276 AttributeId and DataType alone, regardless of the presence, absence, or actual value of
3277 Issuer in the corresponding **context attribute**. In the case of an **attribute** selector, the matching
3278 of the **attribute** to the **named attribute** SHALL be governed by the XPath expression and
3279 DataType.

3280 7.2.5. Attribute Retrieval

3281 The **PDP** SHALL request the values of **attributes** in the request **context** from the **context handler**.
3282 The **PDP** SHALL reference the **attributes** as if they were in a physical request **context** document,
3283 but the **context handler** is responsible for obtaining and supplying the requested values by
3284 whatever means it deems appropriate. The **context handler** SHALL return the values of
3285 **attributes** that match the **attribute** designator or **attribute** selector and form them into a **bag** of
3286 values with the specified data-type. If no **attributes** from the request **context** match, then the
3287 **attribute** SHALL be considered missing. If the **attribute** is missing, then `MustBePresent`
3288 governs whether the **attribute** designator or **attribute** selector returns an empty **bag** or an
3289 "Indeterminate" result. If `MustBePresent` is "False" (default value), then a missing **attribute**
3290 SHALL result in an empty **bag**. If `MustBePresent` is "True", then a missing **attribute** SHALL
3291 result in "Indeterminate". This "Indeterminate" result SHALL be handled in accordance with the
3292 specification of the encompassing expressions, **rules**, **policies** and **policy sets**. If the result is
3293 "Indeterminate", then the `AttributeId`, `DataType` and `Issuer` of the **attribute** MAY be listed in
3294 the **authorization decision** as described in Section 7.13. However, a **PDP** MAY choose not to
3295 return such information for security reasons.

3296 7.2.6. Environment Attributes

3297 Standard **environment attributes** are listed in Section B.8. If a value for one of these **attributes** is
3298 supplied in the **decision request**, then the **context handler** SHALL use that value. Otherwise, the
3299 **context handler** SHALL supply a value. In the case of date and time **attributes**, the supplied
3300 value SHALL have the semantics of the "date and time that apply to the **decision request**".

3301 7.3. Expression evaluation

3302 XACML specifies expressions in terms of the elements listed below, of which the `<Apply>` and
3303 `<Condition>` elements recursively compose greater expressions. Valid expressions SHALL be
3304 type correct, which means that the types of each of the elements contained within `<Apply>` and
3305 `<Condition>` elements SHALL agree with the respective argument types of the function that is
3306 named by the `FunctionId` attribute. The resultant type of the `<Apply>` or `<Condition>`
3307 element SHALL be the resultant type of the function, which MAY be narrowed to a primitive data-
3308 type, or a **bag** of a primitive data-type, by type-unification. XACML defines an evaluation result of
3309 "Indeterminate", which is said to be the result of an invalid expression, or an operational error
3310 occurring during the evaluation of the expression.

3311 XACML defines these elements to be in the substitution group of the `<Expression>` element:

- 3312 • `<xacml:AttributeValue>`
- 3313 • `<xacml:SubjectAttributeDesignator>`
- 3314 • `<xacml:ResourceAttributeDesignator>`
- 3315 • `<xacml:ActionAttributeDesignator>`

- 3316 • <xacml:EnvironmentAttributeDesignator>
- 3317 • <xacml:AttributeSelector>
- 3318 • <xacml:Apply>
- 3319 • <xacml:Condition>
- 3320 • <xacml:Function>
- 3321 • <xacml:VariableReference>

3322 7.4. Arithmetic evaluation

3323 IEEE 754 [IEEE 754] specifies how to evaluate arithmetic functions in a context, which specifies
 3324 defaults for precision, rounding, etc. XACML SHALL use this specification for the evaluation of all
 3325 integer and double functions relying on the *Extended Default Context*, enhanced with double
 3326 precision:

- 3327 flags - all set to 0
- 3328 trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the “division-by-zero” trap
 3329 enabler, which SHALL be set to 1
- 3330 precision - is set to the designated double precision
- 3331 rounding - is set to round-half-even (IEEE 854 §4.1)

3332 7.5. Match evaluation

3333 **Attribute** matching elements appear in the <Target> element of **rules**, **policies** and **policy sets**.
 3334 They are the following:

- 3335 <SubjectMatch>
- 3336 <ResourceMatch>
- 3337 <ActionMatch>
- 3338 <EnvironmentMatch>

3339 These elements represent Boolean expressions over **attributes** of the **subject**, **resource**, **action**
 3340 and **environment**, respectively. A matching element contains a MatchId attribute that specifies
 3341 the function to be used in performing the match evaluation, an <xacml:AttributeValue> and an
 3342 <AttributeDesignator> or <AttributeSelector> element that specifies the **attribute** in the
 3343 **context** that is to be matched against the specified value.

3344 The MatchId attribute SHALL specify a function that compares two arguments, returning a result
 3345 type of "http://www.w3.org/2001/XMLSchema#boolean". The **attribute** value specified in the
 3346 matching element SHALL be supplied to the MatchId function as its first argument. An element of
 3347 the **bag** returned by the <AttributeDesignator> or <AttributeSelector> element SHALL
 3348 be supplied to the MatchId function as its second argument, as explained below. The DataType
 3349 of the <xacml:AttributeValue> SHALL match the data-type of the first argument expected by
 3350 the MatchId function. The DataType of the <AttributeDesignator> or
 3351 <AttributeSelector> element SHALL match the data-type of the second argument expected
 3352 by the MatchId function.

3353 The XACML standard functions that meet the requirements for use as a `MatchId` attribute value
3354 are:

3355 `urn:oasis:names:tc:xacml:2.0:function:-type-equal`

3356 `urn:oasis:names:tc:xacml:2.0:function:-type-greater-than`

3357 `urn:oasis:names:tc:xacml:2.0:function:-type-greater-than-or-equal`

3358 `urn:oasis:names:tc:xacml:2.0:function:-type-less-than`

3359 `urn:oasis:names:tc:xacml:2.0:function:-type-less-than-or-equal`

3360 `urn:oasis:names:tc:xacml:2.0:function:-type-match`

3361 In addition, functions that are strictly within an extension to XACML MAY appear as a value for the
3362 `MatchId` attribute, and those functions MAY use data-types that are also extensions, so long as
3363 the extension function returns a Boolean result and takes two single base types as its inputs. The
3364 function used as the value for the `MatchId` attribute SHOULD be easily indexable. Use of non-
3365 indexable or complex functions may prevent efficient evaluation of **decision requests**.

3366 The evaluation semantics for a matching element is as follows. If an operational error were to
3367 occur while evaluating the `<AttributeDesignator>` or `<AttributeSelector>` element, then
3368 the result of the entire expression SHALL be "Indeterminate". If the `<AttributeDesignator>` or
3369 `<AttributeSelector>` element were to evaluate to an empty **bag**, then the result of the
3370 expression SHALL be "False". Otherwise, the `MatchId` function SHALL be applied between the
3371 `<xacml:AttributeValue>` and each element of the **bag** returned from the
3372 `<AttributeDesignator>` or `<AttributeSelector>` element. If at least one of those function
3373 applications were to evaluate to "True", then the result of the entire expression SHALL be "True".
3374 Otherwise, if at least one of the function applications results in "Indeterminate", then the result
3375 SHALL be "Indeterminate". Finally, if all function applications evaluate to "False", then the result of
3376 the entire expression SHALL be "False".

3377 It is also possible to express the semantics of a **target** matching element in a **condition**. For
3378 instance, the **target** match expression that compares a "subject-name" starting with the name
3379 "John" can be expressed as follows:

```
3380 <SubjectMatch  
3381 MatchId="urn:oasis:names:tc:xacml:1.0:function:regexp-string-match">  
3382   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3383     John.*  
3384   </AttributeValue>  
3385   <SubjectAttributeDesignator  
3386     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3387     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3388 </SubjectMatch>
```

3389 Alternatively, the same match semantics can be expressed as an `<Apply>` element in a **condition**
3390 by using the "urn:oasis:names:tc:xacml:1.0:function:any-of" function, as follows:

```
3391 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">  
3392   <Function  
3393     FunctionId="urn:oasis:names:tc:xacml:1.0:function:regexp-string-match"/>  
3394   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3395     John.*  
3396   </AttributeValue>  
3397   <SubjectAttributeDesignator  
3398     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3399     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3400 </Apply>
```

3401

7.6. Target evaluation

3402
3403
3404
3405
3406
3407

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

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

3408

Table 1 - Target match table

3409
3410
3411
3412

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

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

3413
3414
3415
3416
3417
3418
3419
3420

Table 2 - Subjects match table

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

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

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

3421

Table 3 - Subject match table

3422

7.7. VariableReference Evaluation

3423 The <VariableReference> element references a single <VariableDefinition> element
 3424 contained within the same <Policy> element. A <VariableReference> that does not
 3425 reference a particular <VariableDefinition> element within the encompassing <Policy>
 3426 element is called an undefined reference. **Policies** with undefined references are invalid.

3427 In any place where a <VariableReference> occurs, it has the effect as if the text of the
 3428 <Expression> element defined in the <VariableDefinition> element replaces the
 3429 <VariableReference> element. Any evaluation scheme that preserves this semantic is
 3430 acceptable. For instance, the expression in the <VariableDefinition> element may be
 3431 evaluated to a particular value and cached for multiple references without consequence. (I.e. the
 3432 value of an <Expression> element remains the same for the entire policy evaluation.) This
 3433 characteristic is one of the benefits of XACML being a declarative language.

3434

7.8. Condition evaluation

3435 The **condition** value SHALL be "True" if the <Condition> element is absent, or if it evaluates to
 3436 "True". Its value SHALL be "False" if the <Condition> element evaluates to "False". The
 3437 **condition** value SHALL be "Indeterminate", if the expression contained in the <Condition>
 3438 element evaluates to "Indeterminate."

3439

7.9. Rule evaluation

3440 A **rule** has a value that can be calculated by evaluating its contents. **Rule** evaluation involves
 3441 separate evaluation of the **rule's target** and **condition**. The **rule** truth table is shown in Table 4.

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

3442

Table 4 - Rule truth table

3443 If the **target** value is "No-match" or "Indeterminate" then the **rule** value SHALL be "NotApplicable"
 3444 or "Indeterminate", respectively, regardless of the value of the **condition**. For these cases,
 3445 therefore, the **condition** need not be evaluated.

3446 If the **target** value is "Match" and the **condition** value is "True", then the **effect** specified in the
 3447 enclosing <Rule> element SHALL determine the **rule's** value.

3448 **7.10. Policy evaluation**

3449 The value of a **policy** SHALL be determined only by its contents, considered in relation to the
 3450 contents of the request **context**. A **policy's** value SHALL be determined by evaluation of the
 3451 **policy's target** and **rules**.

3452 The **policy's target** SHALL be evaluated to determine the applicability of the **policy**. If the **target**
 3453 evaluates to "Match", then the value of the **policy** SHALL be determined by evaluation of the
 3454 **policy's rules**, according to the specified **rule-combining algorithm**. If the **target** evaluates to
 3455 "No-match", then the value of the **policy** SHALL be "NotApplicable". If the **target** evaluates to
 3456 "Indeterminate", then the value of the **policy** SHALL be "Indeterminate".

3457 The **policy** truth table is shown in Table 5.

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

3458 **Table 5 - Policy truth table**

3459 A **rules** value of "At least one rule value is its Effect" means either that the <Rule> element is
 3460 absent, or one or more of the **rules** contained in the **policy** is applicable to the **decision request**
 3461 (i.e., it returns the value of its "Effect"; see Section 7.9). A **rules** value of "All rule values are
 3462 'NotApplicable'" SHALL be used if no **rule** contained in the **policy** is applicable to the request and if
 3463 no **rule** contained in the **policy** returns a value of "Indeterminate". If no **rule** contained in the
 3464 **policy** is applicable to the request, but one or more **rule** returns a value of "Indeterminate", then the
 3465 **rules** SHALL evaluate to "At least one rule value is 'Indeterminate'".

3466 If the **target** value is "No-match" or "Indeterminate" then the **policy** value SHALL be
 3467 "NotApplicable" or "Indeterminate", respectively, regardless of the value of the **rules**. For these
 3468 cases, therefore, the **rules** need not be evaluated.

3469 If the **target** value is "Match" and the **rule** value is "At least one rule value is it's Effect" or "At least
 3470 one rule value is 'Indeterminate'", then the **rule-combining algorithm** specified in the **policy**
 3471 SHALL determine the **policy** value.

3472 Note that none of the **rule-combining algorithms** defined by XACML 2.0 take parameters.
 3473 However, non-standard **combining algorithms** MAY take parameters. In such a case, the values
 3474 of these parameters associated with the **rules**, MUST be taken into account when evaluating the
 3475 **policy**. The parameters and their types should be defined in the specification of the **combining**
 3476 **algorithm**. If the implementation supports combiner parameters and if combiner parameters are

3477 present in a **policy**, then the parameter values MUST be supplied to the **combining algorithm**
 3478 implementation.

3479 7.11. Policy Set evaluation

3480 The value of a **policy set** SHALL be determined by its contents, considered in relation to the
 3481 contents of the **request context**. A **policy set's** value SHALL be determined by evaluation of the
 3482 **policy set's target, policies** and **policy sets**, according to the specified **policy-combining**
 3483 **algorithm**.

3484 The **policy set's target** SHALL be evaluated to determine the applicability of the **policy set**. If the
 3485 **target** evaluates to "Match" then the value of the **policy set** SHALL be determined by evaluation of
 3486 the **policy set's policies** and **policy sets**, according to the specified **policy-combining algorithm**.
 3487 If the **target** evaluates to "No-match", then the value of the **policy set** shall be "NotApplicable". If
 3488 the **target** evaluates to "Indeterminate", then the value of the **policy set** SHALL be "Indeterminate".

3489 The **policy set** truth table is shown in Table 6.

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

3490 **Table 6 – Policy set truth table**

3491 A **policies** value of "At least one policy value is its **Decision**" SHALL be used if there are no
 3492 contained or referenced **policies** or **policy sets**, or if one or more of the **policies** or **policy sets**
 3493 contained in or referenced by the **policy set** is applicable to the **decision request** (i.e., returns a
 3494 value determined by its **combining algorithm**) A **policies** value of "All policy values are
 3495 'NotApplicable'" SHALL be used if no **policy** or **policy set** contained in or referenced by the **policy**
 3496 **set** is applicable to the request and if no **policy** or **policy set** contained in or referenced by the
 3497 **policy set** returns a value of "Indeterminate". If no **policy** or **policy set** contained in or referenced
 3498 by the **policy set** is applicable to the request but one or more **policy** or **policy set** returns a value
 3499 of "Indeterminate", then the **policies** SHALL evaluate to "At least one policy value is
 3500 'Indeterminate'".

3501 If the **target** value is "No-match" or "Indeterminate" then the **policy set** value SHALL be
 3502 "NotApplicable" or "Indeterminate", respectively, regardless of the value of the **policies**. For these
 3503 cases, therefore, the **policies** need not be evaluated.

3504 If the **target** value is "Match" and the **policies** value is "At least one policy value is its **Decision**" or
 3505 "At least one policy value is 'Indeterminate'", then the **policy-combining algorithm** specified in the
 3506 **policy set** SHALL determine the **policy set** value.

3507 Note that none of the **policy-combining algorithms** defined by XACML 2.0 take parameters.
 3508 However, non-standard **combining algorithms** MAY take parameters. In such a case, the values

3509 of these parameters associated with the **policies**, MUST be taken into account when evaluating the
3510 **policy set**. The parameters and their types should be defined in the specification of the
3511 **combining algorithm**. If the implementation supports combiner parameters and if combiner
3512 parameters are present in a **policy**, then the parameter values MUST be supplied to the
3513 **combining algorithm** implementation.

3514 7.12. Hierarchical resources

3515 It is often the case that a **resource** is organized as a hierarchy (e.g. file system, XML document).
3516 XACML provides several optional mechanisms for supporting hierarchical resources. These are
3517 described in the XACML Profile for Hierarchical Resources [HIER] and in the XACML Profile for
3518 Requests for Multiple Resources [MULT].

3519 7.13. Authorization decision

3520 In relation to a particular **decision request**, the **PDP** is defined by a **policy-combining algorithm**
3521 and a set of **policies** and/or **policy sets**. The **PDP** SHALL return a response **context** as if it had
3522 evaluated a single **policy set** consisting of this **policy-combining algorithm** and the set of
3523 **policies** and/or **policy sets**.

3524 The **PDP** MUST evaluate the **policy set** as specified in Sections 5 and 7. The **PDP** MUST return a
3525 response **context**, with one <Decision> element of value "Permit", "Deny", "Indeterminate" or
3526 "NotApplicable".

3527 If the **PDP** cannot make a decision, then an "Indeterminate" <Decision> element SHALL be
3528 returned.

3529 7.14. Obligations

3530 A **policy** or **policy set** may contain one or more **obligations**. When such a **policy** or **policy set** is
3531 evaluated, an **obligation** SHALL be passed up to the next level of evaluation (the enclosing or
3532 referencing **policy**, **policy set** or **authorization decision**) only if the **effect** of the **policy** or **policy**
3533 **set** being evaluated matches the value of the FulfillOn attribute of the **obligation**.

3534 As a consequence of this procedure, no **obligations** SHALL be returned to the **PEP** if the **policies**
3535 or **policy sets** from which they are drawn are not evaluated, or if their evaluated result is
3536 "Indeterminate" or "NotApplicable", or if the **decision** resulting from evaluating the **policy** or **policy**
3537 **set** does not match the **decision** resulting from evaluating an enclosing **policy set**.

3538 If the **PDP's** evaluation is viewed as a tree of **policy sets** and **policies**, each of which returns
3539 "Permit" or "Deny", then the set of **obligations** returned by the **PDP** to the **PEP** will include only the
3540 **obligations** associated with those paths where the **effect** at each level of evaluation is the same as
3541 the **effect** being returned by the **PDP**. In situations where any lack of determinism is unacceptable,
3542 a deterministic combining algorithm, such as ordered-deny-overrides, should be used.

3543 Also, see Section 7.1.

3546 7.15. Exception handling

3547 XACML specifies behaviour for the **PDP** in the following situations.

3548

7.15.1. Unsupported functionality

3549 If the **PDP** attempts to evaluate a **policy set** or **policy** that contains an optional element type or
3550 function that the **PDP** does not support, then the **PDP** SHALL return a <Decision> value of
3551 "Indeterminate". If a <StatusCode> element is also returned, then its value SHALL be
3552 "urn:oasis:names:tc:xacml:1.0:status:syntax-error" in the case of an unsupported element type, and
3553 "urn:oasis:names:tc:xacml:1.0:status:processing-error" in the case of an unsupported function.

3554

7.15.2. Syntax and type errors

3555 If a **policy** that contains invalid syntax is evaluated by the XACML **PDP** at the time a **decision**
3556 **request** is received, then the result of that **policy** SHALL be "Indeterminate" with a StatusCode
3557 value of "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

3558 If a **policy** that contains invalid static data-types is evaluated by the XACML **PDP** at the time a
3559 **decision request** is received, then the result of that **policy** SHALL be "Indeterminate" with a
3560 StatusCode value of "urn:oasis:names:tc:xacml:1.0:status:processing-error".

3561

7.15.3. Missing attributes

3562 The absence of matching **attributes** in the request **context** for any of the **attribute** designators or
3563 selectors that are found in the **policy** SHALL result in a <Decision> element containing the
3564 "Indeterminate" value, as described in Sections 5.37 and 5.42. If, in this case, and a status code is
3565 supplied, then the value

3566 "urn:oasis:names:tc:xacml:1.0:status:missing-attribute"

3567 SHALL be used, to indicate that more information is needed in order for a definitive decision to be
3568 rendered. In this case, the <Status> element MAY list the names and data-types of any
3569 **attributes** of the **subjects**, **resource**, **action** or **environment** that are needed by the **PDP** to refine
3570 its decision (see Section 6.16). A **PEP** MAY resubmit a refined request **context** in response to a
3571 <Decision> element contents of "Indeterminate" with a status code of

3572 "urn:oasis:names:tc:xacml:1.0:missing-attribute"

3573 by adding **attribute** values for the **attribute** names that were listed in the previous response. When
3574 the **PDP** returns a <Decision> element contents of "Indeterminate", with a status code of

3575 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3576 it MUST NOT list the names and data-types of any **attribute** of the **subject**, **resource**, **action** or
3577 **environment** for which values were supplied in the original request. Note, this requirement forces
3578 the **PDP** to eventually return an **authorization decision** of "Permit", "Deny" or "Indeterminate" with
3579 some other status code, in response to successively-refined requests.

3580

8. XACML extensibility points (non-normative)

3581 This section describes the points within the XACML model and schema where extensions can be
3582 added

3583 **8.1. Extensible XML attribute types**

3584 The following XML attributes have values that are URIs. These may be extended by the creation of
3585 new URIs associated with new semantics for these attributes.

3586 `AttributeId`,

3587 `DataType`,

3588 `FunctionId`,

3589 `MatchId`,

3590 `ObligationId`,

3591 `PolicyCombiningAlgId`,

3592 `RuleCombiningAlgId`,

3593 `StatusCode`,

3594 `SubjectCategory`.

3595 See Section 5 for definitions of these attribute types.

3596 **8.2. Structured attributes**

3597 `<xacml:AttributeValue>` and `<xacml-context:AttributeValue>` elements MAY
3598 contain an instance of a structured XML data-type. Section 7.2.1 describes a number of standard
3599 techniques to identify data items within such a structured attribute. Listed here are some additional
3600 techniques that require XACML extensions.

3601 1. For a given structured data-type, a community of XACML users MAY define new attribute
3602 identifiers for each leaf sub-element of the structured data-type that has a type conformant
3603 with one of the XACML-defined primitive data-types. Using these new attribute identifiers,
3604 the **PEPs** or **context handlers** used by that community of users can flatten instances of
3605 the structured data-type into a sequence of individual `<Attribute>` elements. Each such
3606 `<Attribute>` element can be compared using the XACML-defined functions. Using this
3607 method, the structured data-type itself never appears in an `<xacml-`
3608 `context:AttributeValue>` element.

3609 2. A community of XACML users MAY define a new function that can be used to compare a
3610 value of the structured data-type against some other value. This method may only be used
3611 by **PDPs** that support the new function.

3612 **9. Security and privacy considerations (non-** 3613 **normative)**

3614 This section identifies possible security and privacy compromise scenarios that should be
3615 considered when implementing an XACML-based system. The section is informative only. It is left
3616 to the implementer to decide whether these compromise scenarios are practical in their
3617 environment and to select appropriate safeguards.

3618

9.1. Threat model

3619 We assume here that the adversary has access to the communication channel between the
3620 XACML actors and is able to interpret, insert, delete and modify messages or parts of messages.

3621 Additionally, an actor may use information from a former message maliciously in subsequent
3622 transactions. It is further assumed that *rules* and *policies* are only as reliable as the actors that
3623 create and use them. Thus it is incumbent on each actor to establish appropriate trust in the other
3624 actors upon which it relies. Mechanisms for trust establishment are outside the scope of this
3625 specification.

3626 The messages that are transmitted between the actors in the XACML model are susceptible to
3627 attack by malicious third parties. Other points of vulnerability include the *PEP*, the *PDP* and the
3628 *PAP*. While some of these entities are not strictly within the scope of this specification, their
3629 compromise could lead to the compromise of *access control* enforced by the *PEP*.

3630 It should be noted that there are other components of a distributed system that may be
3631 compromised, such as an operating system and the domain-name system (DNS) that are outside
3632 the scope of this discussion of threat models. Compromise in these components may also lead to a
3633 policy violation.

3634 The following sections detail specific compromise scenarios that may be relevant to an XACML
3635 system.

3636

9.1.1. Unauthorized disclosure

3637 XACML does not specify any inherent mechanisms to protect the confidentiality of the messages
3638 exchanged between actors. Therefore, an adversary could observe the messages in transit. Under
3639 certain security policies, disclosure of this information is a violation. Disclosure of *attributes* or the
3640 types of *decision requests* that a *subject* submits may be a breach of privacy policy. In the
3641 commercial sector, the consequences of unauthorized disclosure of personal data may range from
3642 embarrassment to the custodian to imprisonment and large fines in the case of medical or financial
3643 data.

3644 Unauthorized disclosure is addressed by confidentiality safeguards.

3645

9.1.2. Message replay

3646 A message replay attack is one in which the adversary records and replays legitimate messages
3647 between XACML actors. This attack may lead to denial of service, the use of out-of-date
3648 information or impersonation.

3649 Prevention of replay attacks requires the use of message freshness safeguards.

3650 Note that encryption of the message does not mitigate a replay attack since the message is simply
3651 replayed and does not have to be understood by the adversary.

3652

9.1.3. Message insertion

3653 A message insertion attack is one in which the adversary inserts messages in the sequence of
3654 messages between XACML actors.

3655 The solution to a message insertion attack is to use mutual authentication and message sequence
3656 integrity safeguards between the actors. It should be noted that just using SSL mutual
3657 authentication is not sufficient. This only proves that the other party is the one identified by the

3658 subject of the X.509 certificate. In order to be effective, it is necessary to confirm that the certificate
3659 subject is authorized to send the message.

3660 **9.1.4. Message deletion**

3661 A message deletion attack is one in which the adversary deletes messages in the sequence of
3662 messages between XACML actors. Message deletion may lead to denial of service. However, a
3663 properly designed XACML system should not render an incorrect authorization decision as a result
3664 of a message deletion attack.

3665 The solution to a message deletion attack is to use message sequence integrity safeguards
3666 between the actors.

3667 **9.1.5. Message modification**

3668 If an adversary can intercept a message and change its contents, then they may be able to alter an
3669 **authorization decision**. A message integrity safeguard can prevent a successful message
3670 modification attack.

3671 **9.1.6. NotApplicable results**

3672 A result of "NotApplicable" means that the **PDP** could not locate a **policy** whose **target** matched
3673 the information in the **decision request**. In general, it is highly recommended that a "Deny" **effect**
3674 **policy** be used, so that when a **PDP** would have returned "NotApplicable", a result of "Deny" is
3675 returned instead.

3676 In some security models, however, such as those found in many Web Servers, an **authorization**
3677 **decision** of "NotApplicable" is treated as equivalent to "Permit". There are particular security
3678 considerations that must be taken into account for this to be safe. These are explained in the
3679 following paragraphs.

3680 If "NotApplicable" is to be treated as "Permit", it is vital that the matching algorithms used by the
3681 **policy** to match elements in the **decision request** be closely aligned with the data syntax used by
3682 the applications that will be submitting the **decision request**. A failure to match will result in
3683 "NotApplicable" and be treated as "Permit". So an unintended failure to match may allow
3684 unintended **access**.

3685 Commercial http responders allow a variety of syntaxes to be treated equivalently. The "%" can be
3686 used to represent characters by hex value. The URL path "/../" provides multiple ways of specifying
3687 the same value. Multiple character sets may be permitted and, in some cases, the same printed
3688 character can be represented by different binary values. Unless the matching algorithm used by
3689 the policy is sophisticated enough to catch these variations, unintended access may be permitted.

3690 It may be safe to treat "NotApplicable" as "Permit" only in a closed environment where all
3691 applications that formulate a **decision request** can be guaranteed to use the exact syntax
3692 expected by the **policies**. In a more open environment, where **decision requests** may be received
3693 from applications that use any legal syntax, it is strongly recommended that "NotApplicable" NOT
3694 be treated as "Permit" unless matching rules have been very carefully designed to match all
3695 possible applicable inputs, regardless of syntax or type variations. Note, however, that according to
3696 Section 7.1, a **PEP** must deny **access** unless it receives an explicit "Permit" **authorization**
3697 **decision**.

3698 **9.1.7. Negative rules**

3699 A negative **rule** is one that is based on a **predicate** not being "True". If not used with care,
3700 negative **rules** can lead to a policy violation, therefore some authorities recommend that they not

3701 be used. However, negative **rules** can be extremely efficient in certain cases, so XACML has
3702 chosen to include them. Nevertheless, it is recommended that they be used with care and avoided
3703 if possible.

3704 A common use for negative **rules** is to deny **access** to an individual or subgroup when their
3705 membership in a larger group would otherwise permit them access. For example, we might want to
3706 write a **rule** that allows all Vice Presidents to see the unpublished financial data, except for Joe,
3707 who is only a Ceremonial Vice President and can be indiscreet in his communications. If we have
3708 complete control over the administration of **subject attributes**, a superior approach would be to
3709 define "Vice President" and "Ceremonial Vice President" as distinct groups and then define **rules**
3710 accordingly. However, in some environments this approach may not be feasible. (It is worth noting
3711 in passing that, generally speaking, referring to individuals in **rules** does not scale well. Generally,
3712 shared **attributes** are preferred.)

3713 If not used with care, negative **rules** can lead to policy violation in two common cases. They are:
3714 when **attributes** are suppressed and when the base group changes. An example of suppressed
3715 **attributes** would be if we have a policy that **access** should be permitted, *unless* the **subject** is a
3716 credit risk. If it is possible that the **attribute** of being a credit risk may be unknown to the **PDP** for
3717 some reason, then unauthorized **access** may result. In some environments, the **subject** may be
3718 able to suppress the publication of **attributes** by the application of privacy controls, or the server or
3719 repository that contains the information may be unavailable for accidental or intentional reasons.

3720 An example of a changing base group would be if there is a policy that everyone in the engineering
3721 department may change software source code, except for secretaries. Suppose now that the
3722 department was to merge with another engineering department and the intent is to maintain the
3723 same policy. However, the new department also includes individuals identified as administrative
3724 assistants, who ought to be treated in the same way as secretaries. Unless the policy is altered,
3725 they will unintentionally be permitted to change software source code. Problems of this type are
3726 easy to avoid when one individual administers all **policies**, but when administration is distributed,
3727 as XACML allows, this type of situation must be explicitly guarded against.

3728 **9.2. Safeguards**

3729 **9.2.1. Authentication**

3730 Authentication provides the means for one party in a transaction to determine the identity of the
3731 other party in the transaction. Authentication may be in one direction, or it may be bilateral.

3732 Given the sensitive nature of **access control** systems, it is important for a **PEP** to authenticate the
3733 identity of the **PDP** to which it sends **decision requests**. Otherwise, there is a risk that an
3734 adversary could provide false or invalid **authorization decisions**, leading to a policy violation.

3735 It is equally important for a **PDP** to authenticate the identity of the **PEP** and assess the level of trust
3736 to determine what, if any, sensitive data should be passed. One should keep in mind that even
3737 simple "Permit" or "Deny" responses could be exploited if an adversary were allowed to make
3738 unlimited requests to a **PDP**.

3739 Many different techniques may be used to provide authentication, such as co-located code, a
3740 private network, a VPN or digital signatures. Authentication may also be performed as part of the
3741 communication protocol used to exchange the **contexts**. In this case, authentication may be
3742 performed either at the message level or at the session level.

3743 **9.2.2. Policy administration**

3744 If the contents of **policies** are exposed outside of the **access control** system, potential **subjects**
3745 may use this information to determine how to gain unauthorized **access**.

3746 To prevent this threat, the repository used for the storage of **policies** may itself require **access**
3747 **control**. In addition, the <Status> element should be used to return values of missing **attributes**
3748 only when exposure of the identities of those **attributes** will not compromise security.

3749 **9.2.3. Confidentiality**

3750 Confidentiality mechanisms ensure that the contents of a message can be read only by the desired
3751 recipients and not by anyone else who encounters the message while it is in transit. There are two
3752 areas in which confidentiality should be considered: one is confidentiality during transmission; the
3753 other is confidentiality within a <Policy> element.

3754 **9.2.3.1. Communication confidentiality**

3755 In some environments it is deemed good practice to treat all data within an **access control** system
3756 as confidential. In other environments, **policies** may be made freely available for distribution,
3757 inspection and audit. The idea behind keeping **policy** information secret is to make it more difficult
3758 for an adversary to know what steps might be sufficient to obtain unauthorized **access**. Regardless
3759 of the approach chosen, the security of the **access control** system should not depend on the
3760 secrecy of the **policy**.

3761 Any security considerations related to transmitting or exchanging XACML <Policy> elements are
3762 outside the scope of the XACML standard. While it is often important to ensure that the integrity
3763 and confidentiality of <Policy> elements is maintained when they are exchanged between two
3764 parties, it is left to the implementers to determine the appropriate mechanisms for their
3765 environment.

3766 Communications confidentiality can be provided by a confidentiality mechanism, such as SSL.
3767 Using a point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points
3768 is compromised.

3769 **9.2.3.2. Statement level confidentiality**

3770 In some cases, an implementation may want to encrypt only parts of an XACML <Policy>
3771 element.

3772 The XML Encryption Syntax and Processing Candidate Recommendation from W3C can be used
3773 to encrypt all or parts of an XML document. This specification is recommended for use with
3774 XACML.

3775 It should go without saying that if a repository is used to facilitate the communication of cleartext
3776 (i.e., unencrypted) **policy** between the **PAP** and **PDP**, then a secure repository should be used to
3777 store this sensitive data.

3778 **9.2.4. Policy integrity**

3779 The XACML **policy**, used by the **PDP** to evaluate the request **context**, is the heart of the system.
3780 Therefore, maintaining its integrity is essential. There are two aspects to maintaining the integrity of
3781 the **policy**. One is to ensure that <Policy> elements have not been altered since they were
3782 originally created by the **PAP**. The other is to ensure that <Policy> elements have not been
3783 inserted or deleted from the set of **policies**.

3784 In many cases, both aspects can be achieved by ensuring the integrity of the actors and
3785 implementing session-level mechanisms to secure the communication between actors. The
3786 selection of the appropriate mechanisms is left to the implementers. However, when **policy** is
3787 distributed between organizations to be acted on at a later time, or when the **policy** travels with the

3788 protected resource, it would be useful to sign the **policy**. In these cases, the XML Signature
3789 Syntax and Processing standard from W3C is recommended to be used with XACML.

3790 Digital signatures should only be used to ensure the integrity of the statements. Digital signatures
3791 should not be used as a method of selecting or evaluating **policy**. That is, the **PDP** should not
3792 request a **policy** based on who signed it or whether or not it has been signed (as such a basis for
3793 selection would, itself, be a matter of policy). However, the **PDP** must verify that the key used to
3794 sign the **policy** is one controlled by the purported issuer of the **policy**. The means to do this are
3795 dependent on the specific signature technology chosen and are outside the scope of this document.

3796 **9.2.5. Policy identifiers**

3797 Since **policies** can be referenced by their identifiers, it is the responsibility of the **PAP** to ensure
3798 that these are unique. Confusion between identifiers could lead to misidentification of the
3799 **applicable policy**. This specification is silent on whether a **PAP** must generate a new identifier
3800 when a **policy** is modified or may use the same identifier in the modified **policy**. This is a matter of
3801 administrative practice. However, care must be taken in either case. If the identifier is reused,
3802 there is a danger that other **policies** or **policy sets** that reference it may be adversely affected.
3803 Conversely, if a new identifier is used, these other **policies** may continue to use the prior **policy**,
3804 unless it is deleted. In either case the results may not be what the **policy** administrator intends.

3805 **9.2.6. Trust model**

3806 Discussions of authentication, integrity and confidentiality safeguards necessarily assume an
3807 underlying trust model: how can one actor come to believe that a given key is uniquely associated
3808 with a specific, identified actor so that the key can be used to encrypt data for that actor or verify
3809 signatures (or other integrity structures) from that actor? Many different types of trust model exist,
3810 including strict hierarchies, distributed authorities, the Web, the bridge and so on.

3811 It is worth considering the relationships between the various actors of the **access control** system in
3812 terms of the interdependencies that do and do not exist.

- 3813 • None of the entities of the authorization system are dependent on the **PEP**. They may
3814 collect data from it, for example authentication data, but are responsible for verifying it
3815 themselves.
- 3816 • The correct operation of the system depends on the ability of the **PEP** to actually enforce
3817 **policy** decisions.
- 3818 • The **PEP** depends on the **PDP** to correctly evaluate **policies**. This in turn implies that the
3819 **PDP** is supplied with the correct inputs. Other than that, the **PDP** does not depend on the
3820 **PEP**.
- 3821 • The **PDP** depends on the **PAP** to supply appropriate policies. The **PAP** is not dependent
3822 on other components.

3823 **9.2.7. Privacy**

3824 It is important to be aware that any transactions that occur with respect to **access control** may
3825 reveal private information about the actors. For example, if an XACML **policy** states that certain
3826 data may only be read by **subjects** with "Gold Card Member" status, then any transaction in which
3827 a **subject** is permitted **access** to that data leaks information to an adversary about the **subject's**
3828 status. Privacy considerations may therefore lead to encryption and/or to access control
3829 requirements surrounding the enforcement of XACML **policy** instances themselves: confidentiality-
3830 protected channels for the request/response protocol messages, protection of **subject attributes** in
3831 storage and in transit, and so on.

3832 Selection and use of privacy mechanisms appropriate to a given environment are outside the scope
3833 of XACML. The decision regarding whether, how and when to deploy such mechanisms is left to
3834 the implementers associated with the environment.

3835 **10. Conformance (normative)**

3836 **10.1. Introduction**

3837 The XACML specification addresses the following aspect of conformance:

3838 The XACML specification defines a number of functions, etc. that have somewhat special
3839 application, therefore they are not required to be implemented in an implementation that claims to
3840 conform with the OASIS standard.

3841 **10.2. Conformance tables**

3842 This section lists those portions of the specification that **MUST** be included in an implementation of
3843 a **PDP** that claims to conform with XACML v2.0. A set of test cases has been created to assist in
3844 this process. These test cases are hosted by Sun Microsystems and can be located from the
3845 XACML Web page. The site hosting the test cases contains a full description of the test cases and
3846 how to execute them.

3847 Note: "M" means mandatory-to-implement. "O" means optional.

3848 **10.2.1. Schema elements**

3849 The implementation **MUST** support those schema elements that are marked "M".

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

xacml:CombinerParameter	O
xacml:Condition	M
xacml:Description	M
xacml:Environment	M
xacml:EnvironmentMatch	M
xacml:EnvironmentAttributeDesignator	M
xacml:Environments	M
xacml:Expression	M
xacml:Function	M
xacml:Obligation	O
xacml:Obligations	O
xacml:Policy	M
xacml:PolicyCombinerParameters	O
xacml:PolicyDefaults	O
xacml:PolicyIdReference	M
xacml:PolicySet	M
xacml:PolicySetDefaults	O
xacml:PolicySetIdReference	M
xacml:Resource	M
xacml:ResourceAttributeDesignator	M
xacml:ResourceMatch	M
xacml:Resources	M
xacml:Rule	M
xacml:RuleCombinerParameters	O
xacml:Subject	M
xacml:SubjectMatch	M
xacml:Subjects	M
xacml:Target	M
xacml:VariableDefinition	M
xacml:VariableReference	M
xacml:XPathVersion	O

3850 **10.2.2. Identifier Prefixes**

3851 The following identifier prefixes are reserved by XACML.

Identifier
urn:oasis:names:tc:xacml:2.0
urn:oasis:names:tc:xacml:2.0:conformance-test
urn:oasis:names:tc:xacml:2.0:context
urn:oasis:names:tc:xacml:2.0:example
urn:oasis:names:tc:xacml:1.0:function
urn:oasis:names:tc:xacml:2.0:function
urn:oasis:names:tc:xacml:2.0:policy
urn:oasis:names:tc:xacml:1.0:subject
urn:oasis:names:tc:xacml:1.0:resource
urn:oasis:names:tc:xacml:1.0:action
urn:oasis:names:tc:xacml:1.0:environment
urn:oasis:names:tc:xacml:1.0:status

3852 **10.2.3. Algorithms**

3853 The implementation MUST include the rule- and policy-combining algorithms associated with the
 3854 following identifiers that are marked "M".

3855

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

3856 **10.2.4. Status Codes**

3857 Implementation support for the <StatusCode> element is optional, but if the element is supported,
3858 then the following status codes must be supported and must be used in the way XACML has
3859 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

3860 **10.2.5. Attributes**

3861 The implementation MUST support the **attributes** associated with the following identifiers as
3862 specified by XACML. If values for these **attributes** are not present in the **decision request**, then
3863 their values MUST be supplied by the **context handler**. So, unlike most other **attributes**, their
3864 semantics are not transparent to the **PDP**.

Identifier	M/O
urn:oasis:names:tc:xacml:1.0:environment:current-time	M
urn:oasis:names:tc:xacml:1.0:environment:current-date	M
urn:oasis:names:tc:xacml:1.0:environment:current-dateTime	M

3865 **10.2.6. Identifiers**

3866 The implementation MUST use the **attributes** associated with the following identifiers in the way
3867 XACML has defined. This requirement pertains primarily to implementations of a **PAP** or **PEP** that
3868 uses XACML, since the semantics of the attributes are transparent to the **PDP**.

3869
3870
3871
3872

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

3873 **10.2.7. Data-types**

3874 The implementation MUST support the data-types associated with the following identifiers marked
3875 "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

3876 **10.2.8. Functions**

3877 The implementation MUST properly process those functions associated with the identifiers marked
3878 with an "M".

Function	M/O
urn:oasis:names:tc:xacml:1.0:function:string-equal	M
urn:oasis:names:tc:xacml:1.0:function:boolean-equal	M
urn:oasis:names:tc:xacml:1.0:function:integer-equal	M
urn:oasis:names:tc:xacml:1.0:function:double-equal	M
urn:oasis:names:tc:xacml:1.0:function:date-equal	M
urn:oasis:names:tc:xacml:1.0:function:time-equal	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-equal	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-equal	M

urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-equal	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-equal	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-equal	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal	M
urn:oasis:names:tc:xacml:1.0:function:integer-add	M
urn:oasis:names:tc:xacml:1.0:function:double-add	M
urn:oasis:names:tc:xacml:1.0:function:integer-subtract	M
urn:oasis:names:tc:xacml:1.0:function:double-subtract	M
urn:oasis:names:tc:xacml:1.0:function:integer-multiply	M
urn:oasis:names:tc:xacml:1.0:function:double-multiply	M
urn:oasis:names:tc:xacml:1.0:function:integer-divide	M
urn:oasis:names:tc:xacml:1.0:function:double-divide	M
urn:oasis:names:tc:xacml:1.0:function:integer-mod	M
urn:oasis:names:tc:xacml:1.0:function:integer-abs	M
urn:oasis:names:tc:xacml:1.0:function:double-abs	M
urn:oasis:names:tc:xacml:1.0:function:round	M
urn:oasis:names:tc:xacml:1.0:function:floor	M
urn:oasis:names:tc:xacml:1.0:function:string-normalize-space	M
urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case	M
urn:oasis:names:tc:xacml:1.0:function:double-to-integer	M
urn:oasis:names:tc:xacml:1.0:function:integer-to-double	M
urn:oasis:names:tc:xacml:1.0:function:or	M
urn:oasis:names:tc:xacml:1.0:function:and	M
urn:oasis:names:tc:xacml:1.0:function:n-of	M
urn:oasis:names:tc:xacml:1.0:function:not	M
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:integer-less-than	M
urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:double-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:double-less-than	M
urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-dayTimeDuration	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:date-subtract-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:string-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:string-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:string-less-than	M
urn:oasis:names:tc:xacml:1.0:function:string-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:time-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:time-less-than	M
urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal	M
urn:oasis:names:tc:xacml:2.0:function:time-in-range	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:date-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal	M

urn:oasis:names:tc:xacml:1.0:function:date-less-than	M
urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:string-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:string-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:string-is-in	M
urn:oasis:names:tc:xacml:1.0:function:string-bag	M
urn:oasis:names:tc:xacml:1.0:function:boolean-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:boolean-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:boolean-is-in	M
urn:oasis:names:tc:xacml:1.0:function:boolean-bag	M
urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:integer-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:integer-is-in	M
urn:oasis:names:tc:xacml:1.0:function:integer-bag	M
urn:oasis:names:tc:xacml:1.0:function:double-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:double-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:double-is-in	M
urn:oasis:names:tc:xacml:1.0:function:double-bag	M
urn:oasis:names:tc:xacml:1.0:function:time-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:time-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:time-is-in	M
urn:oasis:names:tc:xacml:1.0:function:time-bag	M
urn:oasis:names:tc:xacml:1.0:function:date-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:date-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:date-is-in	M
urn:oasis:names:tc:xacml:1.0:function:date-bag	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-is-in	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-bag	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-is-in	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-bag	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-is-in	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-bag	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-is-in	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-bag	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-is-in	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-bag	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-is-in	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-bag	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-is-in	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-bag	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-is-in	M

urn:oasis:names:tc:xacml:1.0:function:rfc822Name-bag	M
urn:oasis:names:tc:xacml:2.0:function:string-concatenate	M
urn:oasis:names:tc:xacml:2.0:function:uri-string-concatenate	M
urn:oasis:names:tc:xacml:1.0:function:any-of	M
urn:oasis:names:tc:xacml:1.0:function:all-of	M
urn:oasis:names:tc:xacml:1.0:function:any-of-any	M
urn:oasis:names:tc:xacml:1.0:function:all-of-any	M
urn:oasis:names:tc:xacml:1.0:function:any-of-all	M
urn:oasis:names:tc:xacml:1.0:function:all-of-all	M
urn:oasis:names:tc:xacml:1.0:function:map	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-match	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match	M
urn:oasis:names:tc:xacml:1.0:function:regex-string-match	M
urn:oasis:names:tc:xacml:1.0:function:regex-uri-match	M
urn:oasis:names:tc:xacml:1.0:function:regex-ipAddress-match	M
urn:oasis:names:tc:xacml:1.0:function:regex-dnsName-match	M
urn:oasis:names:tc:xacml:1.0:function:regex-rfc822Name-match	M
urn:oasis:names:tc:xacml:1.0:function:regex-x500Name-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

3879

3880 11. References

3881 [DS] D. Eastlake et al., *XML-Signature Syntax and Processing*,
3882 <http://www.w3.org/TR/xmldsig-core/>, World Wide Web Consortium.

3883 [Hancock] Hancock, "Polymorphic Type Checking", in Simon L. Peyton Jones,
3884 "Implementation of Functional Programming Languages", Section 8,
3885 Prentice-Hall International, 1987

3886 [Haskell] Haskell, a purely functional language. Available at
3887 <http://www.haskell.org/>

3888 [Hier] Anderson A, ed., *The XACML Profile for Hierarchical Resources*, OASIS
3889 Access Control TC, Committee Draft 01, 16 Sep 2004, <http://www.oasis-open.org/committees/xacml>

3891 [Hinton94] Hinton, H, M, Lee, E, S, The Compatibility of Policies, Proceedings 2nd
3892 ACM Conference on Computer and Communications Security, Nov 1994,
3893 Fairfax, Virginia, USA.

3894	[IEEE754]	IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-7653-8, IEEE Product No. SH10116-TBR
3895		
3896	[ISO10181-3]	ISO/IEC 10181-3:1996 Information technology – Open Systems Interconnection -- Security frameworks for open systems: Access control framework.
3897		
3898		
3899	[Kudo00]	Kudo M and Hada S, XML document security based on provisional authorization, Proceedings of the Seventh ACM Conference on Computer and Communications Security, Nov 2000, Athens, Greece, pp 87-96.
3900		
3901		
3902	[LDAP-1]	RFC2256, A summary of the X500(96) User Schema for use with LDAPv3, Section 5, M Wahl, December 1997 http://www.ietf.org/rfc/rfc2798.txt
3903		
3904	[LDAP-2]	RFC2798, Definition of the inetOrgPerson, M. Smith, April 2000
3905		http://www.ietf.org/rfc/rfc2798.txt
3906	[MathML]	Mathematical Markup Language (MathML), Version 2.0, W3C Recommendation, 21 February 2001. Available at:
3907		http://www.w3.org/TR/MathML2/
3908		
3909	[Multi]	Anderson A, ed., <i>The XACML Profile for Requests for Multiple Resources</i> , OASIS Access Control TC, Working Draft 02, 4 June 2004,
3910		http://www.oasis-open.org/committees/xacml
3911		
3912	[Perritt93]	Perritt, H. Knowbots, Permissions Headers and Contract Law, Conference on Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment, April 1993. Available at:
3913		http://www.ifla.org/documents/infopol/copyright/perh2.txt
3914		
3915		
3916	[RBAC]	Role-Based Access Controls, David Ferraiolo and Richard Kuhn, 15th National Computer Security Conference, 1992. Available at:
3917		http://csrc.nist.gov/rbac
3918		
3919	[RegEx]	XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, Appendix D. Available at: http://www.w3.org/TR/xmlschema-0/
3920		
3921	[RFC2119]	S. Bradner, <i>Key words for use in RFCs to Indicate Requirement Levels</i> , http://www.ietf.org/rfc/rfc2119.txt , IETF RFC 2119, March 1997
3922		
3923	[RFC2396]	Berners-Lee T, Fielding R, Masinter L, Uniform Resource Identifiers (URI): Generic Syntax. Available at: http://www.ietf.org/rfc/rfc2396.txt
3924		
3925	[RFC2732]	Hinden R, Carpenter B, Masinter L, Format for Literal IPv6 Addresses in URL's. Available at: http://www.ietf.org/rfc/rfc2732.txt
3926		
3927	[RFC3198]	IETF RFC 3198: Terminology for Policy-Based Management, November 2001. http://www.ietf.org/rfc/rfc3198.txt
3928		
3929	[SAML]	Security Assertion Markup Language available from http://www.oasis-open.org/committees/security/#documents
3930		
3931	[Sloman94]	Sloman, M. Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Volume 2, part 4. Plenum Press. 1994.
3932		
3933		
3934	[XACMLv1.0]	Extensible access control markup language (XACML) Version 1.0. OASIS Standard. 18 February 2003. Available at: http://www.oasis-open.org/apps/org/workgroup/xacml/download.php/940/oasis-xacml-1.0.pdf
3935		
3936		
3937		
3938	[XACMLv1.1]	Extensible access control markup language (XACML) Version 1.1. OASIS Committee Specification. 7 August 2003. Available at: http://www.oasis-open.org/apps/org/workgroup/xacml/download.php/4104/cs-xacml-specification-1.1.pdf
3939		
3940		
3941		

3942	[XF]	XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft
3943		16 August 2002. Available at: http://www.w3.org/TR/2002/WD-xquery-operators-20020816
3944		
3945	[XS]	XML Schema, parts 1 and 2. Available at:
3946		http://www.w3.org/TR/xmlschema-1/ and
3947		http://www.w3.org/TR/xmlschema-2/
3948	[XPath]	XML Path Language (XPath), Version 1.0, W3C Recommendation 16
3949		November 1999. Available at: http://www.w3.org/TR/xpath
3950	[XSLT]	XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16
3951		November 1999. Available at: http://www.w3.org/TR/xslt
3952		

3953 Appendix A. Data-types and functions (normative)

3954 A.1. Introduction

3955 This section specifies the data-types and functions used in XACML to create *predicates* for
3956 *conditions* and *target* matches.

3957 This specification combines the various standards set forth by IEEE and ANSI for string
3958 representation of numeric values, as well as the evaluation of arithmetic functions. It describes the
3959 primitive data-types and *bags*. The standard functions are named and their operational semantics
3960 are described.

3961 A.2. Data-types

3962 Although XML instances represent all data-types as strings, an XACML *PDP* must reason about
3963 types of data that, while they have string representations, are not just strings. Types such as
3964 Boolean, integer and double **MUST** be converted from their XML string representations to values
3965 that can be compared with values in their domain of discourse, such as numbers. The following
3966 primitive data-types are specified for use with XACML and have explicit data representations:

- 3967 • <http://www.w3.org/2001/XMLSchema#string>
- 3968 • <http://www.w3.org/2001/XMLSchema#boolean>
- 3969 • <http://www.w3.org/2001/XMLSchema#integer>
- 3970 • <http://www.w3.org/2001/XMLSchema#double>
- 3971 • <http://www.w3.org/2001/XMLSchema#time>
- 3972 • <http://www.w3.org/2001/XMLSchema#date>
- 3973 • <http://www.w3.org/2001/XMLSchema#dateTime>
- 3974 • <http://www.w3.org/2001/XMLSchema#anyURI>
- 3975 • <http://www.w3.org/2001/XMLSchema#hexBinary>
- 3976 • <http://www.w3.org/2001/XMLSchema#base64Binary>
- 3977 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration>
- 3978 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration>
- 3979 • <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
- 3980 • <urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name>
- 3981 • <urn:oasis:names:tc:xacml:1.0:data-type:ipAddress>
- 3982 • <urn:oasis:names:tc:xacml:1.0:data-type:dnsName>

3983 For the sake of improved interoperability, it is RECOMMENDED that all time references be in UTC
3984 time.

3985 An XACML **PDP** SHALL be capable of converting string representations into various primitive data-
3986 types. For integers and doubles, XACML SHALL use the conversions described in [IEEE754].

3987 XACML defines three data-types; these are:

3988 "urn:oasis:names:tc:xacml:1.0:data-type:x500Name",

3989 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"

3990 "urn:oasis:names:tc:xacml:1.0:data-type:ipAddress"

3991 "urn:oasis:names:tc:xacml:1.0:data-type:dnsName" and

3992 These types represent identifiers for subjects or resources and appear in several standard
3993 applications, such as TLS/SSL and electronic mail.

3994 **X.500 directory name**

3995 The "urn:oasis:names:tc:xacml:1.0:data-type:x500Name" primitive type represents an ITU-T Rec.
3996 X.520 Distinguished Name. The valid syntax for such a name is described in IETF RFC 2253
3997 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names"

3998 **RFC 822 name**

3999 The "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" primitive type represents an electronic
4000 mail address. The valid syntax for such a name is described in IETF RFC 2821, Section 4.1.2,
4001 Command Argument Syntax, under the term "Mailbox".

4002 **IP address**

4003 The "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" primitive type represents an IPv4 or IPv6
4004 network address, with optional mask and optional port or port range. The syntax SHALL be:

4005
4006 ipAddress = address ["/" mask] [":" [portrange]]

4007
4008 For an IPv4 address, the address and mask are formatted in accordance with the syntax for a
4009 "host" in IETF RFC 2396 "Uniform Resource Identifiers (URI): Generic Syntax", section 3.2.
4010 For an IPv6 address, the address and mask are formatted in accordance with the syntax for an
4011 "ipv6reference" in IETF RFC 2732 "Format for Literal IPv6 Addresses in URL's". (Note that an IPv6
4012 address or mask, in this syntax, is enclosed in literal "[" "]" brackets.)

4013

4014 **DNS name**

4015 The "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" primitive type represents a Domain Name
4016 Service (DNS) host name, with optional port or port range. The syntax SHALL be:

4017
4018 dnsName = hostname [":" portrange]

4019

4020 The hostname is formatted in accordance with IETF RFC 2396 "Uniform Resource Identifiers (URI):
4021 Generic Syntax", section 3.2, except that a wildcard "*" may be used in the left-most component of
4022 the hostname to indicate "any subdomain" under the domain specified to its right.

4023

4024 For both the "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" and
4025 "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" data-types, the port or port range syntax
4026 SHALL be

4027

4028 portrange = portnumber | "-"portnumber | portnumber-"[portnumber]

4029

4030 where "portnumber" is a decimal port number. If the port number is of the form "-x", where "x" is a
4031 port number, then the range is all ports numbered "x" and below. If the port number is of the form

4032 "x-", then the range is all ports numbered "x" and above. [This syntax is taken from the Java
4033 SocketPermission.]

4034 **A.3. Functions**

4035 XACML specifies the following functions. If an argument of one of these functions were to evaluate
4036 to "Indeterminate", then the function SHALL be set to "Indeterminate".

4037 **A.3.1 Equality predicates**

4038 The following functions are the *equality* functions for the various primitive types. Each function for a
4039 particular data-type follows a specified standard convention for that data-type.

- 4040 • urn:oasis:names:tc:xacml:1.0:function:string-equal

4041 This function SHALL take two arguments of data-type
4042 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4043 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and
4044 only if the value of both of its arguments are of equal length and each string is determined
4045 to be equal byte-by-byte according to the function "integer-equal". Otherwise, it SHALL
4046 return "False".

- 4047 • urn:oasis:names:tc:xacml:1.0:function:boolean-equal

4048 This function SHALL take two arguments of data-type
4049 "http://www.w3.org/2001/XMLSchema#boolean" and SHALL return an
4050 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and
4051 only if the arguments are equal. Otherwise, it SHALL return "False".

- 4052 • urn:oasis:names:tc:xacml:1.0:function:integer-equal

4053 This function SHALL take two arguments of data-type
4054 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return an
4055 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on
4056 integers according to IEEE 754 [IEEE 754].

- 4057 • urn:oasis:names:tc:xacml:1.0:function:double-equal

4058 This function SHALL take two arguments of data-type
4059 "http://www.w3.org/2001/XMLSchema#double" and SHALL return an
4060 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on
4061 doubles according to IEEE 754 [IEEE 754].

- 4062 • urn:oasis:names:tc:xacml:1.0:function:date-equal

4063 This function SHALL take two arguments of data-type
4064 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4065 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
4066 according to the "op:date-equal" function [XF Section 8.3.11].

- 4067 • urn:oasis:names:tc:xacml:1.0:function:time-equal

4068 This function SHALL take two arguments of data-type
4069 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
4070 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
4071 according to the "op:time-equal" function [XF Section 8.3.14].

- 4072 • urn:oasis:names:tc:xacml:1.0:function:dateTime-equal
- 4073 This function SHALL take two arguments of data-type
 4074 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
 4075 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
 4076 according to the "op:dateTime-equal" function [XF Section 8.3.8].
- 4077 • urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-equal
- 4078 This function SHALL take two arguments of data-type "http://www.w3.org/TR/2002/WD-
 4079 xquery-operators-20020816#dayTimeDuration" and SHALL return an
 4080 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
 4081 according to the "op:dayTimeDuration-equal" function [XF Section 8.3.5]. Note that the
 4082 lexical representation of each argument MUST be converted to a value expressed in
 4083 fractional seconds [XF Section 8.2.2].
- 4084 • urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-equal
- 4085 This function SHALL take two arguments of data-type "http://www.w3.org/TR/2002/WD-
 4086 xquery-operators-20020816#yearMonthDuration" and SHALL return an
 4087 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
 4088 according to the "op:yearMonthDuration-equal" function [XF Section 8.3.2]. Note that the
 4089 lexical representation of each argument MUST be converted to a value expressed in
 4090 integer months [XF Section 8.2.1].
- 4091 • urn:oasis:names:tc:xacml:1.0:function:anyURI-equal
- 4092 This function SHALL take two arguments of data-type
 4093 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return an
 4094 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
 4095 according to the "op:anyURI-equal" function [XF Section 10.2.1].
- 4096 • urn:oasis:names:tc:xacml:1.0:function:x500Name-equal
- 4097 This function SHALL take two arguments of "urn:oasis:names:tc:xacml:1.0:data-
 4098 type:x500Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It
 4099 SHALL return "True" if and only if each Relative Distinguished Name (RDN) in the two
 4100 arguments matches. Otherwise, it SHALL return "False". Two RDNs shall be said to
 4101 match if and only if the result of the following operations is "True"³.
- 4102 1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory
 4103 Access Protocol (v3): UTF-8 String Representation of Distinguished Names".
 - 4104 2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute
 4105 ValuePairs in that RDN in ascending order when compared as octet strings
 4106 (described in ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").
 - 4107 3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key
 4108 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section
 4109 4.1.2.4 "Issuer".
- 4110 • urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal
- 4111 This function SHALL take two arguments of data-type "urn:oasis:names:tc:xacml:1.0:data-
 4112 type:rfc822Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean".
 4113 It SHALL return "True" if and only if the two arguments are equal. Otherwise, it SHALL

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.

- 4114 return "False". An RFC822 name consists of a *local-part* followed by "@" followed by a
 4115 *domain-part*. The *local-part* is case-sensitive, while the *domain-part* (which is usually a
 4116 DNS host name) is not case-sensitive. Perform the following operations:
- 4117 1. Normalize the *domain-part* of each argument to lower case
 - 4118 2. Compare the expressions by applying the function
 4119 "urn:oasis:names:tc:xacml:1.0:function:string-equal" to the normalized arguments.
- 4120 • urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal

4121 This function SHALL take two arguments of data-type
 4122 "http://www.w3.org/2001/XMLSchema#hexBinary" and SHALL return an
 4123 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the octet
 4124 sequences represented by the value of both arguments have equal length and are equal in
 4125 a conjunctive, point-wise, comparison using the
 4126 "urn:oasis:names:tc:xacml:1.0:function:integer-equal" function. Otherwise, it SHALL return
 4127 "False". The conversion from the string representation to an octet sequence SHALL be as
 4128 specified in [XS Section 8.2.15].
 - 4129 • urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal

4130 This function SHALL take two arguments of data-type
 4131 "http://www.w3.org/2001/XMLSchema#base64Binary" and SHALL return an
 4132 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the octet
 4133 sequences represented by the value of both arguments have equal length and are equal in
 4134 a conjunctive, point-wise, comparison using the
 4135 "urn:oasis:names:tc:xacml:1.0:function:integer-equal" function. Otherwise, it SHALL return
 4136 "False". The conversion from the string representation to an octet sequence SHALL be as
 4137 specified in [XS Section 8.2.16].

4138 **A.3.2 Arithmetic functions**

- 4139 All of the following functions SHALL take two arguments of the specified *data-type*, integer or
 4140 double, and SHALL return an element of integer or double data-type, respectively. However, the
 4141 "add" functions MAY take more than two arguments. Each function evaluation SHALL proceed as
 4142 specified by their logical counterparts in IEEE 754 [IEEE 754]. In an expression that contains any
 4143 of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
 4144 "Indeterminate". In the case of the divide functions, if the divisor is zero, then the function SHALL
 4145 evaluate to "Indeterminate".
- 4146 • urn:oasis:names:tc:xacml:1.0:function:integer-add

4147 This function MAY have two or more arguments.
 - 4148 • urn:oasis:names:tc:xacml:1.0:function:double-add

4149 This function MAY have two or more arguments.
 - 4150 • urn:oasis:names:tc:xacml:1.0:function:integer-subtract
 - 4151 • urn:oasis:names:tc:xacml:1.0:function:double-subtract
 - 4152 • urn:oasis:names:tc:xacml:1.0:function:integer-multiply
 - 4153 • urn:oasis:names:tc:xacml:1.0:function:double-multiply
 - 4154 • urn:oasis:names:tc:xacml:1.0:function:integer-divide
 - 4155 • urn:oasis:names:tc:xacml:1.0:function:double-divide

- 4156 • urn:oasis:names:tc:xacml:1.0:function:integer-mod
- 4157 The following functions SHALL take a single argument of the specified *data-type*. The round and
- 4158 floor functions SHALL take a single argument of data-type
- 4159 “http://www.w3.org/2001/XMLSchema#double” and return a value of the data-type
- 4160 “http://www.w3.org/2001/XMLSchema#double”.
- 4161 • urn:oasis:names:tc:xacml:1.0:function:integer-abs
- 4162 • urn:oasis:names:tc:xacml:1.0:function:double-abs
- 4163 • urn:oasis:names:tc:xacml:1.0:function:round
- 4164 • urn:oasis:names:tc:xacml:1.0:function:floor

4165 **A.3.3 String conversion functions**

- 4166 The following functions convert between values of the data-type
4167 “http://www.w3.org/2001/XMLSchema#string” primitive types.
- 4168 • urn:oasis:names:tc:xacml:1.0:function:string-normalize-space
 - 4169 This function SHALL take one argument of data-type
 - 4170 “http://www.w3.org/2001/XMLSchema#string” and SHALL normalize the value by stripping
 - 4171 off all leading and trailing white space characters.
 - 4172 • urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case
 - 4173 This function SHALL take one argument of data-type
 - 4174 “http://www.w3.org/2001/XMLSchema#string” and SHALL normalize the value by
 - 4175 converting each upper case character to its lower case equivalent.

4176 **A.3.4 Numeric data-type conversion functions**

- 4177 The following functions convert between the data-type
4178 “http://www.w3.org/2001/XMLSchema#integer” and “http://www.w3.org/2001/XMLSchema#double”
4179 primitive types.
- 4180 • urn:oasis:names:tc:xacml:1.0:function:double-to-integer
 - 4181 This function SHALL take one argument of data-type
 - 4182 “http://www.w3.org/2001/XMLSchema#double” and SHALL truncate its numeric value to a
 - 4183 whole number and return an element of data-type
 - 4184 “http://www.w3.org/2001/XMLSchema#integer”.
 - 4185 • urn:oasis:names:tc:xacml:1.0:function:integer-to-double
 - 4186 This function SHALL take one argument of data-type
 - 4187 “http://www.w3.org/2001/XMLSchema#integer” and SHALL promote its value to an element
 - 4188 of data-type “http://www.w3.org/2001/XMLSchema#double” with the same numeric value.

4189 **A.3.5 Logical functions**

- 4190 This section contains the specification for logical functions that operate on arguments of data-type
4191 “http://www.w3.org/2001/XMLSchema#boolean”.
- 4192 • urn:oasis:names:tc:xacml:1.0:function:or

4193 This function SHALL return "False" if it has no arguments and SHALL return "True" if at
4194 least one of its arguments evaluates to "True". The order of evaluation SHALL be from first
4195 argument to last. The evaluation SHALL stop with a result of "True" if any argument
4196 evaluates to "True", leaving the rest of the arguments unevaluated.

4197 • urn:oasis:names:tc:xacml:1.0:function:and

4198 This function SHALL return "True" if it has no arguments and SHALL return "False" if one of
4199 its arguments evaluates to "False". The order of evaluation SHALL be from first argument
4200 to last. The evaluation SHALL stop with a result of "False" if any argument evaluates to
4201 "False", leaving the rest of the arguments unevaluated.

4202 • urn:oasis:names:tc:xacml:1.0:function:n-of

4203 The first argument to this function SHALL be of data-type
4204 <http://www.w3.org/2001/XMLSchema#integer>. The remaining arguments SHALL be of
4205 data-type <http://www.w3.org/2001/XMLSchema#boolean>. The first argument specifies the
4206 minimum number of the remaining arguments that MUST evaluate to "True" for the
4207 expression to be considered "True". If the first argument is 0, the result SHALL be "True".
4208 If the number of arguments after the first one is less than the value of the first argument,
4209 then the expression SHALL result in "Indeterminate". The order of evaluation SHALL be:
4210 first evaluate the integer value, then evaluate each subsequent argument. The evaluation
4211 SHALL stop and return "True" if the specified number of arguments evaluate to "True". The
4212 evaluation of arguments SHALL stop if it is determined that evaluating the remaining
4213 arguments will not satisfy the requirement.

4214 • urn:oasis:names:tc:xacml:1.0:function:not

4215 This function SHALL take one argument of data-type
4216 "<http://www.w3.org/2001/XMLSchema#boolean>". If the argument evaluates to "True", then
4217 the result of the expression SHALL be "False". If the argument evaluates to "False", then
4218 the result of the expression SHALL be "True".

4219 Note: When evaluating and, or, or n-of, it MAY NOT be necessary to attempt a full evaluation of
4220 each argument in order to determine whether the evaluation of the argument would result in
4221 "Indeterminate". Analysis of the argument regarding the availability of its attributes, or other
4222 analysis regarding errors, such as "divide-by-zero", may render the argument error free. Such
4223 arguments occurring in the expression in a position after the evaluation is stated to stop need not
4224 be processed.

4225 **A.3.6 Numeric comparison functions**

4226 These functions form a minimal set for comparing two numbers, yielding a Boolean result. They
4227 SHALL comply with the rules governed by IEEE 754 [IEEE 754].

4228 • urn:oasis:names:tc:xacml:1.0:function:integer-greater-than

4229 • urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal

4230 • urn:oasis:names:tc:xacml:1.0:function:integer-less-than

4231 • urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal

4232 • urn:oasis:names:tc:xacml:1.0:function:double-greater-than

4233 • urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal

4234 • urn:oasis:names:tc:xacml:1.0:function:double-less-than

- 4235 • urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal

4236 **A.3.7 Date and time arithmetic functions**

4237 These functions perform arithmetic operations with date and time.

- 4238 • urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration

4239 This function SHALL take two arguments, the first SHALL be of data-type
4240 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be of data-type
4241 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration". It SHALL
4242 return a result of "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL
4243 return the value by adding the second argument to the first argument according to the
4244 specification of adding durations to date and time [XS Appendix E].

- 4245 • urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration

4246 This function SHALL take two arguments, the first SHALL be a
4247 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a
4248 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It
4249 SHALL return a result of "http://www.w3.org/2001/XMLSchema#dateTime". This function
4250 SHALL return the value by adding the second argument to the first argument according to
4251 the specification of adding durations to date and time [XS Appendix E].

- 4252 • urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-dayTimeDuration

4253 This function SHALL take two arguments, the first SHALL be a
4254 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a
4255 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration". It SHALL
4256 return a result of "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument
4257 is a positive duration, then this function SHALL return the value by adding the
4258 corresponding negative duration, as per the specification [XS Appendix E]. If the second
4259 argument is a negative duration, then the result SHALL be as if the function
4260 "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration" had been applied
4261 to the corresponding positive duration.

- 4262 • urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-yearMonthDuration

4263 This function SHALL take two arguments, the first SHALL be a
4264 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a
4265 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It
4266 SHALL return a result of "http://www.w3.org/2001/XMLSchema#dateTime". If the second
4267 argument is a positive duration, then this function SHALL return the value by adding the
4268 corresponding negative duration, as per the specification [XS Appendix E]. If the second
4269 argument is a negative duration, then the result SHALL be as if the function
4270 "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration" had been
4271 applied to the corresponding positive duration.

- 4272 • urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration

4273 This function SHALL take two arguments, the first SHALL be a
4274 "http://www.w3.org/2001/XMLSchema#date" and the second SHALL be a
4275 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It
4276 SHALL return a result of "http://www.w3.org/2001/XMLSchema#date". This function
4277 SHALL return the value by adding the second argument to the first argument according to
4278 the specification of adding duration to date [XS Appendix E].

- 4279 • urn:oasis:names:tc:xacml:1.0:function:date-subtract-yearMonthDuration

4280 This function SHALL take two arguments, the first SHALL be a
4281 "http://www.w3.org/2001/XMLSchema#date" and the second SHALL be a
4282 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It
4283 SHALL return a result of "http://www.w3.org/2001/XMLSchema#date". If the second
4284 argument is a positive duration, then this function SHALL return the value by adding the
4285 corresponding negative duration, as per the specification [XS Appendix E]. If the second
4286 argument is a negative duration, then the result SHALL be as if the function
4287 "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" had been applied to
4288 the corresponding positive duration.

4289 **A.3.8 Non-numeric comparison functions**

4290 These functions perform comparison operations on two arguments of non-numerical types.

- 4291 • urn:oasis:names:tc:xacml:1.0:function:string-greater-than

4292 This function SHALL take two arguments of data-type
4293 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4294 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4295 arguments are compared byte by byte and, after an initial prefix of corresponding bytes
4296 from both arguments that are considered equal by
4297 "urn:oasis:names:tc:xacml:1.0:function:integer-equal", the next byte by byte comparison is
4298 such that the byte from the first argument is greater than the byte from the second
4299 argument by the use of the function "urn:oasis:names:tc:xacml:2.0:function:integer-greater-
4300 then". Otherwise, it SHALL return "False".

- 4301 • urn:oasis:names:tc:xacml:1.0:function:string-greater-than-or-equal

4302 This function SHALL take two arguments of data-type
4303 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4304 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return a result as if evaluated
4305 with the logical function "urn:oasis:names:tc:xacml:1.0:function:or" with two arguments
4306 containing the functions "urn:oasis:names:tc:xacml:1.0:function:string-greater-than" and
4307 "urn:oasis:names:tc:xacml:1.0:function:string-equal" containing the original arguments

- 4308 • urn:oasis:names:tc:xacml:1.0:function:string-less-than

4309 This function SHALL take two arguments of data-type
4310 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4311 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4312 arguments are compared byte by byte and, after an initial prefix of corresponding bytes
4313 from both arguments that are considered equal by
4314 "urn:oasis:names:tc:xacml:1.0:function:integer-equal", the next byte by byte comparison is
4315 such that the byte from the first argument is less than the byte from the second argument
4316 by the use of the function "urn:oasis:names:tc:xacml:1.0:function:integer-less-than".
4317 Otherwise, it SHALL return "False".

- 4318 • urn:oasis:names:tc:xacml:1.0:function:string-less-than-or-equal

4319 This function SHALL take two arguments of data-type
4320 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4321 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return a result as if evaluated
4322 with the function "urn:oasis:names:tc:xacml:1.0:function:or" with two arguments containing
4323 the functions "urn:oasis:names:tc:xacml:1.0:function:string-less-than" and
4324 "urn:oasis:names:tc:xacml:1.0:function:string-equal" containing the original arguments.

- 4325 • urn:oasis:names:tc:xacml:1.0:function:time-greater-than

4326 This function SHALL take two arguments of data-type
4327 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
4328 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4329 first argument is greater than the second argument according to the order relation specified
4330 for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8]. Otherwise, it SHALL
4331 return "False". Note: it is illegal to compare a time that includes a time-zone value with one
4332 that does not. In such cases, the time-in-range function should be used.

- 4333 • urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal

4334 This function SHALL take two arguments of data-type
4335 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
4336 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4337 first argument is greater than or equal to the second argument according to the order
4338 relation specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].
4339 Otherwise, it SHALL return "False". Note: it is illegal to compare a time that includes a
4340 time-zone value with one that does not. In such cases, the time-in-range function should
4341 be used.

- 4342 • urn:oasis:names:tc:xacml:1.0:function:time-less-than

4343 This function SHALL take two arguments of data-type
4344 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
4345 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4346 first argument is less than the second argument according to the order relation specified for
4347 "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8]. Otherwise, it SHALL
4348 return "False". Note: it is illegal to compare a time that includes a time-zone value with one
4349 that does not. In such cases, the time-in-range function should be used.

- 4350 • urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal

4351 This function SHALL take two arguments of data-type
4352 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
4353 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4354 first argument is less than or equal to the second argument according to the order relation
4355 specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8]. Otherwise, it
4356 SHALL return "False". Note: it is illegal to compare a time that includes a time-zone value
4357 with one that does not. In such cases, the time-in-range function should be used.

- 4358 • urn:oasis:names:tc:xacml:1.0:function:time-in-range

4359 This function SHALL take three arguments of data-type
4360 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
4361 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
4362 argument falls in the range defined inclusively by the second and third arguments.
4363 Otherwise, it SHALL return "False". Regardless of its value, the third argument SHALL be
4364 interpreted as a time that is equal to, or later than by less than twenty-four hours, the
4365 second argument. If no time zone is provided for the first argument, it SHALL use the
4366 default time zone at the context handler. If no time zone is provided for the second or third
4367 arguments, then they SHALL use the time zone from the first argument.

- 4368 • urn:oasis:names:tc:xacml:1.0:function:date-time-greater-than

4369 This function SHALL take two arguments of data-type
4370 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
4371 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4372 first argument is greater than the second argument according to the order relation specified
4373 for "http://www.w3.org/2001/XMLSchema#dateTime" by [XF Section 3.2.7]. Otherwise, it

- 4374 SHALL return "False". Note: if a dateTime value does not include a time-zone value, then
4375 an implicit time-zone value SHALL be assigned, as described in [XF].
- 4376 • urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal
- 4377 This function SHALL take two arguments of data-type
4378 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
4379 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4380 first argument is greater than or equal to the second argument according to the order
4381 relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" by [XF Section
4382 3.2.7]. Otherwise, it SHALL return "False". Note: if a dateTime value does not include a
4383 time-zone value, then an implicit time-zone value SHALL be assigned, as described in
4384 [XF].
- 4385 • urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than
- 4386 This function SHALL take two arguments of data-type
4387 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
4388 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4389 first argument is less than the second argument according to the order relation specified for
4390 "http://www.w3.org/2001/XMLSchema#dateTime" by [XF Section 3.2.7]. Otherwise, it
4391 SHALL return "False". Note: if a dateTime value does not include a time-zone value, then
4392 an implicit time-zone value SHALL be assigned, as described in [XF].
- 4393 • urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal
- 4394 This function SHALL take two arguments of data-type
4395 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
4396 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4397 first argument is less than or equal to the second argument according to the order relation
4398 specified for "http://www.w3.org/2001/XMLSchema#dateTime" by [XF Section 3.2.7].
4399 Otherwise, it SHALL return "False". Note: if a dateTime value does not include a time-zone
4400 value, then an implicit time-zone value SHALL be assigned, as described in [XF].
- 4401 • urn:oasis:names:tc:xacml:1.0:function:date-greater-than
- 4402 This function SHALL take two arguments of data-type
4403 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4404 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4405 first argument is greater than the second argument according to the order relation specified
4406 for "http://www.w3.org/2001/XMLSchema#date" by [XF Section 3.2.9]. Otherwise, it SHALL
4407 return "False". Note: if a date value does not include a time-zone value, then an implicit
4408 time-zone value SHALL be assigned, as described in [XF].
- 4409 • urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal
- 4410 This function SHALL take two arguments of data-type
4411 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4412 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4413 first argument is greater than or equal to the second argument according to the order
4414 relation specified for "http://www.w3.org/2001/XMLSchema#date" by [XF Section 3.2.9].
4415 Otherwise, it SHALL return "False". Note: if a date value does not include a time-zone
4416 value, then an implicit time-zone value SHALL be assigned, as described in [XF].
- 4417 • urn:oasis:names:tc:xacml:1.0:function:date-less-than
- 4418 This function SHALL take two arguments of data-type
4419 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4420 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the

4421 first argument is less than the second argument according to the order relation specified for
4422 "http://www.w3.org/2001/XMLSchema#date" by [XF Section 3.2.9]. Otherwise, it SHALL
4423 return "False". Note: if a date value does not include a time-zone value, then an implicit
4424 time-zone value SHALL be assigned, as described in [XF].

- 4425 • urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal

4426 This function SHALL take two arguments of data-type
4427 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4428 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the
4429 first argument is less than or equal to the second argument according to the order relation
4430 specified for "http://www.w3.org/2001/XMLSchema#date" by [XF Section 3.2.9]. Otherwise,
4431 it SHALL return "False". Note: if a date value does not include a time-zone value, then an
4432 implicit time-zone value SHALL be assigned, as described in [XF].

4433 A.3.9 String functions

4434 The following functions operate on strings and URIs.

- 4435 • urn:oasis:names:tc:xacml:2.0:function:string-concatenate

4436

4437 This function SHALL take two or more arguments of data-type
4438 "http://www.w3.org/2001/XMLSchema#string" and SHALL return a
4439 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the concatenation, in
4440 order, of the arguments.

- 4441 • urn:oasis:names:tc:xacml:2.0:function:url-string-concatenate

4442 This function SHALL take one argument of data-type
4443 "http://www.w3.org/2001/XMLSchema#anyURI" and one or more arguments of type
4444 "http://www.w3.org/2001/XMLSchema#string", and SHALL return a
4445 "http://www.w3.org/2001/XMLSchema#anyURI". The result SHALL be the URI constructed
4446 by appending, in order, the "string" arguments to the "anyURI" argument.

4447 A.3.10 Bag functions

4448 These functions operate on a **bag** of 'type' values, where *type* is one of the primitive data-types.
4449 Some additional conditions defined for each function below SHALL cause the expression to
4450 evaluate to "Indeterminate".

- 4451 • urn:oasis:names:tc:xacml:1.0:function:type-one-and-only

4452 This function SHALL take a **bag** of 'type' values as an argument and SHALL return a value
4453 of '-type'. It SHALL return the only value in the **bag**. If the **bag** does not have one and only
4454 one value, then the expression SHALL evaluate to "Indeterminate".

- 4455 • urn:oasis:names:tc:xacml:1.0:function:type-bag-size

4456 This function SHALL take a **bag** of 'type' values as an argument and SHALL return an
4457 "http://www.w3.org/2001/XMLSchema#integer" indicating the number of values in the **bag**.

- 4458 • urn:oasis:names:tc:xacml:1.0:function:type-is-in

4459 This function SHALL take an argument of 'type' as the first argument and a **bag** of *type*
4460 values as the second argument and SHALL return an
4461 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL evaluate to "True" if

4462 and only if the first argument matches by the "urn:oasis:names:tc:xacml:x.x:function:type-
4463 equal" any value in the **bag**. Otherwise, it SHALL return "False".

- 4464 • urn:oasis:names:tc:xacml:1.0:function:type-bag

4465 This function SHALL take any number of arguments of 'type' and return a **bag** of 'type'
4466 values containing the values of the arguments. An application of this function to zero
4467 arguments SHALL produce an empty **bag** of the specified data-type.

4468 **A.3.11 Set functions**

4469 These functions operate on **bags** mimicking sets by eliminating duplicate elements from a **bag**.

- 4470 • urn:oasis:names:tc:xacml:1.0:function:type-intersection

4471 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL
4472 return a **bag** of 'type' values such that it contains only elements that are common between
4473 the two **bags**, which is determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal".
4474 No duplicates, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal",
4475 SHALL exist in the result.

- 4476 • urn:oasis:names:tc:xacml:1.0:function:type-at-least-one-member-of

4477 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL
4478 return a "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL evaluate to
4479 "True" if and only if at least one element of the first argument is contained in the second
4480 argument as determined by "urn:oasis:names:tc:xacml:x.x:function:type-is-in".

- 4481 • urn:oasis:names:tc:xacml:1.0:function:type-union

4482 This function SHALL take two arguments that are both a **bag** of 'type' values. The
4483 expression SHALL return a **bag** of 'type' such that it contains all elements of both **bags**.
4484 No duplicates, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal",
4485 SHALL exist in the result.

- 4486 • urn:oasis:names:tc:xacml:1.0:function:type-subset

4487 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL
4488 return a "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and
4489 only if the first argument is a subset of the second argument. Each argument SHALL be
4490 considered to have had its duplicates removed, as determined by
4491 "urn:oasis:names:tc:xacml:x.x:function:type-equal", before the subset calculation.

- 4492 • urn:oasis:names:tc:xacml:1.0:function:type-set-equals

4493 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL
4494 return a "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return the result of
4495 applying "urn:oasis:names:tc:xacml:1.0:function:and" to the application of
4496 "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the first and second arguments and
4497 the application of "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the second and
4498 first arguments.

4499 **A.3.12 Higher-order bag functions**

4500 This section describes functions in XACML that perform operations on **bags** such that functions
4501 may be applied to the **bags** in general.

4502 In this section, a general-purpose functional language called Haskell [**Haskell**] is used to formally
4503 specify the semantics of these functions. Although the English description is adequate, a formal
4504 specification of the semantics is helpful.

4505 For a quick summary, in the following Haskell notation, a function definition takes the form of
4506 clauses that are applied to patterns of structures, namely lists. The symbol "[]" denotes the empty
4507 list, whereas the expression "(x:xs)" matches against an argument of a non-empty list of which "x"
4508 represents the first element of the list, and "xs" is the rest of the list, which may be an empty list.
4509 We use the Haskell notion of a list, which is an ordered collection of elements, to model the XACML
4510 **bags** of values.

4511 A simple Haskell definition of a familiar function "urn:oasis:names:tc:xacml:1.0:function:and" that
4512 takes a list of values of type Boolean is defined as follows:

4513 and:: [Bool] -> Bool

4514 and [] = "True"

4515 and (x:xs) = x && (and xs)

4516 The first definition line denoted by a "::" formally describes the data-type of the function, which takes
4517 a list of Booleans, denoted by "[Bool]", and returns a Boolean, denoted by "Bool". The second
4518 definition line is a clause that states that the function "and" applied to the empty list is "True". The
4519 third definition line is a clause that states that for a non-empty list, such that the first element is "x",
4520 which is a value of data-type Bool, the function "and" applied to x SHALL be combined with, using
4521 the logical conjunction function, which is denoted by the infix symbol "&&", the result of recursively
4522 applying the function "and" to the rest of the list. Of course, an application of the "and" function is
4523 "True" if and only if the list to which it is applied is empty or every element of the list is "True". For
4524 example, the evaluation of the following Haskell expressions,

4525 (and []), (and ["True"]), (and ["True","True"]), (and ["True","True","False"])

4526 evaluate to "True", "True", "True", and "False", respectively.

4527 • urn:oasis:names:tc:xacml:1.0:function:any-of

4528 This function applies a Boolean function between a specific primitive value and a **bag** of
4529 values, and SHALL return "True" if and only if the predicate is "True" for at least one
4530 element of the **bag**.

4531 This function SHALL take three arguments. The first argument SHALL be an
4532 <xacml:Function> element that names a Boolean function that takes two arguments of
4533 primitive types. The second argument SHALL be a value of a primitive data-type. The third
4534 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated
4535 as if the function named in the <xacml:Function> argument were applied to the second
4536 argument and each element of the third argument (the **bag**) and the results are combined
4537 with "urn:oasis:names:tc:xacml:1.0:function:or".

4538 In Haskell, the semantics of this operation are as follows:

4539 any_of :: (a -> b -> Bool) -> a -> [b] -> Bool

4540 any_of f a [] = "False"

4541 any_of f a (x:xs) = (f a x) || (any_of f a xs)

4542 In the above notation, "f" is the function to be applied, "a" is the primitive value, and "(x:xs)"
4543 represents the first element of the list as "x" and the rest of the list as "xs".

4544 For example, the following expression SHALL return "True":

```

4545 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
4546   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
4547   <AttributeValue
4548     DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4549   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4550     <AttributeValue
4551       DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4552     <AttributeValue
4553       DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4554     <AttributeValue
4555       DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4556     <AttributeValue
4557       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4558   </Apply>
4559 </Apply>

```

4560 This expression is "True" because the first argument is equal to at least one of the
4561 elements of the **bag**, according to the function.

4562 • urn:oasis:names:tc:xacml:1.0:function:all-of

4563 This function applies a Boolean function between a specific primitive value and a **bag** of
4564 values, and returns "True" if and only if the predicate is "True" for every element of the **bag**.

4565 This function SHALL take three arguments. The first argument SHALL be an
4566 <xacml:Function> element that names a Boolean function that takes two arguments of
4567 primitive types. The second argument SHALL be a value of a primitive data-type. The third
4568 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated
4569 as if the function named in the <xacml:Function> argument were applied to the second
4570 argument and each element of the third argument (the **bag**) and the results were combined
4571 using "urn:oasis:names:tc:xacml:1.0:function:and".

4572 In Haskell, the semantics of this operation are as follows:

```

4573     all_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
4574     all_of f a [] = "False"
4575     all_of f a (x:xs) = (f a x) && (all_of f a xs)

```

4576 In the above notation, "f" is the function to be applied, "a" is the primitive value, and "(x:xs)"
4577 represents the first element of the list as "x" and the rest of the list as "xs".

4578 For example, the following expression SHALL evaluate to "True":

```

4579 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">
4580   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater"/>
4581   <AttributeValue
4582     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4583   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4584     <AttributeValue
4585       DataType="http://www.w3.org/2001/XMLSchema#integer">9</AttributeValue>
4586     <AttributeValue
4587       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4588     <AttributeValue
4589       DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4590     <AttributeValue
4591       DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4592   </Apply>
4593 </Apply>

```

4594 This expression is "True" because the first argument (10) is greater than *all* of the elements
4595 of the **bag** (9,3,4 and 2).

4596 • urn:oasis:names:tc:xacml:1.0:function:any-of-any

4597 This function applies a Boolean function between each element of a **bag** of values and
4598 each element of another **bag** of values, and returns "True" if and only if the predicate is
4599 "True" for at least one comparison.

4600 This function SHALL take three arguments. The first argument SHALL be an
4601 `<xacml:Function>` element that names a Boolean function that takes two arguments of
4602 primitive types. The second argument SHALL be a **bag** of a primitive data-type. The third
4603 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated
4604 as if the function named in the `<xacml:Function>` argument were applied between
4605 every element of the second argument and every element of the third argument and the
4606 results were combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics
4607 are that the result of the expression SHALL be "True" if and only if the applied predicate is
4608 "True" for *at least one* comparison of elements from the two **bags**.

4609 In Haskell, taking advantage of the "any_of" function defined above, the semantics of the
4610 "any_of_any" function are as follows:

```
4611 any_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ]-> Bool  
4612 any_of_any f [] ys = "False"  
4613 any_of_any f (x:xs) ys = (any_of f x ys) || (any_of_any f xs ys)
```

4614 In the above notation, "f" is the function to be applied and "(x:xs)" represents the first
4615 element of the list as "x" and the rest of the list as "xs".

4616 For example, the following expression SHALL evaluate to "True":

```
4617 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">  
4618   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>  
4619   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4620     <AttributeValue  
4621       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>  
4622     <AttributeValue  
4623       DataType="http://www.w3.org/2001/XMLSchema#string">Mary</AttributeValue>  
4624   </Apply>  
4625   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4626     <AttributeValue  
4627       DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>  
4628     <AttributeValue  
4629       DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>  
4630     <AttributeValue  
4631       DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>  
4632     <AttributeValue  
4633       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>  
4634   </Apply>  
4635 </Apply>
```

4636 This expression is "True" because at least one of the elements of the first **bag**, namely
4637 "Ringo", is equal to at least one of the elements of the second **bag**.

4638 • urn:oasis:names:tc:xacml:1.0:function:all-of-any

4639 This function applies a Boolean function between the elements of two **bags**. The
4640 expression SHALL be "True" if and only if the supplied predicate is 'True' between each
4641 element of the first **bag** and any element of the second **bag**.

4642 This function SHALL take three arguments. The first argument SHALL be an
4643 `<xacml:Function>` element that names a Boolean function that takes two arguments of
4644 primitive types. The second argument SHALL be a **bag** of a primitive data-type. The third
4645 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated
4646 as if the "urn:oasis:names:tc:xacml:1.0:function:any-of" function had been applied to each

4647 value of the first **bag** and the whole of the second **bag** using the supplied xacml:Function,
4648 and the results were then combined using “urn:oasis:names:tc:xacml:1.0:function:and”.

4649 In Haskell, taking advantage of the “any_of” function defined in Haskell above, the
4650 semantics of the “all_of_any” function are as follows:

```
4651 all_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4652 all_of_any f [] ys = "False"
4653 all_of_any f (x:xs) ys = (any_of f x ys) && (all_of_any f xs ys)
```

4654 In the above notation, “f” is the function to be applied and “(x:xs)” represents the first
4655 element of the list as “x” and the rest of the list as “xs”.

4656 For example, the following expression SHALL evaluate to "True":

```
4657 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-any">
4658   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater"/>
4659   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4660     <AttributeValue
4661       DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4662     <AttributeValue
4663       DataType="http://www.w3.org/2001/XMLSchema#integer">20</AttributeValue>
4664   </Apply>
4665   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4666     <AttributeValue
4667       DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4668     <AttributeValue
4669       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4670     <AttributeValue
4671       DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4672     <AttributeValue
4673       DataType="http://www.w3.org/2001/XMLSchema#integer">19</AttributeValue>
4674   </Apply>
4675 </Apply>
```

4676 This expression is “True” because each of the elements of the first **bag** is greater than at
4677 least one of the elements of the second **bag**.

4678 • urn:oasis:names:tc:xacml:1.0:function:any-of-all

4679 This function applies a Boolean function between the elements of two **bags**. The
4680 expression SHALL be “True” if and only if the supplied predicate is “True” between each
4681 element of the second **bag** and any element of the first **bag**.

4682 This function SHALL take three arguments. The first argument SHALL be an
4683 <xacml:Function> element that names a Boolean function that takes two arguments of
4684 primitive types. The second argument SHALL be a **bag** of a primitive data-type. The third
4685 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated
4686 as if the “urn:oasis:names:tc:xacml:1.0:function:any-of” function had been applied to each
4687 value of the second **bag** and the whole of the first **bag** using the supplied xacml:Function,
4688 and the results were then combined using “urn:oasis:names:tc:xacml:1.0:function:and”.

4689 In Haskell, taking advantage of the “all_of” function defined in Haskell above, the semantics
4690 of the “any_of_all” function are as follows:

```
4691 any_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4692 any_of_all f [] ys = "False"
4693 any_of_all f (x:xs) ys = (all_of f x ys) || (any_of_all f xs ys)
```

4694 In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first
4695 element of the list as “x” and the rest of the list as “xs”.

4696 For example, the following expression SHALL evaluate to "True":

```

4697 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-all">
4698   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater"/>
4699   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4700     <AttributeValue
4701       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4702     <AttributeValue
4703       DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4704     </Apply>
4705   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4706     <AttributeValue
4707       DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4708     <AttributeValue
4709       DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4710     <AttributeValue
4711       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4712     <AttributeValue
4713       DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4714   </Apply>
4715 </Apply>

```

4716 This expression is "True" because, for all of the values in the second **bag**, there is a value
4717 in the first **bag** that is greater.

4718 • urn:oasis:names:tc:xacml:1.0:function:all-of-all

4719 This function applies a Boolean function between the elements of two **bags**. The
4720 expression SHALL be "True" if and only if the supplied predicate is "True" between each
4721 and every element of the first **bag** collectively against all the elements of the second **bag**.

4722 This function SHALL take three arguments. The first argument SHALL be an
4723 <xacml:Function> element that names a Boolean function that takes two arguments of
4724 primitive types. The second argument SHALL be a **bag** of a primitive data-type. The third
4725 argument SHALL be a **bag** of a primitive data-type. The expression is evaluated as if the
4726 function named in the <xacml:Function> element were applied between every element
4727 of the second argument and every element of the third argument and the results were
4728 combined using "urn:oasis:names:tc:xacml:1.0:function:and". The semantics are that the
4729 result of the expression is "True" if and only if the applied predicate is "True" for *all*
4730 elements of the first **bag** compared to *all* the elements of the second **bag**.

4731 In Haskell, taking advantage of the "all_of" function defined in Haskell above, the semantics
4732 of the "all_of_all" function is as follows:

```

4733               all_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4734               all_of_all f [] ys = "False"
4735               all_of_all f (x:xs) ys = (all_of f x ys) && (all_of_all f xs ys)

```

4736 In the above notation, "f" is the function to be applied and "(x:xs)" represents the first
4737 element of the list as "x" and the rest of the list as "xs".

4738 For example, the following expression SHALL evaluate to "True":

```

4739 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-all">
4740   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater"/>
4741   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4742     <AttributeValue
4743     DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>
4744     <AttributeValue
4745     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4746   </Apply>
4747   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4748     <AttributeValue
4749     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4750     <AttributeValue
4751     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4752     <AttributeValue
4753     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4754     <AttributeValue
4755     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4756   </Apply>
4757 </Apply>

```

4758 This expression is "True" because all elements of the first **bag**, "5" and "6", are each
4759 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4760 • urn:oasis:names:tc:xacml:1.0:function:map

4761 This function converts a **bag** of values to another **bag** of values.

4762 This function SHALL take two arguments. The first function SHALL be an
4763 <xacml:Function> element naming a function that takes a single argument of a primitive
4764 data-type and returns a value of a primitive data-type. The second argument SHALL be a
4765 **bag** of a primitive data-type. The expression SHALL be evaluated as if the function named
4766 in the <xacml:Function> element were applied to each element in the **bag** resulting in a
4767 **bag** of the converted value. The result SHALL be a **bag** of the primitive data-type that is
4768 returned by the function named in the <xacml:Function> element.

4769 In Haskell, this function is defined as follows:

```

4770     map:: (a -> b) -> [a] -> [b]
4771     map f [] = []
4772     map f (x:xs) = (f x) : (map f xs)

```

4773 In the above notation, "f" is the function to be applied and "(x:xs)" represents the first
4774 element of the list as "x" and the rest of the list as "xs".

4775 For example, the following expression,

```

4776 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:map">
4777   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-normalize-
4778   to-lower-case">
4779   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4780     <AttributeValue
4781     DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>
4782     <AttributeValue
4783     DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>
4784   </Apply>
4785 </Apply>

```

4786 evaluates to a **bag** containing "hello" and "world!".

4787

A.3.13 Regular-expression-based functions

4788 These functions operate on various types using regular expressions and evaluate to
4789 "http://www.w3.org/2001/XMLSchema#boolean".

- 4790 • urn:oasis:names:tc:xacml:1.0:function:regexp-string-match

4791 This function decides a regular expression match. It SHALL take two arguments of
4792 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4793 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4794 expression and the second argument SHALL be a general string. The function
4795 specification SHALL be that of the "xf:matches" function with the arguments reversed [XF
4796 Section 6.3.15].

- 4797 • urn:oasis:names:tc:xacml:1.0:function:regexp-uri-match

4798 This function decides a regular expression match. It SHALL take two arguments; the first is
4799 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4800 "http://www.w3.org/2001/XMLSchema#anyURI". It SHALL return an
4801 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4802 expression and the second argument SHALL be a URI. The function SHALL convert the
4803 second argument to type "http://www.w3.org/2001/XMLSchema#string", then apply
4804 "urn:oasis:names:tc:xacml:1.0:function:regexp-string-match".

- 4805 • urn:oasis:names:tc:xacml:1.0:function:regexp-ipAddress-match

4806 This function decides a regular expression match. It SHALL take two arguments; the first is
4807 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4808 "urn:oasis:names:tc:xacml:1.0:data-type:ipAddress". It SHALL return an
4809 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4810 expression and the second argument SHALL be an IPv4 or IPv6 address. The function
4811 SHALL convert the second argument to type "http://www.w3.org/2001/XMLSchema#string",
4812 then apply "urn:oasis:names:tc:xacml:1.0:function:regexp-string-match".

- 4813 • urn:oasis:names:tc:xacml:1.0:function:regexp-dnsName-match

4814 This function decides a regular expression match. It SHALL take two arguments; the first is
4815 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4816 "urn:oasis:names:tc:xacml:1.0:data-type:dnsName". It SHALL return an
4817 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4818 expression and the second argument SHALL be a DNS name. The function SHALL
4819 convert the second argument to type "http://www.w3.org/2001/XMLSchema#string", then
4820 apply "urn:oasis:names:tc:xacml:1.0:function:regexp-string-match".

- 4821 • urn:oasis:names:tc:xacml:1.0:function:regexp-rfc822Name-match

4822 This function decides a regular expression match. It SHALL take two arguments; the first is
4823 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4824 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name". It SHALL return an
4825 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4826 expression and the second argument SHALL be an RFC 822 name. The function SHALL
4827 convert the second argument to type "http://www.w3.org/2001/XMLSchema#string", then
4828 apply "urn:oasis:names:tc:xacml:1.0:function:regexp-string-match".

- 4829 • urn:oasis:names:tc:xacml:1.0:function:regexp-x500Name-match

4830 This function decides a regular expression match. It SHALL take two arguments; the first is
4831 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4832 "urn:oasis:names:tc:xacml:1.0:data-type:x500Name". It SHALL return an

4833 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4834 expression and the second argument SHALL be an X.500 directory name. The function
4835 SHALL convert the second argument to type "http://www.w3.org/2001/XMLSchema#string",
4836 then apply "urn:oasis:names:tc:xacml:1.0:function:regexp-string-match".

4837 **A.3.14 Special match functions**

4838 These functions operate on various types and evaluate to
4839 "http://www.w3.org/2001/XMLSchema#boolean" based on the specified standard matching
4840 algorithm.

- 4841 • urn:oasis:names:tc:xacml:1.0:function:x500Name-match

4842 This function shall take two arguments of "urn:oasis:names:tc:xacml:2.0:data-
4843 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It
4844 shall return "True" if and only if the first argument matches some terminal sequence of
4845 RDNs from the second argument when compared using x500Name-equal.

- 4846 • urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match

4847 This function SHALL take two arguments, the first is of data-type
4848 "http://www.w3.org/2001/XMLSchema#string" and the second is of data-type
4849 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an
4850 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if
4851 the first argument matches the second argument according to the following specification.

4852 An RFC822 name consists of a local-part followed by "@" followed by a domain-part. The
4853 local-part is case-sensitive, while the domain-part (which is usually a DNS name) is not
4854 case-sensitive.⁴

4855 The second argument contains a complete rfc822Name. The first argument is a complete
4856 or partial rfc822Name used to select appropriate values in the second argument as follows.

4857 In order to match a particular address in the second argument, the first argument must
4858 specify the complete mail address to be matched. For example, if the first argument is
4859 "Anderson@sun.com", this matches a value in the second argument of
4860 "Anderson@sun.com" and "Anderson@SUN.COM", but not "Anne.Anderson@sun.com",
4861 "anderson@sun.com" or "Anderson@east.sun.com".

4862 In order to match any address at a particular domain in the second argument, the first
4863 argument must specify only a domain name (usually a DNS name). For example, if the first
4864 argument is "sun.com", this matches a value in the first argument of "Anderson@sun.com"
4865 or "Baxter@SUN.COM", but not "Anderson@east.sun.com".

4866 In order to match any address in a particular domain in the second argument, the first
4867 argument must specify the desired domain-part with a leading ".". For example, if the first
4868 argument is ".east.sun.com", this matches a value in the second argument of
4869 "Anderson@east.sun.com" and "anne.anderson@ISRG.EAST.SUN.COM" but not
4870 "Anderson@sun.com".

4 According to IETF RFC822 and its successor specifications [RFC2821], case is significant in the *local-part*. Many mail systems, as well as the IETF PKIX specification, treat the *local-part* as case-insensitive. This anomaly is considered an error by mail-system designers and is not encouraged. For this reason, rfc822Name-match treats *local-part* as case sensitive.

4871

A.3.15 XPath-based functions

4872 This section specifies functions that take XPath expressions for arguments. An XPath expression
4873 evaluates to a *node-set*, which is a set of XML nodes that match the expression. A node or node-
4874 set is not in the formal data-type system of XACML. All comparison or other operations on node-
4875 sets are performed in isolation of the particular function specified. That is, the XPath expressions in
4876 these functions are restricted to the XACML request **context**. The `<xacml-context:Request>`
4877 element is the context node for every XPath expression. The following functions are defined:

- 4878 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-count

4879 This function SHALL take an "http://www.w3.org/2001/XMLSchema#string" as an
4880 argument, which SHALL be interpreted as an XPath expression, and evaluates to an
4881 "http://www.w3.org/2001/XMLSchema#integer". The value returned from the function
4882 SHALL be the count of the nodes within the node-set that match the given XPath
4883 expression.

- 4884 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal

4885 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,
4886 which SHALL be interpreted as XPath expressions, and SHALL return an
4887 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if any
4888 of the XML nodes in the node-set matched by the first argument equals, according to the
4889 "op:node-equal" function [XF Section 13.1.6], any of the XML nodes in the node-set
4890 matched by the second argument.

- 4891 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-match

4892 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,
4893 which SHALL be interpreted as XPath expressions and SHALL return an
4894 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if
4895 one of the following two conditions is satisfied: (1) Any of the XML nodes in the node-set
4896 matched by the first argument is equal, according to "op:node-equal" [XF Section 13.1.6],
4897 to any of the XML nodes in the node-set matched by the second argument; (2) any attribute
4898 and element node below any of the XML nodes in the node-set matched by the first
4899 argument is equal, according to "op:node-equal" [XF Section 13.1.6], to any of the XML
4900 nodes in the node-set matched by the second argument.

4901 NOTE: The first condition is equivalent to "xpath-node-equal", and guarantees that "xpath-node-
4902 equal" is a special case of "xpath-node-match".

4903 A.3.16 Extension functions and primitive types

4904 Functions and primitive types are specified by string identifiers allowing for the introduction of
4905 functions in addition to those specified by XACML. This approach allows one to extend the XACML
4906 module with special functions and special primitive data-types.

4907 In order to preserve the integrity of the XACML evaluation strategy, the result of an extension
4908 function SHALL depend only on the values of its arguments. Global and hidden parameters SHALL
4909 NOT affect the evaluation of an expression. Functions SHALL NOT have side effects, as
4910 evaluation order cannot be guaranteed in a standard way.

4911 Appendix B. XACML identifiers (normative)

4912 This section defines standard identifiers for commonly used entities.

4913 B.1. XACML namespaces

4914 There are currently two defined XACML namespaces.

4915 Policies are defined using this identifier.

4916 `urn:oasis:names:tc:xacml:2.0:policy:schema:cd`

4917 Request and response **contexts** are defined using this identifier.

4918 `urn:oasis:names:tc:xacml:2.0:context:schema:cd`

4919 B.2. Access subject categories

4920 This identifier indicates the system entity that initiated the **access** request. That is, the initial entity
4921 in a request chain. If **subject** category is not specified, this is the default value.

4922 `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

4923 This identifier indicates the system entity that will receive the results of the request (used when it is
4924 distinct from the access-subject).

4925 `urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject`

4926 This identifier indicates a system entity through which the **access** request was passed. There may
4927 be more than one. No means is provided to specify the order in which they passed the message.

4928 `urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject`

4929 This identifier indicates a system entity associated with a local or remote codebase that generated
4930 the request. Corresponding **subject attributes** might include the URL from which it was loaded
4931 and/or the identity of the code-signer. There may be more than one. No means is provided to
4932 specify the order in which they processed the request.

4933 `urn:oasis:names:tc:xacml:1.0:subject-category:codebase`

4934 This identifier indicates a system entity associated with the computer that initiated the **access**
4935 request. An example would be an IPsec identity.

4936 `urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine`

4937 B.3. Data-types

4938 The following identifiers indicate data-types that are defined in Section A.2.

4939 `urn:oasis:names:tc:xacml:1.0:data-type:x500Name.`

4940 `urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name`

4941 `urn:oasis:names:tc:xacml:1.0:data-type:ipAddress`

4942 `urn:oasis:names:tc:xacml:1.0:data-type:dnsName`

4943 The following data-type identifiers are defined by XML Schema [XS].

4944 `http://www.w3.org/2001/XMLSchema#string`

4945 `http://www.w3.org/2001/XMLSchema#boolean`

4946 `http://www.w3.org/2001/XMLSchema#integer`

4947 <http://www.w3.org/2001/XMLSchema#double>
4948 <http://www.w3.org/2001/XMLSchema#time>
4949 <http://www.w3.org/2001/XMLSchema#date>
4950 <http://www.w3.org/2001/XMLSchema#dateTime>
4951 <http://www.w3.org/2001/XMLSchema#anyURI>
4952 <http://www.w3.org/2001/XMLSchema#hexBinary>
4953 <http://www.w3.org/2001/XMLSchema#base64Binary>
4954 The following data-type identifiers correspond to the `dayTimeDuration` and `yearMonthDuration`
4955 data-types defined in [XF Sections 8.2.2 and 8.2.1, respectively].
4956 <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration>
4957 <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration>

4958 B.4. Subject attributes

4959 These identifiers indicate *attributes* of a *subject*. When used, they SHALL appear within a
4960 `<Subject>` element of the request *context*. They SHALL be accessed by means of a
4961 `<SubjectAttributeDesignator>` element, or an `<AttributeSelector>` element that points
4962 into a `<Subject>` element of the request *context*.

4963 At most one of each of these attributes is associated with each subject. Each attribute associated
4964 with authentication included within a single `<Subject>` element relates to the same authentication
4965 event.

4966 This identifier indicates the name of the *subject*. The default format is
4967 “<http://www.w3.org/2001/XMLSchema#string>”. To indicate other formats, use the `DataType`
4968 attributes listed in B.3

4969 `urn:oasis:names:tc:xacml:1.0:subject:subject-id`

4970 This identifier indicates the *subject* category. “access-subject” is the default value.

4971 `urn:oasis:names:tc:xacml:1.0:subject-category`

4972 This identifier indicates the security domain of the *subject*. It identifies the administrator and policy
4973 that manages the name-space in which the *subject* id is administered.

4974 `urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier`

4975 This identifier indicates a public key used to confirm the *subject's* identity.

4976 `urn:oasis:names:tc:xacml:1.0:subject:key-info`

4977 This identifier indicates the time at which the *subject* was authenticated.

4978 `urn:oasis:names:tc:xacml:1.0:subject:authentication-time`

4979 This identifier indicates the method used to authenticate the *subject*.

4980 `urn:oasis:names:tc:xacml:1.0:subject:authn-locality:authentication-method`

4981 This identifier indicates the time at which the *subject* initiated the *access* request, according to the
4982 *PEP*.

4983 `urn:oasis:names:tc:xacml:1.0:subject:request-time`

4984 This identifier indicates the time at which the *subject's* current session began, according to the
4985 *PEP*.

4986 `urn:oasis:names:tc:xacml:1.0:subject:session-start-time`

4987 The following identifiers indicate the location where authentication credentials were activated. They
4988 are intended to support the corresponding entities from the SAML authentication statement
4989 [SAML].

4990 This identifier indicates that the location is expressed as an IP address.

4991 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address
4992 The corresponding attribute SHALL be of data-type "http://www.w3.org/2001/XMLSchema#string".
4993 This identifier indicates that the location is expressed as a DNS name.
4994 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name
4995 The corresponding attribute SHALL be of data-type "http://www.w3.org/2001/XMLSchema#string".
4996 Where a suitable attribute is already defined in LDAP [LDAP-1, LDAP-2], the XACML identifier
4997 SHALL be formed by adding the **attribute** name to the URI of the LDAP specification. For
4998 example, the **attribute** name for the userPassword defined in the RFC 2256 SHALL be:
4999 http://www.ietf.org/rfc/rfc2256.txt#userPassword

5000 B.6. Resource attributes

5001 These identifiers indicate **attributes** of the **resource**. The corresponding **attributes** MAY appear in
5002 the <Resource> element of the request **context** and be accessed by means of a
5003 <ResourceAttributeDesignator> element, or by an <AttributeSelector> element that
5004 points into the <Resource> element of the request **context**.

5005 This **attribute** identifies the **resource** to which access is requested. If an <xacml-
5006 context:ResourceContent> element is provided, then the resource to which access is
5007 requested SHALL be all or a portion of the resource supplied in the <xacml-
5008 context:ResourceContent> element.

5009 urn:oasis:names:tc:xacml:1.0:resource:resource-id

5010 This **attribute** identifies the namespace of the top element of the contents of the <xacml-
5011 context:ResourceContent> element. In the case where the **resource** content is supplied in the
5012 request **context** and the **resource** namespace is defined in the **resource**, the PDP SHALL confirm
5013 that the namespace defined by this **attribute** is the same as that defined in the **resource**. The type
5014 of the corresponding **attribute** SHALL be "http://www.w3.org/2001/XMLSchema#anyURI".

5015 urn:oasis:names:tc:xacml:2.0:resource:target-namespace

5016 B.7. Action attributes

5017 These identifiers indicate **attributes** of the **action** being requested. When used, they SHALL
5018 appear within the <Action> element of the request **context**. They SHALL be accessed by means
5019 of an <ActionAttributeDesignator> element, or an <AttributeSelector> element that
5020 points into the <Action> element of the request **context**.

5021 This **attribute** identifies the **action** for which **access** is requested.

5022 urn:oasis:names:tc:xacml:1.0:action:action-id

5023 Where the **action** is implicit, the value of the action-id **attribute** SHALL be

5024 urn:oasis:names:tc:xacml:1.0:action:implied-action

5025 This **attribute** identifies the namespace in which the action-id **attribute** is defined.

5026 urn:oasis:names:tc:xacml:1.0:action:action-namespace

5027 B.8. Environment attributes

5028 These identifiers indicate *attributes* of the *environment* within which the *decision request* is to be
5029 evaluated. When used in the *decision request*, they SHALL appear in the <Environment>
5030 element of the request *context*. They SHALL be accessed by means of an
5031 <EnvironmentAttributeDesignator> element, or an <AttributeSelector> element that
5032 points into the <Environment> element of the request *context*.

5033 This identifier indicates the current time at the *context handler*. In practice it is the time at which
5034 the request *context* was created. For this reason, if these identifiers appear in multiple places
5035 within a <Policy> or <PolicySet>, then the same value SHALL be assigned to each occurrence
5036 in the evaluation procedure, regardless of how much time elapses between the processing of the
5037 occurrences.

5038 urn:oasis:names:tc:xacml:1.0:environment:current-time

5039 The corresponding *attribute* SHALL be of data-type
5040 "http://www.w3.org/2001/XMLSchema#time".

5041 urn:oasis:names:tc:xacml:1.0:environment:current-date

5042 The corresponding *attribute* SHALL be of data-type
5043 "http://www.w3.org/2001/XMLSchema#date".

5044 urn:oasis:names:tc:xacml:1.0:environment:current-dateTime

5045 The corresponding *attribute* SHALL be of data-type
5046 "http://www.w3.org/2001/XMLSchema#dateTime".

5047 B.9. Status codes

5048 The following status code values are defined.

5049 This identifier indicates success.

5050 urn:oasis:names:tc:xacml:1.0:status:ok

5051 This identifier indicates that all the attributes necessary to make a policy decision were not available
5052 (see Section 6.16).

5053 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

5054 This identifier indicates that some attribute value contained a syntax error, such as a letter in a
5055 numeric field.

5056 urn:oasis:names:tc:xacml:1.0:status:syntax-error

5057 This identifier indicates that an error occurred during policy evaluation. An example would be
5058 division by zero.

5059 urn:oasis:names:tc:xacml:1.0:status:processing-error

5060 B.10. Combining algorithms

5061 The deny-overrides rule-combining algorithm has the following value for the
5062 ruleCombiningAlgId attribute:

5063 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

5064 The deny-overrides policy-combining algorithm has the following value for the
5065 policyCombiningAlgId attribute:
5066 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides
5067 The permit-overrides rule-combining algorithm has the following value for the
5068 ruleCombiningAlgId attribute:
5069 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides
5070 The permit-overrides policy-combining algorithm has the following value for the
5071 policyCombiningAlgId attribute:
5072 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides
5073 The first-applicable rule-combining algorithm has the following value for the
5074 ruleCombiningAlgId attribute:
5075 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable
5076 The first-applicable policy-combining algorithm has the following value for the
5077 policyCombiningAlgId attribute:
5078 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable
5079 The only-one-applicable-policy policy-combining algorithm has the following value for the
5080 policyCombiningAlgId attribute:
5081 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-
5082 applicable
5083 The ordered-deny-overrides rule-combining algorithm has the following value for the
5084 ruleCombiningAlgId attribute:
5085 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-
5086 overrides
5087 The ordered-deny-overrides policy-combining algorithm has the following value for the
5088 policyCombiningAlgId attribute:
5089 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-
5090 overrides
5091 The ordered-permit-overrides rule-combining algorithm has the following value for the
5092 ruleCombiningAlgId attribute:
5093 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-
5094 overrides
5095 The ordered-permit-overrides policy-combining algorithm has the following value for the
5096 policyCombiningAlgId attribute:
5097 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-
5098 overrides

5099 Appendix C. Combining algorithms (normative)

5100 This section contains a description of the *rule-* and *policy-combining algorithms* specified by
5101 XACML.

5102 C.1. Deny-overrides

5103 The following specification defines the "Deny-overrides" *rule-combining algorithm* of a *policy*.

5104 In the entire set of *rules* in the *policy*, if any *rule* evaluates to "Deny", then the result of the
5105 *rule* combination SHALL be "Deny". If any *rule* evaluates to "Permit" and all other *rules*
5106 evaluate to "NotApplicable", then the result of the *rule* combination SHALL be "Permit". In
5107 other words, "Deny" takes precedence, regardless of the result of evaluating any of the
5108 other *rules* in the combination. If all *rules* are found to be "NotApplicable" to the *decision*
5109 *request*, then the *rule* combination SHALL evaluate to "NotApplicable".

5110 If an error occurs while evaluating the *target* or *condition* of a *rule* that contains an *effect*
5111 value of "Deny" then the evaluation SHALL continue to evaluate subsequent *rules*, looking
5112 for a result of "Deny". If no other *rule* evaluates to "Deny", then the combination SHALL
5113 evaluate to "Indeterminate", with the appropriate error status.

5114 If at least one *rule* evaluates to "Permit", all other *rules* that do not have evaluation errors
5115 evaluate to "Permit" or "NotApplicable" and all *rules* that do have evaluation errors contain
5116 *effects* of "Permit", then the result of the combination SHALL be "Permit".

5117 The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

```
5118 Decision denyOverridesRuleCombiningAlgorithm(Rule rule[])
5119 {
5120     Boolean atLeastOneError = false;
5121     Boolean potentialDeny = false;
5122     Boolean atLeastOnePermit = false;
5123     for( i=0 ; i < lengthOf(rules) ; i++ )
5124     {
5125         Decision decision = evaluate(rule[i]);
5126         if (decision == Deny)
5127         {
5128             return Deny;
5129         }
5130         if (decision == Permit)
5131         {
5132             atLeastOnePermit = true;
5133             continue;
5134         }
5135         if (decision == NotApplicable)
5136         {
5137             continue;
5138         }
5139         if (decision == Indeterminate)
5140         {
5141             atLeastOneError = true;
5142
5143             if (effect(rule[i]) == Deny)
5144             {
5145                 potentialDeny = true;
5146             }
5147             continue;

```

```

5148     }
5149   }
5150   if (potentialDeny)
5151   {
5152     return Indeterminate;
5153   }
5154   if (atLeastOnePermit)
5155   {
5156     return Permit;
5157   }
5158   if (atLeastOneError)
5159   {
5160     return Indeterminate;
5161   }
5162   return NotApplicable;
5163 }

```

5164 The following specification defines the “Deny-overrides” **policy-combining algorithm** of a **policy**
5165 **set**.

5166 In the entire set of **policies** in the **policy set**, if any **policy** evaluates to "Deny", then the
5167 result of the **policy** combination SHALL be "Deny". In other words, "Deny" takes
5168 precedence, regardless of the result of evaluating any of the other **policies** in the **policy**
5169 **set**. If all **policies** are found to be "NotApplicable" to the **decision request**, then the
5170 **policy set** SHALL evaluate to "NotApplicable".

5171 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is
5172 considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set**
5173 SHALL evaluate to "Deny".

5174 The following pseudo-code represents the evaluation strategy of this **policy-combining algorithm**.

```

5175 Decision denyOverridesPolicyCombiningAlgorithm(Policy policy[])
5176 {
5177   Boolean atLeastOnePermit = false;
5178   for( i=0 ; i < lengthOf(policy) ; i++ )
5179   {
5180     Decision decision = evaluate(policy[i]);
5181     if (decision == Deny)
5182     {
5183       return Deny;
5184     }
5185     if (decision == Permit)
5186     {
5187       atLeastOnePermit = true;
5188       continue;
5189     }
5190     if (decision == NotApplicable)
5191     {
5192       continue;
5193     }
5194     if (decision == Indeterminate)
5195     {
5196       return Deny;
5197     }
5198   }
5199   if (atLeastOnePermit)
5200   {
5201     return Permit;
5202   }
5203   return NotApplicable;
5204 }

```

5205 **Obligations** of the individual **policies** shall be combined as described in Section 7.14.

5206 C.2. Ordered-deny-overrides

5207 The following specification defines the "Ordered-deny-overrides" *rule-combining algorithm* of a
5208 *policy*.

5209 The behavior of this algorithm is identical to that of the Deny-overrides *rule-combining*
5210 *algorithm* with one exception. The order in which the collection of *rules* is evaluated SHALL
5211 match the order as listed in the *policy*.

5212 The following specification defines the "Ordered-deny-overrides" *policy-combining algorithm* of a
5213 *policy set*.

5214 The behavior of this algorithm is identical to that of the Deny-overrides *policy-combining*
5215 *algorithm* with one exception. The order in which the collection of *policies* is evaluated
5216 SHALL match the order as listed in the *policy set*.

5217 C.3. Permit-overrides

5218 The following specification defines the "Permit-overrides" *rule-combining algorithm* of a *policy*.

5219 In the entire set of *rules* in the *policy*, if any *rule* evaluates to "Permit", then the result of
5220 the *rule* combination SHALL be "Permit". If any *rule* evaluates to "Deny" and all other
5221 *rules* evaluate to "NotApplicable", then the *policy* SHALL evaluate to "Deny". In other
5222 words, "Permit" takes precedence, regardless of the result of evaluating any of the other
5223 *rules* in the *policy*. If all *rules* are found to be "NotApplicable" to the *decision request*,
5224 then the *policy* SHALL evaluate to "NotApplicable".

5225 If an error occurs while evaluating the *target* or *condition* of a *rule* that contains an *effect*
5226 of "Permit" then the evaluation SHALL continue looking for a result of "Permit". If no other
5227 *rule* evaluates to "Permit", then the *policy* SHALL evaluate to "Indeterminate", with the
5228 appropriate error status.

5229 If at least one *rule* evaluates to "Deny", all other *rules* that do not have evaluation errors
5230 evaluate to "Deny" or "NotApplicable" and all *rules* that do have evaluation errors contain
5231 an *effect* value of "Deny", then the *policy* SHALL evaluate to "Deny".

5232 The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

```
5233 Decision permitOverridesRuleCombiningAlgorithm(Rule rule[])
5234 {
5235     Boolean atLeastOneError = false;
5236     Boolean potentialPermit = false;
5237     Boolean atLeastOneDeny = false;
5238     for( i=0 ; i < lengthOf(rule) ; i++ )
5239     {
5240         Decision decision = evaluate(rule[i]);
5241         if (decision == Deny)
5242         {
5243             atLeastOneDeny = true;
5244             continue;
5245         }
5246         if (decision == Permit)
5247         {
5248             return Permit;
5249         }
5250         if (decision == NotApplicable)
5251         {
5252             continue;
```

```

5253     }
5254     if (decision == Indeterminate)
5255     {
5256         atLeastOneError = true;
5257
5258         if (effect(rule[i]) == Permit)
5259         {
5260             potentialPermit = true;
5261         }
5262         continue;
5263     }
5264 }
5265 if (potentialPermit)
5266 {
5267     return Indeterminate;
5268 }
5269 if (atLeastOneDeny)
5270 {
5271     return Deny;
5272 }
5273 if (atLeastOneError)
5274 {
5275     return Indeterminate;
5276 }
5277 return NotApplicable;
5278 }

```

5279 The following specification defines the "Permit-overrides" **policy-combining algorithm** of a **policy**
5280 **set**.

5281 In the entire set of **policies** in the **policy set**, if any **policy** evaluates to "Permit", then the
5282 result of the **policy** combination SHALL be "Permit". In other words, "Permit" takes
5283 precedence, regardless of the result of evaluating any of the other **policies** in the **policy**
5284 **set**. If all **policies** are found to be "NotApplicable" to the **decision request**, then the
5285 **policy set** SHALL evaluate to "NotApplicable".

5286 If an error occurs while evaluating the **target** of a **policy**, a reference to a **policy** is
5287 considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set**
5288 SHALL evaluate to "Indeterminate", with the appropriate error status, provided no other
5289 **policies** evaluate to "Permit" or "Deny".

5290 The following pseudo-code represents the evaluation strategy of this **policy-combining algorithm**.

```

5291 Decision permitOverridesPolicyCombiningAlgorithm(Policy policy[])
5292 {
5293     Boolean atLeastOneError = false;
5294     Boolean atLeastOneDeny = false;
5295     for( i=0 ; i < lengthOf(policy) ; i++ )
5296     {
5297         Decision decision = evaluate(policy[i]);
5298         if (decision == Deny)
5299         {
5300             atLeastOneDeny = true;
5301             continue;
5302         }
5303         if (decision == Permit)
5304         {
5305             return Permit;
5306         }
5307         if (decision == NotApplicable)
5308         {
5309             continue;
5310         }

```

```

5311     if (decision == Indeterminate)
5312     {
5313         atLeastOneError = true;
5314         continue;
5315     }
5316 }
5317 if (atLeastOneDeny)
5318 {
5319     return Deny;
5320 }
5321 if (atLeastOneError)
5322 {
5323     return Indeterminate;
5324 }
5325 return NotApplicable;
5326 }

```

5327 **Obligations** of the individual *policies* shall be combined as described in Section 7.14.

5328 C.4. Ordered-permit-overrides

5329 The following specification defines the "Ordered-permit-overrides" **rule-combining algorithm** of a
5330 *policy*.

5331 The behavior of this algorithm is identical to that of the Permit-overrides **rule-combining**
5332 **algorithm** with one exception. The order in which the collection of *rules* is evaluated SHALL
5333 match the order as listed in the *policy*.

5334 The following specification defines the "Ordered-permit-overrides" **policy-combining algorithm** of
5335 a *policy set*.

5336 The behavior of this algorithm is identical to that of the Permit-overrides **policy-combining**
5337 **algorithm** with one exception. The order in which the collection of *policies* is evaluated
5338 SHALL match the order as listed in the *policy set*.

5339 C.5. First-applicable

5340 The following specification defines the "First-Applicable " **rule-combining algorithm** of a *policy*.

5341 Each *rule* SHALL be evaluated in the order in which it is listed in the *policy*. For a
5342 particular *rule*, if the **target** matches and the **condition** evaluates to "True", then the
5343 evaluation of the *policy* SHALL halt and the corresponding **effect** of the *rule* SHALL be the
5344 result of the evaluation of the *policy* (i.e. "Permit" or "Deny"). For a particular *rule* selected
5345 in the evaluation, if the **target** evaluates to "False" or the **condition** evaluates to "False",
5346 then the next *rule* in the order SHALL be evaluated. If no further *rule* in the order exists,
5347 then the *policy* SHALL evaluate to "NotApplicable".

5348 If an error occurs while evaluating the **target** or **condition** of a *rule*, then the evaluation
5349 SHALL halt, and the *policy* shall evaluate to "Indeterminate", with the appropriate error
5350 status.

5351 The following pseudo-code represents the evaluation strategy of this **rule-combining algorithm**.

5352

5353


```

5354 Decision firstApplicableEffectRuleCombiningAlgorithm(Rule rule[])
5355 {
5356     for( i = 0 ; i < lengthOf(rule) ; i++ )
5357     {
5358         Decision decision = evaluate(rule[i]);
5359         if (decision == Deny)
5360         {
5361             return Deny;
5362         }
5363         if (decision == Permit)
5364         {
5365             return Permit;
5366         }
5367         if (decision == NotApplicable)
5368         {
5369             continue;
5370         }
5371         if (decision == Indeterminate)
5372         {
5373             return Indeterminate;
5374         }
5375     }
5376     return NotApplicable;
5377 }

```

5378 The following specification defines the "First-applicable" **policy-combining algorithm** of a **policy**
5379 **set**.

5380 Each **policy** is evaluated in the order that it appears in the **policy set**. For a particular
5381 **policy**, if the **target** evaluates to "True" and the **policy** evaluates to a determinate value of
5382 "Permit" or "Deny", then the evaluation SHALL halt and the **policy set** SHALL evaluate to
5383 the **effect** value of that **policy**. For a particular **policy**, if the **target** evaluate to "False", or
5384 the **policy** evaluates to "NotApplicable", then the next **policy** in the order SHALL be
5385 evaluated. If no further **policy** exists in the order, then the **policy set** SHALL evaluate to
5386 "NotApplicable".

5387 If an error were to occur when evaluating the **target**, or when evaluating a specific **policy**,
5388 the reference to the **policy** is considered invalid, or the **policy** itself evaluates to
5389 "Indeterminate", then the evaluation of the **policy-combining algorithm** shall halt, and the
5390 **policy set** shall evaluate to "Indeterminate" with an appropriate error status.

5391 The following pseudo-code represents the evaluation strategy of this **policy-combination**
5392 **algorithm**.

```

5393 Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy policy[])
5394 {
5395     for( i = 0 ; i < lengthOf(policy) ; i++ )
5396     {
5397         Decision decision = evaluate(policy[i]);
5398         if(decision == Deny)
5399         {
5400             return Deny;
5401         }
5402         if(decision == Permit)
5403         {
5404             return Permit;
5405         }
5406         if (decision == NotApplicable)
5407         {
5408             continue;
5409         }
5410         if (decision == Indeterminate)
5411         {

```

```

5412         return Indeterminate;
5413     }
5414 }
5415     return NotApplicable;
5416 }

```

5417 **Obligations** of the individual **policies** shall be combined as described in Section 7.14.

5418 C.6. Only-one-applicable

5419 The following specification defines the “Only-one-applicable” **policy-combining algorithm** of a
5420 **policy set**.

5421 In the entire set of **policies** in the **policy set**, if no **policy** is considered applicable by virtue
5422 of its **target**, then the result of the **policy** combination algorithm SHALL be "NotApplicable".
5423 If more than one **policy** is considered applicable by virtue of its **target**, then the result of
5424 the **policy** combination algorithm SHALL be "Indeterminate".

5425 If only one **policy** is considered applicable by evaluation of its **target**, then the result of the
5426 **policy-combining algorithm** SHALL be the result of evaluating the **policy**.

5427 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is
5428 considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set**
5429 SHALL evaluate to "Indeterminate", with the appropriate error status.

5430 The following pseudo-code represents the evaluation strategy of this policy combining algorithm.

```

5431 Decision onlyOneApplicablePolicyPolicyCombiningAlgoithm(Policy policy[])
5432 {
5433     Boolean          atLeastOne      = false;
5434     Policy           selectedPolicy = null;
5435     ApplicableResult appResult;
5436
5437     for ( i = 0; i < lengthOf(policy) ; i++ )
5438     {
5439         appResult = isApplicable(policy[i]);
5440
5441         if ( appResult == Indeterminate )
5442         {
5443             return Indeterminate;
5444         }
5445         if( appResult == Applicable )
5446         {
5447             if ( atLeastOne )
5448             {
5449                 return Indeterminate;
5450             }
5451             else
5452             {
5453                 atLeastOne      = true;
5454                 selectedPolicy = policy[i];
5455             }
5456         }
5457         if ( appResult == NotApplicable )
5458         {
5459             continue;
5460         }
5461     }
5462     if ( atLeastOne )
5463     {

```

```
5464     return evaluate(selectedPolicy);
5465 }
5466 else
5467 {
5468     return NotApplicable;
5469 }
5470 }
5471
```

5472 **Appendix D. Acknowledgments**

5473 The following individuals contributed to the development of the specification:

5474 Anne Anderson
5475 Anthony Nadalin
5476 Bill Parducci
5477 Carlisle Adams
5478 Daniel Engovatov
5479 Don Flinn
5480 Ed Coyne
5481 Ernesto Damiani
5482 Frank Siebenlist
5483 Gerald Brose
5484 Hal Lockhart
5485 James MacLean
5486 John Merrells
5487 Ken Yagen
5488 Konstantin Beznosov
5489 Michiharu Kudo
5490 Michael McIntosh
5491 Pierangela Samarati
5492 Pirasenna Velandai Thiyagarajan
5493 Polar Humenn
5494 Rebekah Lepro
5495 Ron Jacobson
5496 Satoshi Hada
5497 Sekhar Vajjhala
5498 Seth Proctor
5499 Simon Godik
5500 Steve Anderson
5501 Steve Crocker
5502 Suresh Damodaran
5503 Tim Moses
5504 Von Welch
5505

5506

Appendix E. Revision history

Rev	Date	By whom	What
CD 01	16 Sep 2004	Access Control TC	First committee draft

5507

5508

Appendix F. Notices

5509 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
5510 that might be claimed to pertain to the implementation or use of the technology described in this
5511 document or the extent to which any license under such rights might or might not be available;
5512 neither does it represent that it has made any effort to identify any such rights. Information on
5513 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
5514 website. Copies of claims of rights made available for publication and any assurances of licenses to
5515 be made available, or the result of an attempt made to obtain a general license or permission for
5516 the use of such proprietary rights by implementers or users of this specification, can be obtained
5517 from the OASIS Executive Director.

5518 OASIS has been notified of intellectual property rights claimed in regard to some or all of the
5519 contents of this specification. For more information consult the online list of claimed rights.

5520 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
5521 applications, or other proprietary rights which may cover technology that may be required to
5522 implement this specification. Please address the information to the OASIS Executive Director.

5523 Copyright (C) OASIS Open 2004. All Rights Reserved.

5524 This document and translations of it may be copied and furnished to others, and derivative works
5525 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
5526 published and distributed, in whole or in part, without restriction of any kind, provided that the above
5527 copyright notice and this paragraph are included on all such copies and derivative works. However,
5528 this document itself may not be modified in any way, such as by removing the copyright notice or
5529 references to OASIS, except as needed for the purpose of developing OASIS specifications, in
5530 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights
5531 document must be followed, or as required to translate it into languages other than English.

5532 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
5533 successors or assigns.

5534 This document and the information contained herein is provided on an "AS IS" basis and OASIS
5535 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
5536 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY
5537 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
5538 PARTICULAR PURPOSE.