



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

4 **Committee draft 02, 30 Sep 2004**

5 Document identifier: access\_control-xacml-2.0-core-spec-cd-02

6 Location: [http://docs.oasis-open.org/xacml/access\\_control-xacml-2.0-core-spec-cd-02.pdf](http://docs.oasis-open.org/xacml/access_control-xacml-2.0-core-spec-cd-02.pdf)

7 Editor:

8 Tim Moses, Entrust

9 Committee members:

10 Anne Anderson, Sun Microsystems  
11 Anthony Nadalin, IBM  
12 Bill Parducci, GlueCode Software  
13 Daniel Engovatov, BEA Systems  
14 Ed Coyne, Veterans Health Administration  
15 Frank Siebenlist, Argonne National Labs  
16 Hal Lockhart, BEA Systems  
17 Michael McIntosh, IBM  
18 Michiharu Kudo, IBM  
19 Polar Humenn, Self  
20 Ron Jacobson, Computer Associates  
21 Seth Proctor, Sun Microsystems  
22 Simon Godik, GlueCode Software  
23 Steve Anderson, OpenNetwork  
24 Tim Moses, Entrust

25 Abstract:

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

27 Status:

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

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

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

access\_control-xacml-2.0-core-spec-cd-02

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

40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75

## Table of contents

1.	Introduction (non-normative).....	9
1.1.	Glossary.....	9
1.1.1	Preferred terms .....	9
1.1.2	Related terms.....	10
1.2.	Notation .....	11
1.3.	Schema organization and namespaces .....	11
2.	Background (non-normative) .....	12
2.1.	Requirements.....	12
2.2.	Rule and policy combining.....	13
2.3.	Combining algorithms.....	13
2.4.	Multiple subjects.....	14
2.5.	Policies based on subject and resource attributes.....	14
2.6.	Multi-valued attributes .....	15
2.7.	Policies based on resource contents .....	15
2.8.	Operators.....	15
2.9.	Policy distribution .....	16
2.10.	Policy indexing.....	16
2.11.	Abstraction layer .....	16
2.12.	Actions performed in conjunction with enforcement .....	17
3.	Models (non-normative).....	17
3.1.	Data-flow model .....	17
3.2.	XACML context .....	19
3.3.	Policy language model .....	19
3.3.1	Rule .....	21
3.3.2	Policy .....	22
3.3.3	Policy set.....	23
4.	Examples (non-normative) .....	23
4.1.	Example one.....	24
4.1.1	Example policy .....	24
4.1.2	Example request context .....	26
4.1.3	Example response context.....	27
4.2.	Example two .....	27
4.2.1	Example medical record instance .....	27
4.2.2	Example request context .....	29
4.2.3	Example plain-language rules.....	30

76	4.2.4	Example XACML rule instances.....	31
77	5.	Policy syntax (normative, with the exception of the schema fragments).....	43
78	5.1.	Element <PolicySet> .....	43
79	5.2.	Element <Description> .....	46
80	5.3.	Element <PolicySetDefaults> .....	46
81	5.4.	Element <XPathVersion> .....	46
82	5.5.	Element <Target> .....	46
83	5.6.	Element <Subjects> .....	47
84	5.7.	Element <Subject> .....	47
85	5.8.	Element <SubjectMatch> .....	48
86	5.9.	Element <Resources> .....	48
87	5.10.	Element <Resource>.....	49
88	5.11.	Element <ResourceMatch> .....	49
89	5.12.	Element <Actions> .....	50
90	5.13.	Element <Action>.....	50
91	5.14.	Element <ActionMatch> .....	50
92	5.15.	Element <Environments> .....	51
93	5.16.	Element <Environment> .....	51
94	5.17.	Element <EnvironmentMatch>.....	52
95	5.18.	Element <PolicySetIdReference> .....	52
96	5.19.	Element <PolicyIdReference> .....	53
97	5.20.	Simple type VersionType.....	53
98	5.21.	Simple type VersionMatchType .....	53
99	5.22.	Element <Policy> .....	54
100	5.23.	Element <PolicyDefaults> .....	56
101	5.24.	Element <CombinerParameters> .....	56
102	5.25.	Element <CombinerParameter> .....	56
103	5.26.	Element <RuleCombinerParameters> .....	57
104	5.27.	Element <PolicyCombinerParameters> .....	57
105	5.28.	Element <PolicySetCombinerParameters> .....	58
106	5.29.	Element <Rule> .....	59
107	5.30.	Simple type EffectType.....	59
108	5.31.	Element <VariableDefinition> .....	60
109	5.32.	Element <VariableReference>.....	60
110	5.33.	Element <Expression> .....	61
111	5.34.	Element <Condition>.....	61
112	5.35.	Element <Apply>.....	61

113	5.36.	Element <Function> .....	62
114	5.37.	Complex type AttributeDesignatorType.....	62
115	5.38.	Element <SubjectAttributeDesignator> .....	63
116	5.39.	Element <ResourceAttributeDesignator> .....	64
117	5.40.	Element <ActionAttributeDesignator> .....	64
118	5.41.	Element <EnvironmentAttributeDesignator> .....	65
119	5.42.	Element <AttributeSelector>.....	65
120	5.43.	Element <AttributeValue>.....	67
121	5.44.	Element <Obligations> .....	67
122	5.45.	Element <Obligation>.....	67
123	5.46.	Element <AttributeAssignment> .....	68
124	6.	Context syntax (normative with the exception of the schema fragments).....	69
125	6.1.	Element <Request>.....	69
126	6.2.	Element <Subject> .....	70
127	6.3.	Element <Resource>.....	70
128	6.4.	Element <ResourceContent> .....	71
129	6.5.	Element <Action>.....	71
130	6.6.	Element <Environment> .....	72
131	6.7.	Element <Attribute>.....	72
132	6.8.	Element <AttributeValue>.....	73
133	6.9.	Element <Response> .....	73
134	6.10.	Element <Result>.....	74
135	6.11.	Element <Decision> .....	74
136	6.12.	Element <Status>.....	75
137	6.13.	Element <StatusCode> .....	75
138	6.14.	Element <StatusMessage>.....	76
139	6.15.	Element <StatusDetail>.....	76
140	6.16.	Element <MissingAttributeDetail>.....	77
141	7.	Functional requirements (normative) .....	77
142	7.1.	Policy enforcement point .....	77
143	7.1.1.	Base PEP.....	78
144	7.1.2.	Deny-biased PEP .....	78
145	7.1.3.	Permit-biased PEP .....	78
146	7.2.	Attribute evaluation .....	78
147	7.2.1.	Structured attributes .....	78
148	7.2.2.	Attribute bags .....	79
149	7.2.3.	Multivalued attributes.....	79

150	7.2.4.	Attribute Matching.....	79
151	7.2.5.	Attribute Retrieval.....	80
152	7.2.6.	Environment Attributes .....	80
153	7.3.	Expression evaluation .....	80
154	7.4.	Arithmetic evaluation.....	81
155	7.5.	Match evaluation.....	81
156	7.6.	Target evaluation .....	83
157	7.7.	VariableReference Evaluation .....	84
158	7.8.	Condition evaluation.....	84
159	7.9.	Rule evaluation .....	84
160	7.10.	Policy evaluation.....	85
161	7.11.	Policy Set evaluation.....	86
162	7.12.	Hierarchical resources.....	87
163	7.13.	Authorization decision .....	87
164	7.14.	Obligations.....	87
165	7.15.	Exception handling.....	87
166	7.15.1.	Unsupported functionality .....	88
167	7.15.2.	Syntax and type errors.....	88
168	7.15.3.	Missing attributes.....	88
169	8.	XACML extensibility points (non-normative).....	88
170	8.1.	Extensible XML attribute types .....	89
171	8.2.	Structured attributes.....	89
172	9.	Security and privacy considerations (non-normative).....	89
173	9.1.	Threat model.....	90
174	9.1.1.	Unauthorized disclosure .....	90
175	9.1.2.	Message replay .....	90
176	9.1.3.	Message insertion .....	90
177	9.1.4.	Message deletion .....	91
178	9.1.5.	Message modification.....	91
179	9.1.6.	NotApplicable results.....	91
180	9.1.7.	Negative rules .....	91
181	9.2.	Safeguards .....	92
182	9.2.1.	Authentication.....	92
183	9.2.2.	Policy administration.....	92
184	9.2.3.	Confidentiality.....	93
185	9.2.4.	Policy integrity .....	93
186	9.2.5.	Policy identifiers .....	94

187	9.2.6.	Trust model .....	94
188	9.2.7.	Privacy .....	94
189	10.	Conformance (normative) .....	95
190	10.1.	Introduction.....	95
191	10.2.	Conformance tables.....	95
192	10.2.1.	Schema elements.....	95
193	10.2.2.	Identifier Prefixes.....	96
194	10.2.3.	Algorithms .....	96
195	10.2.4.	Status Codes.....	97
196	10.2.5.	Attributes.....	97
197	10.2.6.	Identifiers .....	97
198	10.2.7.	Data-types.....	98
199	10.2.8.	Functions .....	98
200	11.	References.....	102
201	Appendix A.	Data-types and functions (normative).....	105
202	A.1.	Introduction .....	105
203	A.2.	Data-types .....	105
204	A.3.	Functions .....	107
205	A.3.1	Equality predicates .....	107
206	A.3.2	Arithmetic functions .....	109
207	A.3.3	String conversion functions.....	110
208	A.3.4	Numeric data-type conversion functions.....	110
209	A.3.5	Logical functions.....	110
210	A.3.6	Numeric comparison functions.....	111
211	A.3.7	Date and time arithmetic functions .....	112
212	A.3.8	Non-numeric comparison functions.....	113
213	A.3.9	String functions.....	116
214	A.3.10	Bag functions.....	116
215	A.3.11	Set functions .....	117
216	A.3.12	Higher-order bag functions .....	117
217	A.3.13	Regular-expression-based functions .....	124
218	A.3.14	Special match functions.....	125
219	A.3.15	XPath-based functions.....	126
220	A.3.16	Extension functions and primitive types.....	126
221	Appendix B.	XACML identifiers (normative).....	127
222	B.1.	XACML namespaces.....	127
223	B.2.	Access subject categories.....	127

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



242

---

## 243 1. Introduction (non-normative)

### 244 1.1. Glossary

#### 245 1.1.1 Preferred terms

246 **Access** - Performing an *action*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 306 1.1.2 Related terms

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

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

## 314 1.2. Notation

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

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

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

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

326 Listings of XACML schema appear like this.

327  
328 [a01] Example code listings appear like this.

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

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

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

## 341 1.3. Schema organization and namespaces

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

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

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

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

346

## 2. Background (non-normative)

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

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

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

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

### 373 2.1. Requirements

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

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

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

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

## 398 2.2. Rule and policy combining

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

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

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

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

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

## 418 2.3. Combining algorithms

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

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

- 428 • First-applicable and
- 429 • Only-one-applicable.

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

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

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

438 The "Only-one-applicable" **policy-combining algorithm** only applies to **policies**. The result of this  
439 combining algorithm ensures that one and only one **policy** or **policy set** is applicable by virtue of  
440 their **targets**. If no **policy** or **policy set** applies, then the result is "NotApplicable", but if more than  
441 one **policy** or **policy set** is applicable, then the result is "Indeterminate". When exactly one **policy**  
442 or **policy set** is applicable, the result of the combining algorithm is the result of evaluating the  
443 single **applicable policy** or **policy set**.

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

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

## 447 2.4. Multiple subjects

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

## 454 2.5. Policies based on subject and resource attributes

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

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

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



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

## 472 **2.6. Multi-valued attributes**

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

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

## 482 **2.7. Policies based on resource contents**

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

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

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

## 493 **2.8. Operators**

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

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

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

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

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

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

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

## 522 2.9. Policy distribution

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

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

## 534 2.10. Policy indexing

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

540 Two approaches are supported:

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

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

## 549 2.11. Abstraction layer

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



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

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

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

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

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

## 572 **2.12. Actions performed in conjunction with enforcement**

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

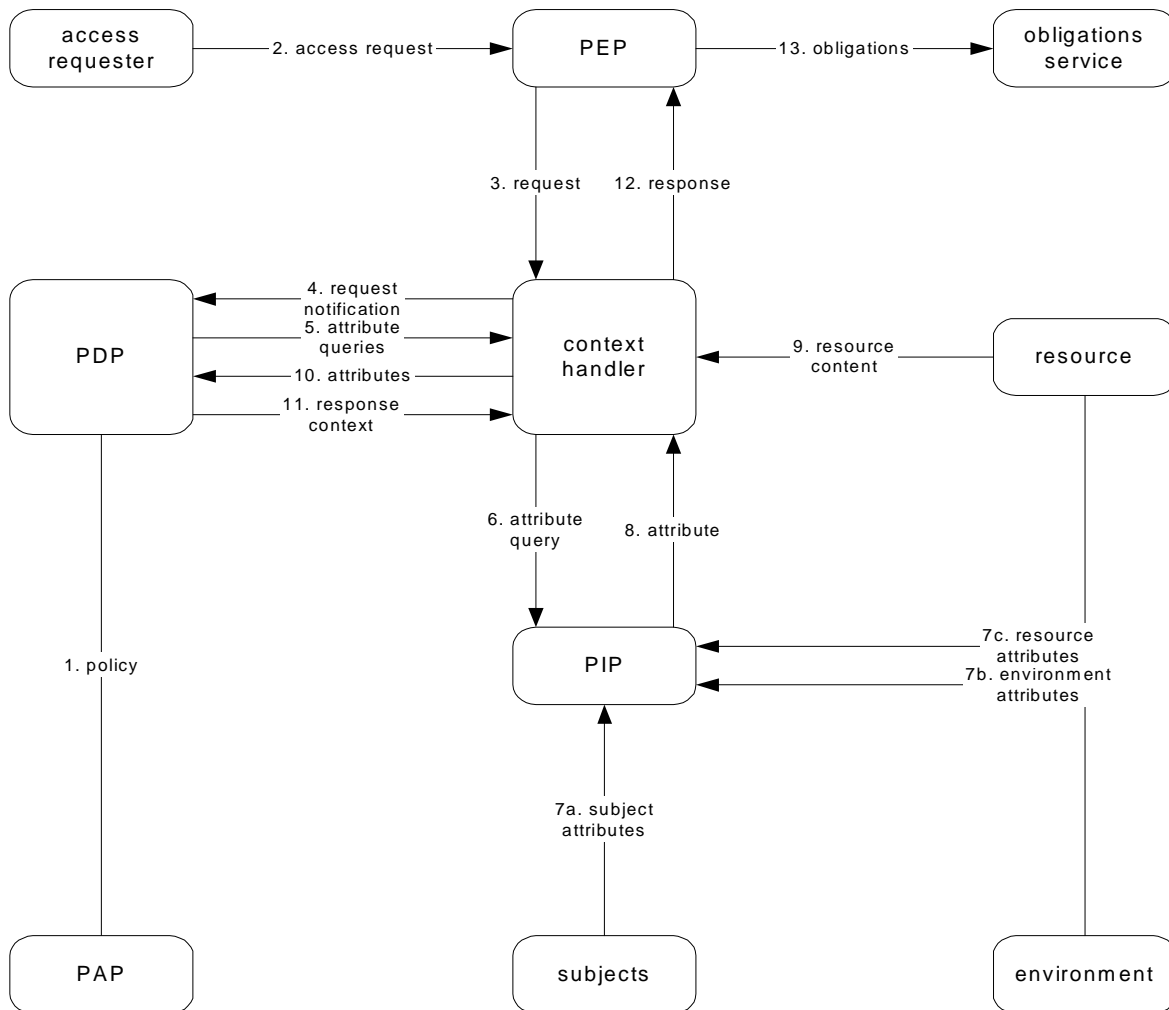
---

## 582 **3. Models (non-normative)**

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

### 584 **3.1. Data-flow model**

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



586

587

**Figure 1 - Data-flow diagram**

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

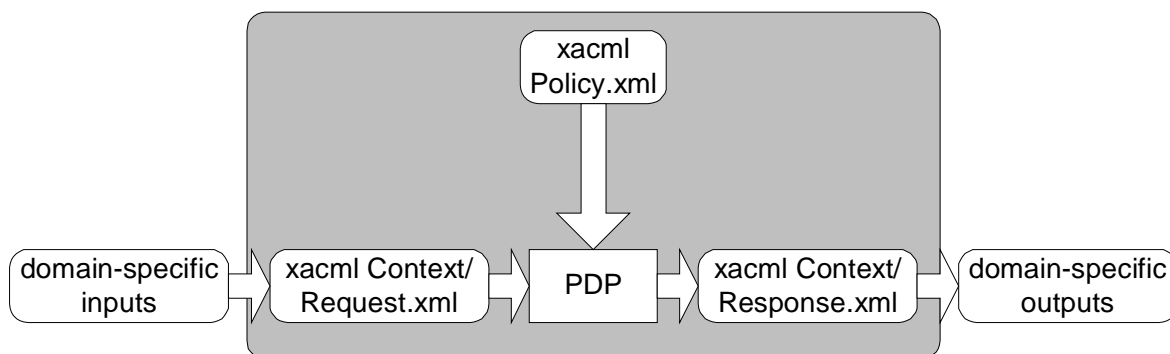
593 The model operates by the following steps.

- 594 1. **PAPs** write **policies** and **policy sets** and make them available to the **PDP**. These **policies** or  
 595 **policy sets** represent the complete policy for a specified **target**.
- 596 2. The access requester sends a request for access to the **PEP**.
- 597 3. The **PEP** sends the request for **access** to the **context handler** in its native request format,  
 598 optionally including **attributes** of the **subjects**, **resource**, **action** and **environment**.
- 599 4. The **context handler** constructs an XACML request **context** and sends it to the **PDP**.
- 600 5. The **PDP** requests any additional **subject**, **resource**, **action** and **environment attributes** from  
 601 the **context handler**.
- 602 6. The context handler requests the attributes from a **PIP**.

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

### 615 3.2. XACML context

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



628

629

Figure 2 - XACML context

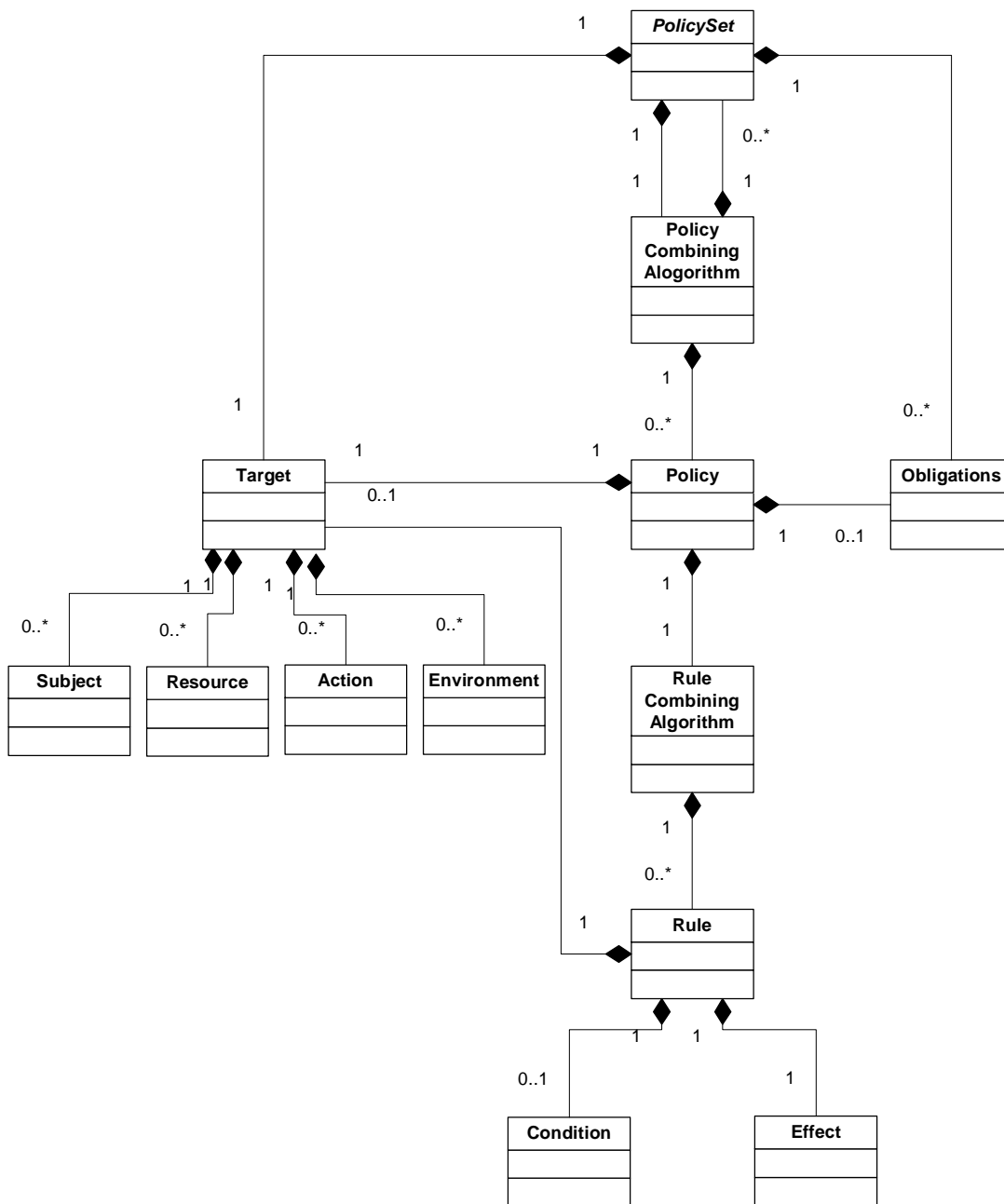
630 Note: The **PDP** is not required to operate directly on the XACML representation of a policy. It may  
 631 operate directly on an alternative representation.

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

### 633 3.3. Policy language model

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

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



639

640

**Figure 3 - Policy language model**

641

### 3.3.1 Rule

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

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

649 These are discussed in the following sub-sections.

650

#### 3.3.1.1. Rule target

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

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

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

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

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

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

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

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

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

### 684 3.3.1.2. Effect

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

### 687 3.3.1.3. Condition

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

## 690 3.3.2 Policy

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

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

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

### 699 3.3.2.1. Policy target

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

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

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

### 721 3.3.2.2. Rule-combining algorithm

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

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

### 728 3.3.2.3. Obligations

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

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

## 732 3.3.3 Policy set

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

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

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

### 740 3.3.3.1. Policy-combining algorithm

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

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

### 747 3.3.3.2. Obligations

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

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

---

## 752 4. Examples (non-normative)

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



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

## 757 4.1. Example one

### 758 4.1.1 Example policy

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

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

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

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

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

804 [a03] introduces the XACML Policy itself.

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



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

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

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

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

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

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

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

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

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

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

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

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

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

851

## 4.1.2 Example request context

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

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

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

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

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

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

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

---

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

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

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

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

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

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

### 917 4.1.3 Example response context

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

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

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

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

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

## 936 4.2. Example two

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

### 940 4.2.1 Example medical record instance

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

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

```

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

```

```

1009 [a135] <drug>
1010 [a136] <name>methylphenidate hydrochloride</name>
1011 [a137] <dailyDosage>30mgs</dailyDosage>
1012 [a138] <startDate>1999-01-12</startDate>
1013 [a139] </drug>
1014 [a140] <comment>
1015 [a141] patient exhibits side-effects of skin coloration and carpal
1016 degeneration
1017 [a142] </comment>
1018 [a143] </treatment>
1019 [a144] <result>
1020 [a145] <test>blood pressure</test>
1021 [a146] <value>120/80</value>
1022 [a147] <date>2001-06-09</date>
1023 [a148] <performedBy>Nurse Betty</performedBy>
1024 [a149] </result>
1025 [a150] </medical>
1026 [a151] </record>

```

## 1027 4.2.2 Example request context

1028 The following example illustrates a request *context* to which the example *rules* may be applicable.  
1029 It represents a request by the physician Julius Hibbert to read the patient date of birth in the record  
1030 of Bartholomew Simpson.

```

1031 [a152] <?xml version="1.0" encoding="UTF-8"?>
1032 [a153] <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:cd"
1033 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
1034 urn:oasis:names:tc:xacml:2.0:context:schema:cd http://docs.oasis-
1035 open.org/xacml/access_control-xacml-2.0-context-schema-cd.xsd">
1036 [a154] <Subject>
1037 [a155] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject-category"
1038 DataType="http://www.w3.org/2001/XMLSchema#anyURI">
1039 [a156] <AttributeValue>urn:oasis:names:tc:xacml:1.0:subject-category:access-
1040 subject</AttributeValue>
1041 [a157] </Attribute>
1042 [a158] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
1043 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="med.example.com">
1044 [a159] <AttributeValue>CN=Julius Hibbert</AttributeValue>
1045 [a160] </Attribute>
1046 [a161] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:name-
1047 format" DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1048 Issuer="med.example.com">
1049 [a162] <AttributeValue>
1050 [a163] urn:oasis:names:tc:xacml:1.0:datatype:x500name
1051 [a164] </AttributeValue>
1052 [a165] </Attribute>
1053 [a166] <Attribute
1054 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:role"
1055 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="med.example.com">
1056 [a167] <AttributeValue>physician</AttributeValue>
1057 [a168] </Attribute>
1058 [a169] <Attribute
1059 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:physician-id"
1060 DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="med.example.com">
1061 [a170] <AttributeValue>jh1234</AttributeValue>
1062 [a171] </Attribute>
1063 [a172] </Subject>
1064 [a173] <Resource>
1065 [a174] <ResourceContent>
1066 [a175] <md:record xmlns:md="urn:example:med:schemas:record"
1067 xsi:schemaLocation="urn:example:med:schemas:record
1068 http://www.med.example.com/schemas/record.xsd">

```

```

1069 [a176] <md:patient>
1070 [a177] <md:patientDoB>1992-03-21</md:patientDoB>
1071 [a178] <md:patient-number>555555</md:patient-number>
1072 [a179] </md:patient>
1073 [a180] </md:record>
1074 [a181] </ResourceContent>
1075 [a182] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-
1076 id" DataType="http://www.w3.org/2001/XMLSchema#string">
1077 [a183] <AttributeValue>
1078 [a184] //med.example.com/records/bart-simpson.xml#
1079 [a185] xmlns(md=:Resource/ResourceContent/xpointer
1080 [a186] (/md:record/md:patient/md:patientDoB)
1081 [a187] </AttributeValue>
1082 [a188] </Attribute>
1083 [a189] </Resource>
1084 [a190] <Action>
1085 [a191] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1086 DataType="http://www.w3.org/2001/XMLSchema#string">
1087 [a192] <AttributeValue>read</AttributeValue>
1088 [a193] </Attribute>
1089 [a194] </Action>
1090 [a195] <Environment/>
1091 [a196] </Request>

```

1092 [a152] - [a153] Standard namespace declarations.

1093 [a154] - [a172] **Subject** attributes are placed in the <Subject> element of the <Request>  
1094 element. Each **attribute** consists of the **attribute** meta-data and the **attribute** value. There is only  
1095 one subject involved in this **request**.

1096 [a155] - [a157] Each <Subject> element has a SubjectCategory attribute. The value of this  
1097 attribute describes the role that the related **subject** plays in making the **decision request**. The  
1098 value of "access-subject" denotes the identity for which the request was issued.

1099 [a158] - [a160] **Subject** subject-id **attribute**.

1100 [a161] - [a165] The format of the subject-id.

1101 [a166] - [a168] **Subject** role **attribute**.

1102 [a169] - [a171] **Subject** physician-id **attribute**.

1103 [a173] - [a189] **Resource attributes** are placed in the <Resource> element of the <Request>  
1104 element. Each **attribute** consists of **attribute** meta-data and an **attribute** value.

1105 [a174] - [a181] **Resource** content. The XML resource instance, access to all or part of which may  
1106 be requested, is placed here.

1107 [a182] - [a188] The identifier of the **Resource** instance for which access is requested, which is an  
1108 XPath expression into the <ResourceContent> element that selects the data to be accessed.

1109 [a190] - [a194] **Action attributes** are placed in the <Action> element of the <Request> element.

1110 [a192] **Action** identifier.

1111 [a195] The empty <Environment> element.

## 1112 4.2.3 Example plain-language rules

1113 The following plain-language rules are to be enforced:



- 1114 Rule 1: A person, identified by his or her patient number, may read any record for which he  
1115 or she is the designated patient.
- 1116 Rule 2: A person may read any record for which he or she is the designated parent or  
1117 guardian, and for which the patient is under 16 years of age.
- 1118 Rule 3: A physician may write to any medical element for which he or she is the designated  
1119 primary care physician, provided an email is sent to the patient.
- 1120 Rule 4: An administrator shall not be permitted to read or write to medical elements of a  
1121 patient record.
- 1122 These **rules** may be written by different **PAPs** operating independently, or by a single **PAP**.

## 1123 4.2.4 Example XACML rule instances

### 1124 4.2.4.1. Rule 1

1125 Rule 1 illustrates a simple **rule** with a single <Condition> element. It also illustrates the use of  
1126 the <VariableDefinition> element to define a function that may be used throughout the  
1127 **policy**. The following XACML <Rule> instance expresses Rule 1:

```
1128 [a197] <?xml version="1.0" encoding="UTF-8"?>
1129 [a198] <Policy
1130 [a199] xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd" xmlns:xacml-
1131 context="urn:oasis:names:tc:xacml:2.0:context:schema:cd"
1132 [a200] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
1133 urn:oasis:names:tc:xacml:2.0:policy:schema:cd http://docs.oasis-
1134 open.org/xacml/access_control-xacml-2.0-context-schema-cd.xsd"
1135 [a201] xmlns:md="http://www.med.example.com/schemas/record.xsd"
1136 [a202] PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:1"
1137 [a203] RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1138 algorithm:deny-overrides">
1139 [a204] <PolicyDefaults>
1140 [a205] <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-
1141 19991116</XPathVersion>
1142 [a206] </PolicyDefaults>
1143 [a207] <Target/>
1144 [a208] <VariableDefinition VariableId="17590034">
1145 [a209] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1146 [a210] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1147 and-only">
1148 [a211] <SubjectAttributeDesignator
1149 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:patient-number"
1150 [a212] <DataTypes="http://www.w3.org/2001/XMLSchema#string"/>
1151 [a213] </Apply>
1152 [a214] <Apply
1153 [a215] FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1154 [a216] <AttributeSelector
1155 [a217] RequestContextPath="//xacml-context:Resource/xacml-
1156 context:ResourceContent/md:record/md:patient/md:patient-number/text()"
1157 [a218] <DataTypes="http://www.w3.org/2001/XMLSchema#string"/>
1158 [a219] </Apply>
1159 [a220] </Apply>
1160 [a221] </VariableDefinition>
1161 [a222] <Rule
1162 [a223] RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:1"
1163 [a224] Effect="Permit">
1164 [a225] <Description>
1165 [a226] A person may read any medical record in the
1166 [a227] http://www.med.example.com/schemas/record.xsd namespace
```

```

1167 [a228]     for which he or she is the designated patient
1168 [a229] </Description>
1169 [a230] <Target>
1170 [a231]   <Resources>
1171 [a232]     <Resource>
1172 [a233]       <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1173 equal">
1174 [a234]         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1175 [a235]           urn:example:med:schemas:record
1176 [a236]         </AttributeValue>
1177 [a237]         <ResourceAttributeDesignator AttributeId=
1178 [a238]           "urn:oasis:names:tc:xacml:2.0:resource:target -namespace"
1179 [a239]           DataType="http://www.w3.org/2001/XMLSchema#string"/>
1180 [a240]         </ResourceMatch>
1181 [a241]       <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-
1182 node-match">
1183 [a242]         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1184 [a243]           /md:record
1185 [a244]         </AttributeValue>
1186 [a245]         <ResourceAttributeDesignator
1187 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
1188 [a246]           DataType="http://www.w3.org/2001/XMLSchema#string"/>
1189 [a247]         </ResourceMatch>
1190 [a248]       </Resource>
1191 [a249]     </Resources>
1192 [a250]   <Actions>
1193 [a251]     <Action>
1194 [a252]       <ActionMatch
1195 [a253]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1196 [a254]           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1197 [a255]             read
1198 [a256]           </AttributeValue>
1199 [a257]           <ActionAttributeDesignator
1200 [a258]             AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1201 [a259]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1202 [a260]           </ActionMatch>
1203 [a261]         </Action>
1204 [a262]       </Actions>
1205 [a263]     </Target>
1206 [a264]   <Condition>
1207 [a265]     <VariableReference VariableId="17590034"/>
1208 [a266]   </Condition>
1209 [a267] </Rule>
1210 [a268] </Policy>

```

1211 [a199] - [a201]. XML namespace declarations.

1212 [a205] XPath expressions in the *policy* are to be interpreted according to the 1.0 version of the  
1213 XPath specification.

1214 [a208] - [a221] A <VariableDefinition> element. It defines a function that evaluates the truth  
1215 of the statement: the patient-number *subject attribute* is equal to the patient-number in the  
1216 *resource*.

1217 [a209] The FunctionId attribute names the function to be used for comparison. In this case,  
1218 comparison is done with the "urn:oasis:names:tc:xacml:1.0:function:string-equal" function; this  
1219 function takes two arguments of type "http://www.w3.org/2001/XMLSchema#string".

1220 [a210] The first argument of the variable definition is a function specified by the FunctionId  
1221 attribute. Since urn:oasis:names:tc:xacml:1.0:function:string-equal takes  
1222 arguments of type "http://www.w3.org/2001/XMLSchema#string" and  
1223 SubjectAttributeDesignator selects a *bag* of type



1224 "http://www.w3.org/2001/XMLSchema#string", "urn:oasis:names:tc:xacml:1.0:function:string-one-  
1225 and-only" is used. This function guarantees that its argument evaluates to a **bag** containing exactly  
1226 one value.

1227 [a211] The `SubjectAttributeDesignator` selects a **bag** of values for the `patient-number`  
1228 **subject attribute** in the request **context**.

1229 [a215] The second argument of the variable definition is a function specified by the `FunctionId`  
1230 attribute. Since "urn:oasis:names:tc:xacml:1.0:function:string-equal" takes arguments of type  
1231 "http://www.w3.org/2001/XMLSchema#string" and the `AttributeSelector` selects a **bag** of type  
1232 "http://www.w3.org/2001/XMLSchema#string", "urn:oasis:names:tc:xacml:1.0:function:string-one-  
1233 and-only" is used. This function guarantees that its argument evaluates to a **bag** containing exactly  
1234 one value.

1235 [a216] The `<AttributeSelector>` element selects a **bag** of values from the request **context**  
1236 using a free-form XPath expression. In this case, it selects the value of the `patient-number` in  
1237 the **resource**. Note that the namespace prefixes in the XPath expression are resolved with the  
1238 standard XML namespace declarations.

1239 [a223] **Rule** identifier.

1240 [a224]. **Rule effect** declaration. When a **rule** evaluates to 'True' it emits the value of the `Effect`  
1241 attribute. This value is then combined with the `Effect` values of other **rules** according to the **rule-**  
1242 **combining algorithm**.

1243 [a225] - [a229] Free form description of the **rule**.

1244 [a230] - [a263]. A **rule target** defines a set of **decision requests** that the **rule** is intended to  
1245 evaluate. In this example, the `<Subjects>` and `<Environments>` elements are omitted.

1246 [a231] - [a249] The `<Resources>` element contains a **disjunctive sequence** of `<Resource>`  
1247 elements. In this example, there is just one.

1248 [a232] - [a248] The `<Resource>` element encloses the **conjunctive sequence** of  
1249 `ResourceMatch` elements. In this example, there are two.

1250 [a233] - [a240] The first `<ResourceMatch>` element compares its first and second child elements  
1251 according to the matching function. A match is positive if the value of the first argument matches  
1252 any of the values selected by the second argument. This match compares the target namespace of  
1253 the requested document with the value of "urn:example:med:schemas:record".

1254 [a233] The `MatchId` attribute names the matching function.

1255 [a235] Literal attribute value to match.

1256 [a237] - [a239] The `<ResourceAttributeDesignator>` element selects the target namespace  
1257 from the resource contained in the request **context**. The **attribute** name is specified by the  
1258 `AttributeId`.

1259 [a241] - [a247] The second `<ResourceMatch>` element. This match compares the results of two  
1260 XPath expressions. The second XPath expression is the location path to the requested XML  
1261 element and the first XPath expression is the literal value `"/md:record"`. The "xpath-node-match"  
1262 function evaluates to "True" if the requested XML element is below the `"/md:record"` element.

1263 [a250] - [a262] The `<Actions>` element contains a **disjunctive sequence** of `<Action>` elements.  
1264 In this case, there is just one `<Action>` element.

1265 [a251] - [a261] The `<Action>` element contains a **conjunctive sequence** of `<ActionMatch>`  
1266 elements. In this case, there is just one `<ActionMatch>` element.

1267 [a252] - [a260] The <ActionMatch> element compares its first and second child elements  
1268 according to the matching function. The match is positive if the value of the first argument matches  
1269 any of the values selected by the second argument. In this case, the value of the action-id  
1270 action attribute in the request **context** is compared with the literal value "read".

1271 [a264] - [a266] The <Condition> element. A **condition** must evaluate to "True" for the **rule** to be  
1272 applicable. This **condition** contains a reference to a variable definition defined elsewhere in the  
1273 **policy**.

#### 1274 4.2.4.2. Rule 2

1275 Rule 2 illustrates the use of a mathematical function, i.e. the <Apply> element with functionId  
1276 "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" to calculate the date of the  
1277 patient's sixteenth birthday. It also illustrates the use of **predicate** expressions, with the  
1278 functionId "urn:oasis:names:tc:xacml:1.0:function:and". This example has one function  
1279 embedded in the <Condition> element and another one referenced in a  
1280 <VariableDefinition> element.

```
1281 [a269] <?xml version="1.0" encoding="UTF-8" ?>
1282 [a270] <Policy
1283 [a271] xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd" xmlns:xacml-
1284 context="urn:oasis:names:tc:xacml:2.0:context:schema:cd"
1285 [a272] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1286 xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:cd
1287 http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-cd.xsd"
1288 [a273] xmlns:xf="http://www.w3.org/TR/2002/WD-xquery-operators-20020816/#"
1289 [a274] xmlns:md="http://www.med.example.com/schemas/record.xsd"
1290 [a275] PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:2"
1291 RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
1292 overrides">
1293 [a276] <PolicyDefaults>
1294 [a277] <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-
1295 19991116</XPathVersion>
1296 [a278] </PolicyDefaults>
1297 [a279] <Target/>
1298 [a280] <VariableDefinition VariableId="17590035">
1299 [a281] <Apply FunctionId="urn:oasis:names:tc:xacml:2.0:function:date-less-or-
1300 equal">
1301 [a282] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-
1302 only">
1303 [a283] <EnvironmentAttributeDesignator
1304 [a284] AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date"
1305 [a285] DataType="http://www.w3.org/2001/XMLSchema#date"/>
1306 [a286] </Apply>
1307 [a287] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-
1308 yearMonthDuration">
1309 [a288] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-
1310 only">
1311 [a289] <AttributeSelector RequestContextPath=
1312 [a290] "//md:record/md:patient/md:patientDoB/text()"
1313 [a291] DataType="http://www.w3.org/2001/XMLSchema#date"/>
1314 [a292] </Apply>
1315 [a293] <AttributeValue
1316 [a294] DataType="http://www.w3.org/TR/2002/WD-xquery-operators-
1317 20020816#yearMonthDuration">
1318 [a295] <xf:dt-yearMonthDuration>
1319 [a296] P16Y
1320 [a297] </xf:dt-yearMonthDuration>
1321 [a298] </AttributeValue>
1322 [a299] </Apply>
1323 [a300] </Apply>
```

```

1324 [a301] </VariableDefinition>
1325 [a302] <Rule
1326 [a303] RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:2"
1327 [a304] Effect="Permit">
1328 [a305] <Description>
1329 [a306]   A person may read any medical record in the
1330 [a307]   http://www.med.example.com/records.xsd namespace
1331 [a308]   for which he or she is the designated parent or guardian,
1332 [a309]   and for which the patient is under 16 years of age
1333 [a310] </Description>
1334 [a311] <Target>
1335 [a312] <Resources>
1336 [a313] <Resource>
1337 [a314] <ResourceMatch
1338 [a315]   MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1339 [a316] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1340 [a317]   http://www.med.example.com/schemas/record.xsd
1341 [a318] </AttributeValue>
1342 [a319] <ResourceAttributeDesignator AttributeId=
1343 "urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1344 [a320]   DataType="http://www.w3.org/2001/XMLSchema#string"/>
1345 [a321] </ResourceMatch>
1346 [a322] <ResourceMatch
1347 [a323]   MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1348 [a324] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1349 [a325]   /md:record
1350 [a326] </AttributeValue>
1351 [a327] <ResourceAttributeDesignator
1352 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
1353 [a328]   DataType="http://www.w3.org/2001/XMLSchema#string"/>
1354 [a329] </ResourceMatch>
1355 [a330] </Resource>
1356 [a331] </Resources>
1357 [a332] <Actions>
1358 [a333] <Action>
1359 [a334] <ActionMatch
1360 [a335]   MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1361 [a336] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1362 [a337]   read
1363 [a338] </AttributeValue>
1364 [a339] <ActionAttributeDesignator
1365 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1366 [a340]   DataType="http://www.w3.org/2001/XMLSchema#string"/>
1367 [a341] </ActionMatch>
1368 [a342] </Action>
1369 [a343] </Actions>
1370 [a344] </Target>
1371 [a345] <Condition>
1372 [a346] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
1373 [a347] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1374 [a348] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1375 and-only">
1376 [a349] <SubjectAttributeDesignator
1377 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:
1378 [a350] parent-guardian-id"
1379 [a351]   DataType="http://www.w3.org/2001/XMLSchema#string"/>
1380 [a352] </Apply>
1381 [a353] <Apply
1382 [a354]   FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-
1383 only">
1384 [a355] <AttributeSelector
1385 [a356]   RequestContextPath="//xacml-context:Resource/xacml-
1386 context:ResourceContent/md:record/md:parentGuardian/md:parentGuardianId/text()"

```

1387 [a357]        DataType="http://www.w3.org/2001/XMLSchema#string" />  
1388 [a358]        </Apply>  
1389 [a359]        </Apply>  
1390 [a360]        <VariableReference VariableId="17590035" />  
1391 [a361]        </Apply>  
1392 [a362]        </Condition>  
1393 [a363]        </Rule>  
1394 [a364] </Policy>

1395 [a280] - [a301] The <VariableDefinition> element contains part of the **condition** (i.e. is the  
1396 patient under 16 years of age?). The patient is under 16 years of age if the current date is less than  
1397 the date computed by adding 16 to the patient's date of birth.

1398 [a281] - [a300] "urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal" is used to compute the  
1399 difference of two date arguments.

1400 [a282] - [a286] The first date argument uses "urn:oasis:names:tc:xacml:1.0:function:date-one-and-  
1401 only" to ensure that the **bag** of values selected by its argument contains exactly one value of type  
1402 "http://www.w3.org/2001/XMLSchema#date".

1403 [a284] The current date is evaluated by selecting the  
1404 "urn:oasis:names:tc:xacml:1.0:environment:current-date" **environment attribute**.

1405 [a287] - [a299] The second date argument uses "urn:oasis:names:tc:xacml:1.0:function:date-add-  
1406 yearMonthDuration" to compute the date of the patient's sixteenth birthday by adding 16 years to  
1407 the patient's date of birth. The first of its arguments is of type  
1408 "http://www.w3.org/2001/XMLSchema#date" and the second is of type  
1409 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration".

1410 [a289] The <AttributeSelector> element selects the patient's date of birth by taking the XPath  
1411 expression over the **resource** content.

1412 [a293] - [a298] Year Month Duration of 16 years.

1413 [a311] - [a344] **Rule** declaration and **rule target**. See Rule 1 in Section 4.2.4.1 for the detailed  
1414 explanation of these elements.

1415 [a345] - [a362] The <Condition> element. The **condition** must evaluate to "True" for the **rule** to  
1416 be applicable. This **condition** evaluates the truth of the statement: the requestor is the designated  
1417 parent or guardian and the patient is under 16 years of age. It contains one embedded <Apply>  
1418 element and one referenced <VariableDefinition> element.

1419 [a346] The **condition** uses the "urn:oasis:names:tc:xacml:1.0:function:and" function. This is a  
1420 Boolean function that takes one or more Boolean arguments (2 in this case) and performs the  
1421 logical "AND" operation to compute the truth value of the expression.

1422 [a347] - [a359] The first part of the **condition** is evaluated (i.e. is the requestor the designated  
1423 parent or guardian?). The function is "urn:oasis:names:tc:xacml:1.0:function:string-equal" and it  
1424 takes two arguments of type "http://www.w3.org/2001/XMLSchema#string".

1425 [a348] designates the first argument. Since "urn:oasis:names:tc:xacml:1.0:function:string-equal"  
1426 takes arguments of type "http://www.w3.org/2001/XMLSchema#string",  
1427 "urn:oasis:names:tc:xacml:1.0:function:string-one-and-only" is used to ensure that the **subject**  
1428 **attribute** "urn:oasis:names:tc:xacml:2.0:example:attribute:parent-guardian-id" in the request  
1429 **context** contains exactly one value.

1430 [a353] designates the second argument. The value of the **subject attribute**  
1431 "urn:oasis:names:tc:xacml:2.0:example:attribute:parent-guardian-id" is selected from the request  
1432 **context** using the <SubjectAttributeDesignator> element.

1433 [a354] As above, the “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used to ensure  
 1434 that the **bag** of values selected by it’s argument contains exactly one value of type  
 1435 “http://www.w3.org/2001/XMLSchema#string”.

1436 [a355] The second argument selects the value of the <md:parentGuardianId> element from the  
 1437 **resource** content using the <AttributeSelector> element. This element contains a free-form  
 1438 XPath expression, pointing into the request **context**. Note that all namespace prefixes in the XPath  
 1439 expression are resolved with standard namespace declarations. The AttributeSelector  
 1440 evaluates to the **bag** of values of type “http://www.w3.org/2001/XMLSchema#string”.

1441 [a360] references the <VariableDefinition> element, where the second part of the **condition**  
 1442 is defined.

#### 1443 4.2.4.3. Rule 3

1444 Rule 3 illustrates the use of an **obligation**. The XACML <Rule> element syntax does not include  
 1445 an element suitable for carrying an **obligation**, therefore Rule 3 has to be formatted as a  
 1446 <Policy> element.

```

1447 [a365] <?xml version="1.0" encoding="UTF-8"?>
1448 [a366] <Policy
1449 [a367]   xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd" xmlns:xacml-
1450 [a368]   context="urn:oasis:names:tc:xacml:2.0:context:schema:cd"
1451 [a369]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1452 [a369]   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:cd
1453 http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-cd.xsd"
1454 [a370]   xmlns:md="http://www.med.example.com/schemas/record.xsd"
1455 [a371]   PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:3"
1456 [a372]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1457 algorithm:deny-overrides">
1458 [a373]   <Description>
1459 [a374]     Policy for any medical record in the
1460 [a375]     http://www.med.example.com/schemas/record.xsd namespace
1461 [a376]   </Description>
1462 [a377]   <PolicyDefaults>
1463 [a378]     <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-
1464 19991116</XPathVersion>
1465 [a379]   </PolicyDefaults>
1466 [a380]   <Target>
1467 [a381]     <Resources>
1468 [a382]       <Resource>
1469 [a383]         <ResourceMatch
1470 [a384]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1471 [a385]             <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1472 [a386]               urn:example:med:schemas:record
1473 [a387]             </AttributeValue>
1474 [a388]             <ResourceAttributeDesignator AttributeId=
1475 [a389]               "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1476 [a390]               DataType="http://www.w3.org/2001/XMLSchema#string" />
1477 [a391]             </ResourceMatch>
1478 [a392]           </Resource>
1479 [a393]         </Resources>
1480 [a394]     </Target>
1481 [a395]   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:3"
1482 [a396]     Effect="Permit">
1483 [a397]     <Description>
1484 [a398]       A physician may write any medical element in a record
1485 [a399]       for which he or she is the designated primary care
1486 [a400]       physician, provided an email is sent to the patient
1487 [a401]     </Description>
1488 [a402]   </Rule>
  
```



```

1489 [a403] <Subjects>
1490 [a404] <Subject>
1491 [a405] <SubjectMatch
1492 [a406] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1493 [a407] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1494 [a408] physician
1495 [a409] </AttributeValue>
1496 [a410] <SubjectAttributeDesignator AttributeId=
1497 "urn:oasis:names:tc:xacml:2.0:example:attribute:role"
1498 [a411] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1499 [a412] </SubjectMatch>
1500 [a413] </Subject>
1501 [a414] </Subjects>
1502 [a415] <Resources>
1503 [a416] <Resource>
1504 [a417] <ResourceMatch
1505 [a418] MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1506 [a419] <AttributeValue
1507 [a420] DataType="http://www.w3.org/2001/XMLSchema#string">
1508 [a421] /md:record/md:medical
1509 [a422] </AttributeValue>
1510 [a423] <ResourceAttributeDesignator
1511 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
1512 [a424] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1513 [a425] </ResourceMatch>
1514 [a426] </Resource>
1515 [a427] </Resources>
1516 [a428] <Actions>
1517 [a429] <Action>
1518 [a430] <ActionMatch
1519 [a431] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1520 [a432] <AttributeValue
1521 [a433] DataType="http://www.w3.org/2001/XMLSchema#string">
1522 [a434] write
1523 [a435] </AttributeValue>
1524 [a436] <ActionAttributeDesignator
1525 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1526 [a437] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1527 [a438] </ActionMatch>
1528 [a439] </Action>
1529 [a440] </Actions>
1530 [a441] </Target>
1531 [a442] <Condition>
1532 [a443] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1533 [a444] <Apply
1534 [a445] FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1535 [a446] <SubjectAttributeDesignator
1536 [a447] AttributeId="urn:oasis:names:tc:xacml:2.0:example:
1537 attribute:physician-id"
1538 [a448] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1539 [a449] </Apply>
1540 [a450] <Apply
1541 [a451] FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1542 [a452] <AttributeSelector RequestContextPath=
1543 [a453] "//xacml-context:Resource/xacml-
1544 context:ResourceContent/md:record/md:primaryCarePhysician/md:registrationID/text(
1545 )"
1546 [a454] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1547 [a455] </Apply>
1548 [a456] </Apply>
1549 [a457] </Condition>
1550 [a458] </Rule>
1551 [a459] <Obligations>

```

```

1552 [a460] <Obligation
1553 ObligationId="urn:oasis:names:tc:xacml:example:obligation:email"
1554 [a461] FulfillOn="Permit">
1555 [a462] <AttributeAssignment
1556 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:mailto"
1557 [a463] DataType="http://www.w3.org/2001/XMLSchema#string">
1558 [a464] &lt;AttributeSelector RequestContextPath=
1559 [a465] "/md:/record/md:patient/md:patientContact/md:email"
1560 [a466] DataType="http://www.w3.org/2001/XMLSchema#string"/ &gt; ;
1561 [a467] </AttributeAssignment>
1562 [a468] <AttributeAssignment
1563 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
1564 [a469] DataType="http://www.w3.org/2001/XMLSchema#string">
1565 [a470] Your medical record has been accessed by:
1566 [a471] </AttributeAssignment>
1567 [a472] <AttributeAssignment
1568 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
1569 [a473] DataType="http://www.w3.org/2001/XMLSchema#string">
1570 [a474] &lt;SubjectAttributeDesignator
1571 AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
1572 [a475] DataType="http://www.w3.org/2001/XMLSchema#string"/ &gt; ;
1573 [a476] </AttributeAssignment>
1574 [a477] </Obligation>
1575 [a478] </Obligations>
1576 [a479] </Policy>

```

1577 [a366] - [a372] The <Policy> element includes standard namespace declarations as well as policy  
1578 specific parameters, such as PolicyId and RuleCombiningAlgId.

1579 [a371] **Policy** identifier. This parameter allows the **policy** to be referenced by a **policy set**.

1580 [a372] The **Rule combining algorithm** identifies the algorithm for combining the outcomes of **rule**  
1581 evaluation.

1582 [a373] - [a376] Free-form description of the **policy**.

1583 [a379] - [a394] **Policy target**. The **policy target** defines a set of applicable decision requests. The  
1584 structure of the <Target> element in the <Policy> is identical to the structure of the <Target>  
1585 element in the <Rule>. In this case, the **policy target** is the set of all XML resources that conform  
1586 to the namespace "urn:example:med:schemas:record".

1587 [a395] The only <Rule> element included in this <Policy>. Two parameters are specified in the  
1588 **rule** header: RuleId and Effect.

1589 [a402] - [a441] The **rule target** further constrains the **policy target**.

1590 [a405] - [a412] The <SubjectMatch> element targets the **rule** at **subjects** whose  
1591 "urn:oasis:names:tc:xacml:2.0:example:attribute:role" **subject attribute** is equal to "physician".

1592 [a417] - [a425] The <ResourceMatch> element targets the **rule** at **resources** that match the  
1593 XPath expression "/md:record/md:medical".

1594 [a430] - [a438] The <ActionMatch> element targets the **rule** at **actions** whose  
1595 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** is equal to "write".

1596 [a442] - [a457] The <Condition> element. For the **rule** to be applicable to the **decision request**,  
1597 the **condition** must evaluate to "True". This **condition** compares the value of the  
1598 "urn:oasis:names:tc:xacml:2.0:example:attribute:physician-id" **subject attribute** with the value of  
1599 the <registrationId> element in the medical record that is being accessed.

1600 [a459] - [a478] The <Obligations> element. **Obligations** are a set of operations that must be  
1601 performed by the **PEP** in conjunction with an **authorization decision**. An **obligation** may be

1602 associated with a "Permit" or "Deny" **authorization decision**. The element contains a single  
1603 **obligation**.

1604 [a460] - [a477] The <Obligation> element consists of the ObligationId attribute, the  
1605 **authorization decision** value for which it must be fulfilled, and a set of **attribute** assignments. The  
1606 **PDP** does not resolve the attribute assignments. This is the job of the **PEP**.

1607 [a460] The ObligationId attribute identifies the **obligation**. In this case, the **PEP** is required to  
1608 send email.

1609 [a461] The FulfillOn attribute defines the **authorization decision** value for which this  
1610 **obligation** must be fulfilled. In this case, when access is permitted.

1611 [a462] - [a467] The first parameter indicates where the **PEP** will find the email address in the  
1612 resource.

1613 [a468] - [a471] The second parameter contains literal text for the email body.

1614 [a472] - [a476] The third parameter indicates where the **PEP** will find further text for the email body  
1615 in the resource.

#### 1616 4.2.4.4. Rule 4

1617 Rule 4 illustrates the use of the "Deny" Effect value, and a <Rule> with no <Condition>  
1618 element.

```

1619 [a480] <?xml version="1.0" encoding="UTF-8"?>
1620 [a481] <Policy
1621 [a482] xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd"
1622 [a483] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1623 xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:cd
1624 http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-cd.xsd"
1625 [a484] xmlns:md="http://www.med.example.com/schemas/record.xsd"
1626 [a485] PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:4"
1627 [a486] RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1628 algorithm:deny-overrides">
1629 [a487] <PolicyDefaults>
1630 [a488] <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-
1631 19991116</XPathVersion>
1632 [a489] </PolicyDefaults>
1633 [a490] <Target/>
1634 [a491] <Rule
1635 [a492] RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:4"
1636 [a493] Effect="Deny">
1637 [a494] <Description>
1638 [a495] An Administrator shall not be permitted to read or write
1639 [a496] medical elements of a patient record in the
1640 [a497] http://www.med.example.com/records.xsd namespace.
1641 [a498] </Description>
1642 [a499] <Target>
1643 [a500] <Subjects>
1644 [a501] <Subject>
1645 [a502] <SubjectMatch
1646 [a503] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1647 [a504] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1648 [a505] administrator
1649 [a506] </AttributeValue>
1650 [a507] <SubjectAttributeDesignator AttributeId=
1651 [a508] "urn:oasis:names:tc:xacml:2.0:example:attribute:role"
1652 [a509] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1653 [a510] </SubjectMatch>
1654 [a511] </Subject>

```



```

1655 [a512] </Subjects>
1656 [a513] <Resources>
1657 [a514] <Resource>
1658 [a515] <ResourceMatch
1659 [a516] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1660 [a517] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1661 [a518] urn:example:med:schemas:record
1662 [a519] </AttributeValue>
1663 [a520] <ResourceAttributeDesignator
1664 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1665 [a521] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1666 [a522] </ResourceMatch>
1667 [a523] <ResourceMatch
1668 [a524] MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1669 [a525] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1670 [a526] /md:record/md:medical
1671 [a527] </AttributeValue>
1672 [a528] <ResourceAttributeDesignator
1673 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
1674 [a529] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1675 [a530] </ResourceMatch>
1676 [a531] </Resource>
1677 [a532] </Resources>
1678 [a533] <Actions>
1679 [a534] <Action>
1680 [a535] <ActionMatch
1681 [a536] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1682 [a537] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1683 [a538] read
1684 [a539] </AttributeValue>
1685 [a540] <ActionAttributeDesignator
1686 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1687 [a541] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1688 [a542] </ActionMatch>
1689 [a543] </Action>
1690 [a544] <Action>
1691 [a545] <ActionMatch
1692 [a546] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1693 [a547] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
1694 [a548] write
1695 [a549] </AttributeValue>
1696 [a550] <ActionAttributeDesignator
1697 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1698 [a551] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1699 [a552] </ActionMatch>
1700 [a553] </Action>
1701 [a554] </Actions>
1702 [a555] </Target>
1703 [a556] </Rule>
1704 [a557] </Policy>

```

1705 [a492] - [a493] The <Rule> element declaration.

1706 [a493] **Rule** Effect. Every **rule** that evaluates to “True” emits the **rule effect** as its value. This  
1707 **rule** Effect is “Deny” meaning that according to this **rule**, access must be denied when it  
1708 evaluates to “True”.

1709 [a494] - [a498] Free form description of the **rule**.

1710 [a499] - [a555] **Rule target**. The **Rule target** defines the set of **decision requests** that are  
1711 applicable to the **rule**.

1712 [a502] - [a510] The <SubjectMatch> element targets the **rule** at **subjects** whose  
 1713 "urn:oasis:names:tc:xacml:2.0:example:attribute:role" **subject attribute** is equal to  
 1714 "administrator".

1715 [a513] - [a532] The <Resources> element contains one <Resource> element, which (in turn)  
 1716 contains two <ResourceMatch> elements. The **target** matches if the **resource** identified by the  
 1717 request **context** matches both **resource** match criteria.

1718 [a558] [a515]-[a522] The first <ResourceMatch> element targets the **rule** at  
 1719 **resources** whose "urn:oasis:names:tc:xacml:2.0:resource:target-namespace" **resource**  
 1720 **attribute** is equal to "urn:example:med:schemas:record".

1721 [a523] - [a530] The second <ResourceMatch> element targets the **rule** at XML elements that  
 1722 match the XPath expression "/md:record/md:medical".

1723 [a533] - [a554] The <Actions> element contains two <Action> elements, each of which contains  
 1724 one <ActionMatch> element. The **target** matches if the **action** identified in the request **context**  
 1725 matches either of the **action** match criteria.

1726 [a535] - [a552] The <ActionMatch> elements target the **rule** at **actions** whose  
 1727 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** is equal to "read" or "write".

1728 This **rule** does not have a <Condition> element.

#### 1729 **4.2.4.5. Example PolicySet**

1730 This section uses the examples of the previous sections to illustrate the process of combining  
 1731 **policies**. The policy governing read access to medical elements of a record is formed from each of  
 1732 the four **rules** described in Section 4.2.3. In plain language, the combined rule is:

- 1733 • Either the requestor is the patient; or
- 1734 • the requestor is the parent or guardian and the patient is under 16; or
- 1735 • the requestor is the primary care physician and a notification is sent to the patient; and
- 1736 • the requestor is not an administrator.

1737 The following **policy set** illustrates the combined **policies**. **Policy 3** is included by reference and  
 1738 **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>

```

1739

1740 [a560] - [a566] The <PolicySet> element declaration. Standard XML namespace declarations  
1741 are included.

1742 [a563] The PolicySetId attribute is used for identifying this **policy set** for possible inclusion in  
1743 another **policy set**.

1744 [a565]The **policy combining algorithm** identifier. **Policies** and **policy sets** in this **policy set** are  
1745 combined according to the specified **policy combining algorithm** when the **authorization**  
1746 **decision** is computed.

1747 [a567] - [a569] Free form description of the **policy set**.

1748 [a570] - [a584] The **policy set** <Target> element defines the set of **decision requests** that are  
1749 applicable to this <PolicySet> element.

1750 [a585] PolicyIdReference includes a **policy** by id.

1751 [a589] Policy 2 is explicitly included in this **policy set**. The **rules** in Policy 2 are omitted for  
1752 clarity.

---

1753 **5. Policy syntax (normative, with the exception of**  
1754 **the schema fragments)**

1755 **5.1. Element <PolicySet>**

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

1759 <PolicySetIdReference> element. **Policies** MAY be included in an enclosing <PolicySet>  
1760 element either directly using the <Policy> element or indirectly using the  
1761 <PolicyIdReference> element.

1762 A <PolicySet> element MAY be evaluated, in which case the evaluation procedure defined in  
1763 Section 7.11 SHALL be used.

1764 If a <PolicySet> element contains references to other **policy sets** or **policies** in the form of  
1765 URLs, then these references MAY be resolvable.

1766 **Policy sets** and **policies** included in a <PolicySet> element MUST be combined using the  
1767 algorithm identified by the PolicyCombiningAlgId attribute. <PolicySet> is treated exactly  
1768 like a <Policy> in all **policy combining algorithms**.

1769 The <Target> element defines the applicability of the <PolicySet> element to a set of **decision**  
1770 **requests**. If the <Target> element within the <PolicySet> element matches the **request**  
1771 **context**, then the <PolicySet> element MAY be used by the **PDP** in making its **authorization**  
1772 **decision**. See Section 7.11.

1773 The <Obligations> element contains a set of **obligations** that MUST be fulfilled by the **PEP** in  
1774 conjunction with the **authorization decision**. If the **PEP** does not understand, or cannot fulfill, any  
1775 of the **obligations**, then it MUST act as if the **PDP** had returned a "Deny" **authorization decision**  
1776 value. See Section 7.14.

```
1777 <xs:element name="PolicySet" type="xacml:PolicySetType"/>  
1778 <xs:complexType name="PolicySetType">  
1779   <xs:sequence>  
1780     <xs:element ref="xacml:Description" minOccurs="0"/>  
1781     <xs:element ref="xacml:PolicySetDefaults" minOccurs="0"/>  
1782     <xs:element ref="xacml:Target"/>  
1783     <xs:choice minOccurs="0" maxOccurs="unbounded">  
1784       <xs:element ref="xacml:PolicySet"/>  
1785       <xs:element ref="xacml:Policy"/>  
1786       <xs:element ref="xacml:PolicySetIdReference"/>  
1787       <xs:element ref="xacml:PolicyIdReference"/>  
1788       <xs:element ref="xacml:CombinerParameters"/>  
1789       <xs:element ref="xacml:PolicyCombinerParameters"/>  
1790       <xs:element ref="xacml:PolicySetCombinerParameters"/>  
1791     </xs:choice>  
1792     <xs:element ref="xacml:Obligations" minOccurs="0"/>  
1793   </xs:sequence>  
1794   <xs:attribute name="PolicySetId" type="xs:anyURI" use="required"/>  
1795   <xs:attribute name="Version" type="xacml:VersionType" default="1.0"/>  
1796   <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI" use="required"/>  
1797 </xs:complexType>
```

1798 The <PolicySet> element is of **PolicySetType** complex type.

1799 The <PolicySet> element contains the following attributes and elements:

1800 PolicySetId [Required]

1801 **Policy set** identifier. It is the responsibility of the **PAP** to ensure that no two **policies**  
1802 visible to the **PDP** have the same identifier. This MAY be achieved by following a  
1803 predefined URN or URI scheme. If the **policy set** identifier is in the form of a URL, then it  
1804 MAY be resolvable.

1805 Version [Default 1.0]

1806 The version number of the **PolicySet**.

1807 PolicyCombiningAlgId [Required]

1808 The identifier of the **policy-combining algorithm** by which the <PolicySet>,  
1809 <CombinerParameters>, <PolicyCombinerParameters> and  
1810 <PolicySetCombinerParameters> components MUST be combined. Standard  
1811 **policy-combining algorithms** are listed in Appendix C. Standard **policy-combining**  
1812 **algorithm** identifiers are listed in Section B.10.

1813 <Description> [Optional]  
1814 A free-form description of the **policy set**.

1815 <PolicySetDefaults> [Optional]  
1816 A set of default values applicable to the **policy set**. The scope of the  
1817 <PolicySetDefaults> element SHALL be the enclosing **policy set**.

1818 <Target> [Required]  
1819 The <Target> element defines the applicability of a **policy set** to a set of **decision**  
1820 **requests**.  
1821 The <Target> element MAY be declared by the creator of the <PolicySet> or it MAY be  
1822 computed from the <Target> elements of the referenced <Policy> elements, either as  
1823 an intersection or as a union.

1824 <PolicySet> [Any Number]  
1825 A **policy set** that is included in this **policy set**.

1826 <Policy> [Any Number]  
1827 A **policy** that is included in this **policy set**.

1828 <PolicySetIdReference> [Any Number]  
1829 A reference to a **policy set** that MUST be included in this **policy set**. If  
1830 <PolicySetIdReference> is a URL, then it MAY be resolvable.

1831 <PolicyIdReference> [Any Number]  
1832 A reference to a **policy** that MUST be included in this **policy set**. If the  
1833 <PolicyIdReference> is a URL, then it MAY be resolvable.

1834 <Obligations> [Optional]  
1835 Contains the set of <Obligation> elements. See Section 7.14 for a description of how  
1836 the set of **obligations** to be returned by the **PDP** shall be determined.

1837 <CombinerParameters> [Optional]  
1838 Contains a sequence of <CombinerParameter> elements.

1839 <PolicyCombinerParameters> [Optional]  
1840 Contains a sequence of <CombinerParameter> elements that are associated with a  
1841 particular <Policy> or <PolicyIdReference> element within the <PolicySet>.

1842 <PolicySetCombinerParameters> [Optional]  
1843 Contains a sequence of <CombinerParameter> elements that are associated with a  
1844 particular <PolicySet> or <PolicySetIdReference> element within the  
1845 <PolicySet>.

## 1846 **5.2. Element <Description>**

1847 The <Description> element contains a free-form description of the <PolicySet>, <Policy>  
1848 or <Rule> element. The <Description> element is of **xs:string** simple type.

```
1849 <xs:element name="Description" type="xs:string"/>
```

## 1850 **5.3. Element <PolicySetDefaults>**

1851 The <PolicySetDefaults> element SHALL specify default values that apply to the  
1852 <PolicySet> element.

```
1853 <xs:element name="PolicySetDefaults" type="xacml:DefaultsType"/>  
1854 <xs:complexType name="DefaultsType">  
1855 <xs:sequence>  
1856 <xs:choice>  
1857 <xs:element ref="xacml:XPathVersion" minOccurs="0"/>  
1858 </xs:choice>  
1859 </xs:sequence>  
1860 </xs:complexType>
```

1861 <PolicySetDefaults> element is of **DefaultsType** complex type.

1862 The <PolicySetDefaults> element contains the following elements:

1863 <XPathVersion> [Optional]

1864 Default XPath version.

## 1865 **5.4. Element <XPathVersion>**

1866 The <XPathVersion> element SHALL specify the version of the XPath specification to be used by  
1867 <AttributeSelector> elements and XPath-based functions in the **policy set** or **policy**.

```
1868 <xs:element name="XPathVersion" type="xs:anyURI"/>
```

1869 The URI for the XPath 1.0 specification is "http://www.w3.org/TR/1999/Rec-xpath-19991116". The  
1870 <XPathVersion> element is REQUIRED if the XACML enclosing **policy set** or **policy** contains  
1871 <AttributeSelector> elements or XPath-based functions.

## 1872 **5.5. Element <Target>**

1873 The <Target> element identifies the set of **decision requests** that the parent element is intended  
1874 to evaluate. The <Target> element SHALL appear as a child of a <PolicySet> and <Policy>  
1875 element and MAY appear as a child of a <Rule> element. It contains definitions for **subjects**,  
1876 **resources**, **actions** and **environments**.

1877 The <Target> element SHALL contain a **conjunctive sequence** of <Subjects>, <Resources>  
1878 <Actions> and <Environments> elements. For the parent of the <Target> element to be  
1879 applicable to the **decision request**, there MUST be at least one positive match between each  
1880 section of the <Target> element and the corresponding section of the <xacml-  
1881 context:Request> element.

```
1882 <xs:element name="Target" type="xacml:TargetType"/>  
1883 <xs:complexType name="TargetType">  
1884 <xs:sequence>  
1885 <xs:element ref="xacml:Subjects" minOccurs="0"/>  
1886 <xs:element ref="xacml:Resources" minOccurs="0"/>  
1887 <xs:element ref="xacml:Actions" minOccurs="0"/>  
1888 <xs:element ref="xacml:Environments" minOccurs="0"/>
```



1889 `</xs:sequence>`  
 1890 `</xs:complexType>`

1891 The `<Target>` element is of **TargetType** complex type.

1892 The `<Target>` element contains the following elements:

1893 `<Subjects>` [Optional]

1894 Matching specification for the **subject attributes** in the **context**. If this element is missing,  
 1895 then the **target** SHALL match all **subjects**.

1896 `<Resources>` [Optional]

1897 Matching specification for the **resource attributes** in the **context**. If this element is  
 1898 missing, then the **target** SHALL match all **resources**.

1899 `<Actions>` [Optional]

1900 Matching specification for the **action attributes** in the **context**. If this element is missing,  
 1901 then the **target** SHALL match all **actions**.

1902 `<Environments>` [Optional]

1903 Matching specification for the **environment attributes** in the **context**. If this element is  
 1904 missing, then the **target** SHALL match all **environments**.

## 5.6. Element `<Subjects>`

1905

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

```
1907 <xs:element name="Subjects" type="xacml:SubjectsType" />
1908 <xs:complexType name="SubjectsType">
1909   <xs:sequence>
1910     <xs:element ref="xacml:Subject" maxOccurs="unbounded" />
1911   </xs:sequence>
1912 </xs:complexType>
```

1913 The `<Subjects>` element is of **SubjectsType** complex type.

1914 The `<Subjects>` element contains the following elements:

1915 `<Subject>` [One to Many, Required]

1916 See Section 5.7.

## 5.7. Element `<Subject>`

1917

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

```
1920 <xs:element name="Subject" type="xacml:SubjectType" />
1921 <xs:complexType name="SubjectType">
1922   <xs:sequence>
1923     <xs:element ref="xacml:SubjectMatch" maxOccurs="unbounded" />
1924   </xs:sequence>
1925 </xs:complexType>
```

1926 The `<Subject>` element is of **SubjectType** complex type.

1927 The `<Subject>` element contains the following elements:

1928 `<SubjectMatch>` [One to Many]



1929 A **conjunctive sequence** of individual matches of the **subject attributes** in the request  
1930 **context** and the embedded **attribute** values. See Section 5.8.

## 1931 **5.8. Element <SubjectMatch>**

1932 The <SubjectMatch> element SHALL identify a set of **subject**-related entities by matching  
1933 **attribute** values in a <xacml-context:Subject> element of the request **context** with the  
1934 embedded **attribute** value.

```
1935 <xs:element name="SubjectMatch" type="xacml:SubjectMatchType"/>  
1936 <xs:complexType name="SubjectMatchType">  
1937   <xs:sequence>  
1938     <xs:element ref="xacml:AttributeValue"/>  
1939     <xs:choice>  
1940       <xs:element ref="xacml:SubjectAttributeDesignator"/>  
1941       <xs:element ref="xacml:AttributeSelector"/>  
1942     </xs:choice>  
1943   </xs:sequence>  
1944   <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>  
1945 </xs:complexType>
```

1946 The <SubjectMatch> element is of **SubjectMatchType** complex type.

1947 The <SubjectMatch> element contains the following attributes and elements:

1948 MatchId [Required]

1949 Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI** with  
1950 legal values documented in Section 7.5.

1951 <xacml:AttributeValue> [Required]

1952 Embedded attribute value.

1953 <SubjectAttributeDesignator> [Required choice]

1954 MAY be used to identify one or more **attribute** values in a <Subject> element of the  
1955 request **context**.

1956 <AttributeSelector> [Required choice]

1957 MAY be used to identify one or more **attribute** values in the request **context**. The XPath  
1958 expression SHOULD resolve to an **attribute** in a <Subject> element of the request  
1959 **context**.

## 1960 **5.9. Element <Resources>**

1961 The <Resources> element SHALL contain a **disjunctive sequence** of <Resource> elements.

```
1962 <xs:element name="Resources" type="xacml:ResourcesType"/>  
1963 <xs:complexType name="ResourcesType">  
1964   <xs:sequence>  
1965     <xs:element ref="xacml:Resource" maxOccurs="unbounded"/>  
1966   </xs:sequence>  
1967 </xs:complexType>
```

1968 The <Resources> element is of **ResourcesType** complex type.

1969 The <Resources> element contains the following elements:

1970 <Resource> [One to Many, Required]

1971 See Section 5.10.

## 1972 **5.10. Element <Resource>**

1973 The <Resource> element SHALL contain a **conjunctive sequence** of <ResourceMatch>  
1974 elements.

```
1975 <xs:element name="Resource" type="xacml:ResourceType" />  
1976 <xs:complexType name="ResourceType">  
1977   <xs:sequence>  
1978     <xs:element ref="xacml:ResourceMatch" maxOccurs="unbounded" />  
1979   </xs:sequence>  
1980 </xs:complexType>
```

1981 The <Resource> element is of **ResourceType** complex type.

1982 The <Resource> element contains the following elements:

1983 <ResourceMatch> [One to Many]

1984 A **conjunctive sequence** of individual matches of the **resource attributes** in the request  
1985 **context** and the embedded **attribute** values. See Section 5.11.

## 1986 **5.11. Element <ResourceMatch>**

1987 The <ResourceMatch> element SHALL identify a set of **resource**-related entities by matching  
1988 **attribute** values in the <xacml-context:Resource> element of the request **context** with the  
1989 embedded **attribute** value.

```
1990 <xs:element name="ResourceMatch" type="xacml:ResourceMatchType" />  
1991 <xs:complexType name="ResourceMatchType">  
1992   <xs:sequence>  
1993     <xs:element ref="xacml:AttributeValue" />  
1994     <xs:choice>  
1995       <xs:element ref="xacml:ResourceAttributeDesignator" />  
1996       <xs:element ref="xacml:AttributeSelector" />  
1997     </xs:choice>  
1998   </xs:sequence>  
1999   <xs:attribute name="MatchId" type="xs:anyURI" use="required" />  
2000 </xs:complexType>
```

2001 The <ResourceMatch> element is of **ResourceMatchType** complex type.

2002 The <ResourceMatch> element contains the following attributes and elements:

2003 MatchId [Required]

2004 Specifies a matching function. Values of this attribute MUST be of type **xs:anyURI**, with  
2005 legal values documented in Section 7.5.

2006 <xacml:AttributeValue> [Required]

2007 Embedded attribute value.

2008 <ResourceAttributeDesignator> [Required Choice]

2009 MAY be used to identify one or more **attribute** values in the <Resource> element of the  
2010 request **context**.

2011 <AttributeSelector> [Required Choice]

2012 MAY be used to identify one or more **attribute** values in the request **context**. The XPath  
2013 expression SHOULD resolve to an **attribute** in the <Resource> element of the request  
2014 **context**.

## 2015 **5.12. Element <Actions>**

2016 The <Actions> element SHALL contain a **disjunctive sequence** of <Action> elements.

```
2017 <xs:element name="Actions" type="xacml:ActionTypes" />  
2018 <xs:complexType name="ActionTypes">  
2019   <xs:sequence>  
2020     <xs:element ref="xacml:Action" maxOccurs="unbounded" />  
2021   </xs:sequence>  
2022 </xs:complexType>
```

2023 The <Actions> element is of **ActionTypes** complex type.

2024 The <Actions> element contains the following elements:

2025 <Action> [One to Many, Required]

2026 See Section 5.13.

## 2027 **5.13. Element <Action>**

2028 The <Action> element SHALL contain a **conjunctive sequence** of <ActionMatch> elements.

```
2029 <xs:element name="Action" type="xacml:ActionType" />  
2030 <xs:complexType name="ActionType">  
2031   <xs:sequence>  
2032     <xs:element ref="xacml:ActionMatch" maxOccurs="unbounded" />  
2033   </xs:sequence>  
2034 </xs:complexType>
```

2035 The <Action> element is of **ActionType** complex type.

2036 The <Action> element contains the following elements:

2037 <ActionMatch> [One to Many]

2038 A **conjunctive sequence** of individual matches of the **action attributes** in the request  
2039 **context** and the embedded **attribute** values. See Section 5.14.

## 2040 **5.14. Element <ActionMatch>**

2041 The <ActionMatch> element SHALL identify a set of **action**-related entities by matching **attribute**  
2042 values in the <xacml-context:Action> element of the request **context** with the embedded  
2043 **attribute** value.

```
2044 <xs:element name="ActionMatch" type="xacml:ActionMatchType" />  
2045 <xs:complexType name="ActionMatchType">  
2046   <xs:sequence>  
2047     <xs:element ref="xacml:AttributeValue" />  
2048     <xs:choice>  
2049       <xs:element ref="xacml:ActionAttributeDesignator" />  
2050       <xs:element ref="xacml:AttributeSelector" />  
2051     </xs:choice>  
2052   </xs:sequence>  
2053   <xs:attribute name="MatchId" type="xs:anyURI" use="required" />  
2054 </xs:complexType>
```

2055 The <ActionMatch> element is of **ActionMatchType** complex type.

2056 The <ActionMatch> element contains the following attributes and elements:

2057 MatchId [Required]

2058        Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI**, with  
2059        legal values documented in Section 7.5.

2060 <xacml:AttributeValue> [Required]

2061        Embedded attribute value.

2062 <ActionAttributeDesignator> [Required Choice]

2063        MAY be used to identify one or more **attribute** values in the <Action> element of the  
2064        request **context**.

2065 <AttributeSelector> [Required Choice]

2066        MAY be used to identify one or more **attribute** values in the request **context**. The XPath  
2067        expression SHOULD resolve to an **attribute** in the <Action> element of the **context**.

## 2068        **5.15. Element <Environments>**

2069 The <Environments> element SHALL contain a **disjunctive sequence** of <Environment>  
2070 elements.

```
2071 <xs:element name="Environments" type="xacml:EnvironmentsType" />
2072 <xs:complexType name="EnvironmentsType">
2073   <xs:sequence>
2074     <xs:element ref="xacml:Environment" maxOccurs="unbounded" />
2075   </xs:sequence>
2076 </xs:complexType>
```

2077 The <Environments> element is of **EnvironmentsType** complex type.

2078 The <Environments> element contains the following elements:

2079 <Environment> [One to Many, Required]

2080        See Section 5.16.

## 2081        **5.16. Element <Environment>**

2082 The <Environment> element SHALL contain a **conjunctive sequence** of  
2083 <EnvironmentMatch> elements.

```
2084 <xs:element name="Environment" type="xacml:EnvironmentType" />
2085 <xs:complexType name="EnvironmentType">
2086   <xs:sequence>
2087     <xs:element ref="xacml:EnvironmentMatch" maxOccurs="unbounded" />
2088   </xs:sequence>
2089 </xs:complexType>
```

2090 The <Environment> element is of **EnvironmentType** complex type.

2091 The <Environment> element contains the following elements:

2092 <EnvironmentMatch> [One to Many]

2093 A **conjunctive sequence** of individual matches of the **environment** attributes in the  
2094 request **context** and the embedded **attribute** values.

## 2095 5.17. Element <EnvironmentMatch>

2096 The <EnvironmentMatch> element SHALL identify an environment by matching **attribute** values  
2097 in the <xacml-context:Environment> element of the request **context** with the embedded  
2098 **attribute** value.

```
2099 <xs:element name="EnvironmentMatch" type="xacml:EnvironmentMatchType" />  
2100 <xs:complexType name="EnvironmentMatchType">  
2101   <xs:sequence>  
2102     <xs:element ref="xacml:AttributeValue" />  
2103     <xs:choice>  
2104       <xs:element ref="xacml:EnvironmentAttributeDesignator" />  
2105       <xs:element ref="xacml:AttributeSelector" />  
2106     </xs:choice>  
2107   </xs:sequence>  
2108   <xs:attribute name="MatchId" type="xs:anyURI" use="required" />  
2109 </xs:complexType>
```

2110 The <EnvironmentMatch> element is of **EnvironmentMatchType** complex type.

2111 The <EnvironmentMatch> element contains the following attributes and elements:

2112 MatchId [Required]

2113 Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI**, with  
2114 legal values documented in Section A.3.

2115 <xacml:AttributeValue> [Required]

2116 Embedded attribute value.

2117 <EnvironmentAttributeDesignator> [Required Choice]

2118 MAY be used to identify one or more **attribute** values in the <Environment> element of  
2119 the request **context**.

2120 <AttributeSelector> [Required Choice]

2121 MAY be used to identify one or more **attribute** values in the request **context**. The XPath  
2122 expression SHOULD resolve to an **attribute** in the <Environment> element of the  
2123 request **context**.

## 2124 5.18. Element <PolicySetIdReference>

2125 The <PolicySetIdReference> element SHALL be used to reference a <PolicySet> element  
2126 by id. If <PolicySetIdReference> is a URL, then it MAY be resolvable to the <PolicySet>  
2127 element. However, the mechanism for resolving a **policy set** reference to the corresponding  
2128 **policy set** is outside the scope of this specification.

```
2129 <xs:element name="PolicySetIdReference" type="xacml:IdReferenceType" />  
2130 <xs:complexType name="IdReferenceType">  
2131   <xs:simpleContent>  
2132     <xs:extension base="xs:anyURI" />  
2133     <xs:attribute name="xacml:Version" type="xacml:VersionMatchType"  
2134     use="optional" />  
2135     <xs:attribute name="xacml:EarliestVersion" type="xacml:VersionMatchType"  
2136     use="optional" />
```

```

2137     <xs:attribute name="xacml:LatestVersion" type="xacml:VersionMatchType"
2138 use="optional" />
2139     </xs:extension>
2140     </xs:simpleContent>
2141 </xs:complexType>

```

2142 Element <PolicySetIdReference> is of **xacml:IdReferenceType** complex type.

2143 **IdReferenceType** extends the **xs:anyURI** type with the following attributes:

2144 Version [Optional]

2145 Specifies a matching expression for the version of the **policy set** referenced.

2146 EarliestVersion [Optional]

2147 Specifies a matching expression for the earliest acceptable version of the **policy set**  
2148 referenced.

2149 LatestVersion [Optional]

2150 Specifies a matching expression for the latest acceptable version of the **policy set**  
2151 referenced.

2152 The matching operation is defined in Section 5.21. Any combination of these attributes MAY be  
2153 present in a <PolicySetIdReference>. The referenced **policy set** MUST match all  
2154 expressions. If none of these attributes is present, then any version of the **policy set** is acceptable.  
2155 In the case that more than one matching version can be obtained, then the most recent one  
2156 SHOULD be used.

## 2157 5.19. Element <PolicyIdReference>

2158 The <xacml:PolicyIdReference> element SHALL be used to reference a <Policy> element  
2159 by id. If <PolicyIdReference> is a URL, then it MAY be resolvable to the <Policy> element.  
2160 However, the mechanism for resolving a **policy** reference to the corresponding **policy** is outside  
2161 the scope of this specification.

```
2162 <xs:element name="PolicyIdReference" type="xacml:IdReferenceType" />
```

2163 Element <PolicyIdReference> is of **xacml:IdReferenceType** complex type (see Section 5.18) .

## 2164 5.20. Simple type VersionType

2165 Elements of this type SHALL contain the version number of the **policy** or **policy set**.

```

2166 <xs:simpleType name="VersionType">
2167   <xs:restriction base="xs:string">
2168     <xs:pattern value="(\d+\.)*\d+"/>
2169   </xs:restriction>
2170 </xs:simpleType>

```

2171 The version number is expressed as a sequence of decimal numbers, each separated by a period  
2172 (.). 'd+' represents a sequence of one or more decimal digits.

## 2173 5.21. Simple type VersionMatchType

2174 Elements of this type SHALL contain a restricted regular expression matching a version number  
2175 (see Section 5.20). The expression SHALL match versions of a referenced **policy** or **policy set**  
2176 that are acceptable for inclusion in the referencing **policy** or **policy set**.

```
2177 <xs:simpleType name="VersionMatchType">
```

```

2178 <xs:restriction base="xs:string">
2179   <xs:pattern value="((\d+|\*)\.)*(\d+|\*|\+)" />
2180 </xs:restriction>
2181 </xs:simpleType>

```

2182 A version match is '.'-separated, like a version string. A number represents a direct numeric match.  
 2183 A '\*' means that any single number is valid. A '+' means that any number, and any subsequent  
 2184 numbers, are valid. In this manner, the following four patterns would all match the version string  
 2185 '1.2.3': '1.2.3', '1.\*.3', '1.2.\*' and '1.+'.

## 5.22. Element <Policy>

2186 The <Policy> element is the smallest entity that SHALL be presented to the **PDP** for evaluation.

2188 A <Policy> element MAY be evaluated, in which case the evaluation procedure defined in  
 2189 Section 7.10 SHALL be used.

2190 The main components of this element are the <Target>, <Rule>, <CombinerParameters>,  
 2191 <RuleCombinerParameters> and <Obligations> elements and the RuleCombiningAlgId  
 2192 attribute.

2193 The <Target> element defines the applicability of the <Policy> element to a set of **decision**  
 2194 **requests**. If the <Target> element within the <Policy> element matches the **request context**,  
 2195 then the <Policy> element MAY be used by the **PDP** in making its **authorization decision**. See  
 2196 Section 7.10.

2197 The <Policy> element includes a sequence of choices between <VariableDefinition> and  
 2198 <Rule> elements.

2199 **Rules** included in the <Policy> element MUST be combined by the algorithm specified by the  
 2200 RuleCombiningAlgId attribute.

2201 The <Obligations> element contains a set of **obligations** that MUST be fulfilled by the **PEP** in  
 2202 conjunction with the **authorization decision**.

```

2203 <xs:element name="Policy" type="xacml:PolicyType" />
2204 <xs:complexType name="PolicyType">
2205   <xs:sequence>
2206     <xs:element ref="xacml:Description" minOccurs="0" />
2207     <xs:element ref="xacml:PolicyDefaults" minOccurs="0" />
2208     <xs:element ref="xacml:CombinerParameters" minOccurs="0" />
2209     <xs:element ref="xacml:Target" />
2210     <xs:choice maxOccurs="unbounded">
2211       <xs:element ref="xacml:CombinerParameters" minOccurs="0" />
2212       <xs:element ref="xacml:RuleCombinerParameters" minOccurs="0" />
2213       <xs:element ref="xacml:VariableDefinition" />
2214       <xs:element ref="xacml:Rule" />
2215     </xs:choice>
2216     <xs:element ref="xacml:Obligations" minOccurs="0" />
2217   </xs:sequence>
2218   <xs:attribute name="PolicyId" type="xs:anyURI" use="required" />
2219   <xs:attribute name="Version" type="xacml:VersionType" default="1.0" />
2220   <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required" />
2221 </xs:complexType>

```

2222 The <Policy> element is of **PolicyType** complex type.

2223 The <Policy> element contains the following attributes and elements:

2224 PolicyId [Required]



2225            **Policy** identifier. It is the responsibility of the **PAP** to ensure that no two **policies** visible to  
2226            the **PDP** have the same identifier. This MAY be achieved by following a predefined URN or  
2227            URI scheme. If the **policy** identifier is in the form of a URL, then it MAY be resolvable.

2228            Version [Default 1.0]

2229            The version number of the **Policy**.

2230            RuleCombiningAlgId [Required]

2231            The identifier of the **rule-combining algorithm** by which the <Policy>,  
2232            <CombinerParameters> and <RuleCombinerParameters> components MUST be  
2233            combined. Standard **rule-combining algorithms** are listed in Appendix C. Standard **rule-**  
2234            **combining algorithm** identifiers are listed in Section B.10.

2235            <Description> [Optional]

2236            A free-form description of the **policy**. See Section 5.2.

2237            <PolicyDefaults> [Optional]

2238            Defines a set of default values applicable to the **policy**. The scope of the  
2239            <PolicyDefaults> element SHALL be the enclosing **policy**.

2240            <CombinerParameters> [Optional]

2241            A sequence of parameters to be used by the **rule-combining algorithm**.

2242            <RuleCombinerParameters> [Optional]

2243            A sequence of parameters to be used by the **rule-combining algorithm**.

2244            <Target> [Required]

2245            The <Target> element defines the applicability of a <Policy> to a set of **decision requests**.

2246            The <Target> element MAY be declared by the creator of the <Policy> element, or it  
2247            MAY be computed from the <Target> elements of the referenced <Rule> elements either  
2248            as an intersection or as a union.

2249            <VariableDefinition> [Any Number]

2250            Common function definitions that can be referenced from anywhere in a **rule** where an  
2251            expression can be found.

2252            <Rule> [Any Number]

2253            A sequence of **rules** that MUST be combined according to the RuleCombiningAlgId  
2254            attribute. **Rules** whose <Target> elements match the **decision request** MUST be  
2255            considered. **Rules** whose <Target> elements do not match the **decision request** SHALL  
2256            be ignored.

2257            <Obligations> [Optional]

2258            A **conjunctive sequence** of **obligations** that MUST be fulfilled by the **PEP** in conjunction  
2259            with the **authorization decision**. See Section 7.14 for a description of how the set of  
2260            **obligations** to be returned by the **PDP** SHALL be determined.

2261

## 5.23. Element <PolicyDefaults>

2262 The <PolicyDefaults> element SHALL specify default values that apply to the <Policy>  
2263 element.

```
2264 <xs:element name="PolicyDefaults" type="xacml:DefaultsType"/>
2265 <xs:complexType name="DefaultsType">
2266   <xs:sequence>
2267     <xs:choice>
2268       <xs:element ref="xacml:XPathVersion" minOccurs="0"/>
2269     </xs:choice>
2270   </xs:sequence>
2271 </xs:complexType>
```

2272 <PolicyDefaults> element is of **DefaultsType** complex type.

2273 The <PolicyDefaults> element contains the following elements:

2274 <XPathVersion> [Optional]

2275         Default XPath version.

2276

## 5.24. Element <CombinerParameters>

2277 The <CombinerParameters> element conveys parameters for a *policy-* or *rule-combining*  
2278 *algorithm*.

2279 If multiple <CombinerParameters> elements occur within the same *policy* or *policy set*, they  
2280 SHALL be considered equal to one <CombinerParameters> element containing the  
2281 concatenation of all the sequences of <CombinerParameters> contained in all the aforementioned  
2282 <CombinerParameters> elements, such that the order of occurrence of the  
2283 <CominberParameters> elements is preserved in the concatenation of the  
2284 <CombinerParameter> elements.

2285 Note that none of the *combining algorithms* specified in XACML 2.0 is parameterized.

```
2286 <xs:element name="CombinerParameters" type="xacml:CombinerParametersType"/>
2287 <xs:complexType name="CombinerParametersType">
2288   <xs:sequence>
2289     <xs:element ref="xacml:CombinerParameter" minOccurs="0"
2290     maxOccurs="unbounded"/>
2291   </xs:sequence>
2292 </xs:complexType>
```

2293 The <CombinerParameters> element is of **CombinerParametersType** complex type.

2294 The <CombinerParameters> element contains the following elements:

2295 <CombinerParameter> [Any Number]

2296         A single parameter. See Section 5.25.

2297 Support for the <CombinerParameters> element is optional.

2298

## 5.25. Element <CombinerParameter>

2299 The <CombinerParameter> element conveys a single parameter for a *policy-* or *rule-*  
2300 *combining algorithm*.

```
2301 <xs:element name="CombinerParameter" type="xacml:CombinerParameterType"/>
2302 <xs:complexType name="CombinerParameterType">
```

```

2303     <xs:sequence>
2304         <xs:element ref="xacml:AttributeValue" />
2305     </xs:sequence>
2306     <xs:attribute name="ParameterName" type="xs:string" use="required" />
2307 </xs:complexType>

```

2308 The <CombinerParameter> element is of **CombinerParameterType** complex type.

2309 The <CombinerParameter> element contains the following attribute:

2310 ParameterName [Required]

2311         The identifier of the parameter.

2312 AttributeValue [Required]

2313         The value of the parameter.

2314 Support for the <CombinerParameter> element is optional.

## 2315         **5.26. Element <RuleCombinerParameters>**

2316 The <RuleCombinerParameters> element conveys *parameters* associated with a particular  
2317 *rule* within a *policy* for a *rule-combining algorithm*.

2318 Each <RuleCombinerParameters> element MUST be associated with a *rule* contained within  
2319 the same *policy*. If multiple <RuleCombinerParameters> elements reference the same *rule*,  
2320 they SHALL be considered equal to one <RuleCombinerParameters> element containing the  
2321 concatenation of all the sequences of <CombinerParameters> contained in all the aforementioned  
2322 <RuleCombinerParameters> elements, such that the order of occurrence of the  
2323 <RuleCominberParameters> elements is preserved in the concatenation of the  
2324 <CombinerParameter> elements.

2325 Note that none of the *rule-combining algorithms* specified in XACML 2.0 is parameterized.

```

2326 <xs:element name="RuleCombinerParameters"
2327 type="xacml:RuleCombinerParametersType" />
2328 <xs:complexType name="RuleCombinerParametersType">
2329     <xs:complexContent>
2330         <xs:extension base="xacml:CombinerParametersType">
2331             <xs:attribute name="RuleIdRef" type="xs:string" use="required" />
2332         </xs:extension>
2333     </xs:complexContent>
2334 </xs:complexType>

```

2335 The <RuleCombinerParameters> element contains the following elements:

2336 RuleIdRef [Required]

2337         The identifier of the <Rule> contained in the *policy*.

2338 Support for the <RuleCombinerParameters> element is optional, only if support for *combiner*  
2339 *parameters* is optional.

## 2340         **5.27. Element <PolicyCombinerParameters>**

2341 The <PolicyCombinerParameters> element conveys *parameters* associated with a particular  
2342 *policy* within a *policy set* for a *policy-combining algorithm*.

2343 Each <PolicyCombinerParameters> element MUST be associated with a **policy** contained  
2344 within the same **policy set**. If multiple <PolicyCombinerParameters> elements reference the  
2345 same **policy**, they SHALL be considered equal to one <PolicyCombinerParameters> element  
2346 containing the concatenation of all the sequences of <CombinerParameters> contained in all the  
2347 aforementioned <PolicyCombinerParameters> elements, such that the order of occurrence of  
2348 the <PolicyCominberParameters> elements is preserved in the concatenation of the  
2349 <CombinerParameter> elements.

2350 Note that none of the **policy-combining algorithms** specified in XACML 2.0 is parameterized.

```
2351 <xs:element name="PolicyCombinerParameters"  
2352 type="xacml:PolicyCombinerParametersType"/>  
2353 <xs:complexType name="PolicyCombinerParametersType">  
2354   <xs:complexContent>  
2355     <xs:extension base="xacml:CombinerParametersType">  
2356       <xs:attribute name="PolicyIdRef" type="xs:anyURI" use="required"/>  
2357     </xs:extension>  
2358   </xs:complexContent>  
2359 </xs:complexType>
```

2360 The <PolicyCombinerParameters> element is of **PolicyCombinerParametersType** complex  
2361 type.

2362 The <PolicyCombinerParameters> element contains the following elements:

2363 PolicyIdRef [Required]

2364         The identifier of a <Policy> or the value of a <PolicyIdReference> contained in the  
2365         **policy set**.

2366 Support for the <PolicyCombinerParameters> element is optional, only if support for  
2367 **combiner parameters** is optional.

## 2368         **5.28. Element <PolicySetCombinerParameters>**

2369 The <PolicySetCombinerParameters> element conveys **parameters** associated with a  
2370 particular **policy set** within a **policy set** for a **policy-combining algorithm**.

2371 Each <PolicySetCombinerParameters> element MUST be associated with a **policy set**  
2372 contained within the same **policy set**. If multiple <PolicySetCombinerParameters> elements  
2373 reference the same **policy set**, they SHALL be considered equal to one  
2374 <PolicySetCombinerParameters> element containing the concatenation of all the sequences  
2375 of <CombinerParameters> contained in all the aforementioned  
2376 <PolicySetCombinerParameters> elements, such that the order of occurrence of the  
2377 <PolicySetCominberParameters> elements is preserved in the concatenation of the  
2378 <CombinerParameter> elements.

2379 Note that none of the **policy-combining algorithms** specified in XACML 2.0 is parameterized.

```
2380 <xs:element name="PolicySetCombinerParameters"  
2381 type="xacml:PolicySetCombinerParametersType"/>  
2382 <xs:complexType name="PolicySetCombinerParametersType">  
2383   <xs:complexContent>  
2384     <xs:extension base="xacml:CombinerParametersType">  
2385       <xs:attribute name="PolicySetIdRef" type="xs:anyURI" use="required"/>  
2386     </xs:extension>  
2387   </xs:complexContent>  
2388 </xs:complexType>
```

2389 The <PolicySetCombinerParameters> element is of **PolicySetCombinerParametersType**  
2390 complex type.

2391 The <PolicySetCombinerParameters> element contains the following elements:  
2392 PolicySetIdRef [Required]  
2393       The identifier of a <PolicySet> or the value of a <PolicySetIdReference> contained  
2394       in the **policy set**.  
2395 Support for the <PolicySetCombinerParameters> element is optional, only if support for  
2396 **combiner parameters** is optional.

## 2397       **5.29. Element <Rule>**

2398 The <Rule> element SHALL define the individual **rules** in the **policy**. The main components of  
2399 this element are the <Target> and <Condition> elements and the Effect attribute.

2400 A <Rule> element MAY be evaluated, in which case the evaluation procedure defined in Section  
2401 7.9 SHALL be used.

```
2402 <xs:element name="Rule" type="xacml:RuleType" />  
2403 <xs:complexType name="RuleType">  
2404   <xs:sequence>  
2405     <xs:element ref="xacml:Description" minOccurs="0" />  
2406     <xs:element ref="xacml:Target" minOccurs="0" />  
2407     <xs:element ref="xacml:Condition" minOccurs="0" />  
2408   </xs:sequence>  
2409   <xs:attribute name="RuleId" type="xs:string" use="required" />  
2410   <xs:attribute name="Effect" type="xacml:EffectType" use="required" />  
2411 </xs:complexType>
```

2412 The <Rule> element is of **RuleType** complex type.

2413 The <Rule> element contains the following attributes and elements:

2414 RuleId [Required]

2415       A string identifying this **rule**.

2416 Effect [Required]

2417       **Rule effect.** The value of this attribute is either “Permit” or “Deny”.

2418 <Description> [Optional]

2419       A free-form description of the **rule**.

2420 <Target> [Optional]

2421       Identifies the set of **decision requests** that the <Rule> element is intended to evaluate. If  
2422       this element is omitted, then the **target** for the <Rule> SHALL be defined by the  
2423       <Target> element of the enclosing <Policy> element. See Section 7.6 for details.

2424 <Condition> [Optional]

2425       A **predicate** that MUST be satisfied for the **rule** to be assigned its Effect value.

## 2426       **5.30. Simple type EffectType**

2427 The **EffectType** simple type defines the values allowed for the Effect attribute of the <Rule>  
2428 element and for the FulfillOn attribute of the <Obligation> element.

```
2429 <xs:simpleType name="EffectType">
```

```

2430 <xs:restriction base="xs:string">
2431   <xs:enumeration value="Permit" />
2432   <xs:enumeration value="Deny" />
2433 </xs:restriction>
2434 </xs:simpleType>

```

### 2435 **5.31. Element <VariableDefinition>**

2436 The <VariableDefinition> element SHALL be used to define a value that can be referenced  
2437 by a <VariableReference> element. The name supplied for its VariableId attribute SHALL  
2438 NOT occur in the VariableId attribute of any other <VariableDefinition> element within the  
2439 encompassing **policy**. The <VariableDefinition> element MAY contain undefined  
2440 <VariableReference> element, but if it does, a corresponding <VariableDefinition> element  
2441 MUST be defined later in the encompassing **policy**. <VariableDefinition> elements MAY be  
2442 grouped together or MAY be placed close to the reference in the encompassing **policy**. There  
2443 MAY be zero or more references to each <VariableDefinition> element.

```

2444 <xs:element name="VariableDefinition" type="xacml:VariableDefinitionType" />
2445 <xs:complexType name="VariableDefinitionType">
2446   <xs:sequence>
2447     <xs:element ref="xacml:Expression" />
2448   </xs:sequence>
2449   <xs:attribute name="VariableId" type="xs:string" use="required" />
2450 </xs:complexType>

```

2451 The <VariableDefinition> element is of **VariableDefinitionType** complex type. The  
2452 <VariableDefinition> element has the following elements and attributes:

2453 <Expression> [Required]

2454 Any element of **ExpressionType** complex type.

2455 VariableId [Required]

2456 The name of the variable definition.

### 2457 **5.32. Element <VariableReference>**

2458 The <VariableReference> element is used to reference a value defined within the same  
2459 encompassing <Policy> element. The <VariableReference> element SHALL refer to the  
2460 <VariableDefinition> element by string equality on the value of their respective VariableId  
2461 attributes. There SHALL exist one and only one <VariableDefinition> within the same  
2462 encompassing <Policy> element to which the <VariableReference> refers. There MAY be  
2463 zero or more <VariableReference> elements that refer to the same <VariableDefinition>  
2464 element.

```

2465 <xs:element name="VariableReference" type="xacml:VariableReferenceType"
2466 substitutionGroup="xacml:Expression" />
2467 <xs:complexType name="VariableReferenceType">
2468   <xs:complexContent>
2469     <xs:extension base="xacml:ExpressionType">
2470       <xs:attribute name="VariableId" type="xs:string" use="required" />
2471     </xs:extension>
2472   </xs:complexContent>
2473 </xs:complexType>

```

2474 The <VariableReference> element is of the **VariableReferenceType** complex type, which is of  
2475 the **ExpressionType** complex type and is a member of the <Expression> element substitution  
2476 group. The <VariableReference> element MAY appear any place where an <Expression>  
2477 element occurs in the schema.



2478 The <VariableReference> element has the following attributes:

2479 VariableId [Required]

2480 The name used to refer to the value defined in a <VariableDefinition> element.

### 2481 **5.33. Element <Expression>**

2482 The <Expression> element is not used directly in a *policy*. The <Expression> element  
2483 signifies that an element that extends the **ExpressionType** and is a member of the  
2484 <Expression> element substitution group SHALL appear in its place.

```
2485 <xs:element name="Expression" type="xacml:ExpressionType" abstract="true"/>  
2486 <xs:complexType name="ExpressionType" abstract="true"/>
```

2487 The following elements are in the <Expression> element substitution group:

2488 <Apply>, <AttributeSelector>, <AttributeValue>, <Function>,  
2489 <VariableReference>, <ActionAttributeDesignator>,  
2490 <ResourceAttributeDesignator>, <SubjectAttributeDesignator> and  
2491 <EnvironmentAttributeDesignator>.

### 2492 **5.34. Element <Condition>**

2493 The <Condition> element is a Boolean function over *subject*, *resource*, *action* and  
2494 *environment attributes* or functions of *attributes*.

```
2495 <xs:element name="Condition" type="xacml:ConditionType"/>  
2496 <xs:complexType name="ConditionType">  
2497   <xs:sequence>  
2498     <xs:element ref="xacml:Expression"/>  
2499   </xs:sequence>  
2500 </xs:complexType>
```

2501 The <Condition> contains one <Expression> element, with the restriction that the  
2502 <Expression> return data-type MUST be "http://www.w3.org/2001/XMLSchema#boolean".  
2503 Evaluation of the <Condition> element is described in Section 7.8.

### 2504 **5.35. Element <Apply>**

2505 The <Apply> element denotes application of a function to its arguments, thus encoding a function  
2506 call. The <Apply> element can be applied to any combination of the members of the  
2507 <Expression> element substitution group. See Section 5.33.

```
2508 <xs:element name="Apply" type="xacml:ApplyType"  
2509 substitutionGroup="xacml:Expression"/>  
2510 <xs:complexType name="ApplyType">  
2511   <xs:complexContent>  
2512     <xs:extension base="xacml:ExpressionType">  
2513       <xs:sequence>  
2514         <xs:element ref="xacml:Expression" minOccurs="0"  
2515 maxOccurs="unbounded"/>  
2516       </xs:sequence>  
2517       <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>  
2518     </xs:extension>  
2519   </xs:complexContent>  
2520 </xs:complexType>
```

2521 The <Apply> element is of **ApplyType** complex type.



2522 The <Apply> element contains the following attributes and elements:

2523 FunctionId [Required]

2524 The identifier of the function to be applied to the arguments. XACML-defined functions are  
2525 described in Appendix A.

2526 <Expression> [Optional]

2527 Arguments to the function, which may include other functions.

## 2528 5.36. Element <Function>

2529 The <Function> element SHALL be used to name a function as an argument to the function  
2530 defined by the parent <Apply> element. In the case where the parent <Apply> element is a  
2531 higher-order **bag** function, the named function is applied to every element of the **bag** or **bags**  
2532 identified in the other arguments of the parent element. The higher-order **bag** functions are  
2533 described in Section A3A.3.12.

```
2534 <xs:element name="Function" type="xacml:FunctionType"  
2535 substitutionGroup="xacml:Expression"/>  
2536 <xs:complexType name="FunctionType">  
2537   <xs:complexContent>  
2538     <xs:extension base="xacml:ExpressionType">  
2539       <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>  
2540     </xs:extension>  
2541   </xs:complexContent>  
2542 </xs:complexType>
```

2543 The Function element is of **FunctionType** complex type.

2544 The Function element contains the following attributes:

2545 FunctionId [Required]

2546 The identifier of the function.

## 2547 5.37. Complex type AttributeDesignatorType

2548 The **AttributeDesignatorType** complex type is the type for elements that identify **attributes** by  
2549 name. It contains the information required to match **attributes** in the request **context**. See Section  
2550 7.2.4.

2551 It also contains information to control behaviour in the event that no matching **attribute** is present in  
2552 the **context**.

2553 Elements of this type SHALL NOT alter the match semantics of **named attributes**, but MAY narrow  
2554 the search space.

```
2555 <xs:complexType name="AttributeDesignatorType">  
2556   <xs:complexContent>  
2557     <xs:extension base="xacml:ExpressionType">  
2558       <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2559       <xs:attribute name="DataType" type="xs:anyURI" use="required"/>  
2560       <xs:attribute name="Issuer" type="xs:string" use="optional"/>  
2561       <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"  
2562 default="false"/>  
2563     </xs:extension>  
2564   </xs:complexContent>  
2565 </xs:complexType>
```

2566 A **named attribute** SHALL match an **attribute** if the values of their respective `AttributeId`,  
2567 `DataType` and `Issuer` attributes match. The **attribute** designator's `AttributeId` MUST match,  
2568 by URI equality, the `AttributeId` of the **attribute**. The **attribute** designator's `DataType` MUST  
2569 match, by URI equality, the `DataType` of the same **attribute**.

2570 If the `Issuer` attribute is present in the **attribute** designator, then it MUST match, using the  
2571 "urn:oasis:names:tc:xacml:1.0:function:string-equal" function, the `Issuer` of the same **attribute**. If  
2572 the `Issuer` is not present in the **attribute** designator, then the matching of the **attribute** to the  
2573 **named attribute** SHALL be governed by `AttributeId` and `DataType` attributes alone.

2574 The `<AttributeDesignatorType>` contains the following attributes:

2575 `AttributeId` [Required]

2576 This attribute SHALL specify the `AttributeId` with which to match the **attribute**.

2577 `DataType` [Required]

2578 The bag returned by the `<AttributeDesignator>` element SHALL contain values of this  
2579 data-type.

2580 `Issuer` [Optional]

2581 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **attribute**.

2582 `MustBePresent` [Optional]

2583 This attribute governs whether the element returns "Indeterminate" or an empty **bag** in the  
2584 event the **named attribute** is absent from the request **context**. See Section 7.2.5. Also  
2585 see Sections 7.15.2 and 7.15.3.

## 2586 **5.38. Element `<SubjectAttributeDesignator>`**

2587 The `<SubjectAttributeDesignator>` element retrieves a **bag** of values for a **named**  
2588 categorized **subject attribute** from the request **context**. A **subject attribute** is an **attribute**  
2589 contained within a `<Subject>` element of the request **context**. A categorized **subject** is a **subject**  
2590 that is identified by a particular **subject-category** attribute. A **named categorized subject attribute**  
2591 is a **named subject attribute** for a particular **categorized subject**.

2592 The `<SubjectAttributeDesignator>` element SHALL return a **bag** containing all the **subject**  
2593 **attribute** values that are matched by the **named categorized subject attribute**. In the event that  
2594 no matching attribute is present in the context, the `MustBePresent` attribute governs whether this  
2595 element returns an empty **bag** or "Indeterminate". See Section 7.2.5.

2596 The `SubjectAttributeDesignatorType` extends the match semantics of the  
2597 `AttributeDesignatorType` (See Section 5.37) such that it narrows the **attribute** search space to  
2598 the specific **categorized subject** such that the value of this element's `SubjectCategory` attribute  
2599 matches, by URI equality, the value of the request **context's** `<Subject>` element's  
2600 `SubjectCategory` attribute.

2601 If the request context contains multiple **subjects** with the same `SubjectCategory` XML attribute,  
2602 then they SHALL be treated as if they were one **categorized subject**.

2603 The `<SubjectAttributeDesignator>` MAY appear in the `<SubjectMatch>` element and  
2604 MAY be passed to the `<Apply>` element as an argument.

```

2605 <xs:element name="SubjectAttributeDesignator"
2606 type="xacml:SubjectAttributeDesignatorType"
2607 substitutionGroup="xacml:Expression"/>
2608 <xs:complexType name="SubjectAttributeDesignatorType">
2609   <xs:complexContent>
2610     <xs:extension base="xacml:AttributeDesignatorType">
2611       <xs:attribute name="SubjectCategory" type="xs:anyURI" use="optional"
2612 default="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"/>
2613     </xs:extension>
2614   </xs:complexContent>
2615 </xs:complexType>

```

2616 The <SubjectAttributeDesignator> element is of type **SubjectAttributeDesignatorType**.  
 2617 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType**  
 2618 complex type with a SubjectCategory attribute.

2619 SubjectCategory [Optional]

2620 This attribute SHALL specify the *categorized subject* from which to match *named subject*  
 2621 *attributes*. If SubjectCategory is not present, then its default value of  
 2622 "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" SHALL be used. Standard  
 2623 values of the SubjectCategory are listed in Section B.2.

## 2624 5.39. Element <ResourceAttributeDesignator>

2625 The <ResourceAttributeDesignator> element retrieves a **bag** of values for a *named*  
 2626 **resource attribute** from the request **context**. A **resource attribute** is an **attribute** contained  
 2627 within the <Resource> element of the request **context**. A *named resource attribute* is a **named**  
 2628 **attribute** that matches a **resource attribute**. A *named resource attribute* SHALL be considered  
 2629 *present* if there is at least one **resource attribute** that matches the criteria set out below. A  
 2630 **resource attribute** value is an **attribute** value that is contained within a **resource attribute**.

2631 The <ResourceAttributeDesignator> element SHALL return a **bag** containing all the  
 2632 **resource attribute** values that are matched by the *named resource attribute*. In the event that no  
 2633 matching attribute is present in the context, the MustBePresent attribute governs whether this  
 2634 element returns an empty **bag** or "Indeterminate". See Section 7.2.5.

2635 A *named resource attribute* SHALL match a **resource attribute** as per the match semantics  
 2636 specified in the **AttributeDesignatorType** complex type. See Section 5.37.

2637 The <ResourceAttributeDesignator> MAY appear in the <ResourceMatch> element and  
 2638 MAY be passed to the <Apply> element as an argument.

```

2639 <xs:element name="ResourceAttributeDesignator"
2640 type="xacml:AttributeDesignatorType" substitutionGroup="xacml:Expression"/>

```

2641 The <ResourceAttributeDesignator> element is of the **AttributeDesignatorType** complex  
 2642 type.

## 2643 5.40. Element <ActionAttributeDesignator>

2644 The <ActionAttributeDesignator> element retrieves a **bag** of values for a *named action*  
 2645 **attribute** from the request **context**. An **action attribute** is an **attribute** contained within the  
 2646 <Action> element of the request **context**. A *named action attribute* has specific criteria  
 2647 (described below) with which to match an **action attribute**. A *named action attribute* SHALL be  
 2648 considered *present*, if there is at least one **action attribute** that matches the criteria. An **action**  
 2649 **attribute value** is an **attribute value** that is contained within an **action attribute**.

2650 The <ActionAttributeDesignator> element SHALL return a **bag** of all the **action attribute**  
2651 values that are matched by the **named action attribute**. In the event that no matching attribute is  
2652 present in the context, the `MustBePresent` attribute governs whether this element returns an  
2653 empty **bag** or “Indeterminate”. See Section 7.2.5.

2654 A **named action attribute** SHALL match an **action attribute** as per the match semantics specified  
2655 in the **AttributeDesignatorType** complex type. See Section 5.37.

2656 The <ActionAttributeDesignator> MAY appear in the <ActionMatch> element and MAY  
2657 be passed to the <Apply> element as an argument.

```
2658 <xs:element name="ActionAttributeDesignator" type="xacml:AttributeDesignatorType"  
2659 substitutionGroup="xacml:Expression" />
```

2660 The <ActionAttributeDesignator> element is of the **AttributeDesignatorType** complex  
2661 type.

## 2662 5.41. Element <EnvironmentAttributeDesignator>

2663 The <EnvironmentAttributeDesignator> element retrieves a **bag** of values for a **named**  
2664 **environment attribute** from the request **context**. An **environment attribute** is an **attribute**  
2665 contained within the <Environment> element of request **context**. A **named environment**  
2666 **attribute** has specific criteria (described below) with which to match an **environment attribute**. A  
2667 **named environment attribute** SHALL be considered *present*, if there is at least one **environment**  
2668 **attribute** that matches the criteria. An **environment attribute value** is an **attribute** value that is  
2669 contained within an **environment attribute**.

2670 The <EnvironmentAttributeDesignator> element SHALL evaluate to a **bag** of all the  
2671 **environment attribute** values that are matched by the **named environment attribute**. In the  
2672 event that no matching attribute is present in the context, the `MustBePresent` attribute governs  
2673 whether this element returns an empty **bag** or “Indeterminate”. See Section 7.2.5.

2674 A **named environment attribute** SHALL match an **environment attribute** as per the match  
2675 semantics specified in the **AttributeDesignatorType** complex type. See Section 5.37.

2676 The <EnvironmentAttributeDesignator> MAY be passed to the <Apply> element as an  
2677 argument.

```
2678 <xs:element name="EnvironmentAttributeDesignator"  
2679 type="xacml:AttributeDesignatorType" substitutionGroup="xacml:Expression" />
```

2680 The <EnvironmentAttributeDesignator> element is of the **AttributeDesignatorType**  
2681 complex type.

## 2682 5.42. Element <AttributeSelector>

2683 The <AttributeSelector> element identifies attributes by their location in the request **context**.  
2684 Support for the <AttributeSelector> element is OPTIONAL.

2685 The <AttributeSelector> element's `RequestContextPath` XML attribute SHALL contain a  
2686 legal XPath expression whose context node is the <xacml-context:Request> element. The  
2687 **AttributeSelector** element SHALL evaluate to a **bag** of values whose data-type is specified by  
2688 the element's `DataType` attribute. If the `DataType` specified in the **AttributeSelector** is a  
2689 primitive data type defined in [XF] or [XS], then the value returned by the XPath expression SHALL  
2690 be converted to the `DataType` specified in the <AttributeSelector> using the constructor  
2691 function below [XF Section 4] that corresponds to the `DataType`. If an error results from using the  
2692 constructor function, then the value of the <AttributeSelector> SHALL be "Indeterminate".  
2693

2694 xs:string()  
2695 xs:boolean()  
2696 xs:integer()  
2697 xs:double()  
2698 xs:dateTime()  
2699 xs:date()  
2700 xs:time()  
2701 xs:hexBinary()  
2702 xs:base64Binary()  
2703 xs:anyURI()  
2704 xf:yearMonthDuration()  
2705 xf:dayTimeDuration()  
2706

2707 If the `Data Type` specified in the `AttributeSelector` is not one of the preceding primitive  
2708 `Data Types`, then the `AttributeSelector` SHALL return a **bag** of instances of the specified  
2709 `Data Type`. If an error occurs when converting the values returned by the XPath expression to the  
2710 specified `Data Type`, then the result of the `AttributeSelector` SHALL be "Indeterminate".  
2711

2712 Each node selected by the specified XPath expression MUST be either a text node, an attribute  
2713 node, a processing instruction node or a comment node. The string representation of the value of  
2714 each node MUST be converted to an **attribute** value of the specified data-type, and the result of  
2715 the `AttributeSelector` is the **bag** of the **attribute** values generated from all the selected  
2716 nodes.  
2717

2718 If the node selected by the specified XPath expression is not one of those listed above (i.e. a text  
2719 node, an attribute node, a processing instruction node or a comment node), then the result of the  
2720 enclosing **policy** SHALL be "Indeterminate" with a `Status Code` value of  
2721 "urn:oasis:names:tc:xacml:1.0:status:syntax-error".  
2722

```
2723 <xs:element name="AttributeSelector" type="xacml:AttributeSelectorType"  
2724 substitutionGroup="xacml:Expression" />  
2725 <xs:complexType name="AttributeSelectorType" >  
2726 <xs:complexContent>  
2727 <xs:extension base="xacml:ExpressionType">  
2728 <xs:attribute name="RequestContextPath" type="xs:string" use="required"/>  
2729 <xs:attribute name="DataType" type="xs:anyURI" use="required"/>  
2730 <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"  
2731 default="false" />  
2732 </xs:extension>  
2733 </xs:complexContent>  
2734 </xs:complexType>
```

2735 The `<AttributeSelector>` element is of **AttributeSelectorType** complex type.

2736 The `<AttributeSelector>` element has the following attributes:

2737 RequestContextPath [Required]

2738 An XPath expression whose context node is the `<xacml-context:Request>` element.  
2739 There SHALL be no restriction on the XPath syntax. See also Section 5.4.

2740 DataType [Required]

2741 The **bag** returned by the `<AttributeSelector>` element SHALL contain values of this  
2742 data-type.

2743 MustBePresent [Optional]

2744 This attribute governs whether the element returns "Indeterminate" or an empty **bag** in the  
2745 event the XPath expression selects no node. See Section 7.2.5. Also see Sections 7.15.2  
2746 and 7.15.3.

### 2747 **5.43. Element <AttributeValue>**

2748 The <xacml:AttributeValue> element SHALL contain a literal **attribute** value.

```
2749 <xs:element name="AttributeValue" type="xacml:AttributeValueType"  
2750 substitutionGroup="xacml:Expression" />  
2751 <xs:complexType name="AttributeValueType" mixed="true">  
2752 <xs:complexContent>  
2753 <xs:extension base="xacml:ExpressionType">  
2754 <xs:sequence>  
2755 <xs:any namespace="##any" processContents="lax" minOccurs="0"  
2756 maxOccurs="unbounded" />  
2757 </xs:sequence>  
2758 <xs:attribute name="DataType" type="xs:anyURI" use="required" />  
2759 <xs:anyAttribute namespace="##any" processContents="lax" />  
2760 </xs:extension>  
2761 </xs:complexContent>  
2762 </xs:complexType>
```

2763 The <xacml:AttributeValue> element is of **AttributeValueType** complex type.

2764 The <xacml:AttributeValue> element has the following attributes:

2765 **DataType** [Required]

2766 The data-type of the **attribute** value.

### 2767 **5.44. Element <Obligations>**

2768 The <Obligations> element SHALL contain a set of <Obligation> elements.

2769 Support for the <Obligations> element is OPTIONAL.

```
2770 <xs:element name="Obligations" type="xacml:ObligationsType" />  
2771 <xs:complexType name="ObligationsType">  
2772 <xs:sequence>  
2773 <xs:element ref="xacml:Obligation" maxOccurs="unbounded" />  
2774 </xs:sequence>  
2775 </xs:complexType>
```

2776 The <Obligations> element is of **ObligationsType** complexType.

2777 The <Obligations> element contains the following element:

2778 <Obligation> [One to Many]

2779 A sequence of **obligations**. See Section 5.45.

### 2780 **5.45. Element <Obligation>**

2781 The <Obligation> element SHALL contain an identifier for the **obligation** and a set of **attributes**  
2782 that form arguments of the action defined by the **obligation**. The FulfillOn attribute SHALL  
2783 indicate the **effect** for which this **obligation** must be fulfilled by the **PEP**.

```
2784 <xs:element name="Obligation" type="xacml:ObligationType" />  
2785 <xs:complexType name="ObligationType">  
2786 <xs:sequence>
```



```

2787     <xs:element ref="xacml:AttributeAssignment" minOccurs="0"
2788     maxOccurs="unbounded" />
2789   </xs:sequence>
2790   <xs:attribute name="ObligationId" type="xs:anyURI" use="required" />
2791   <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required" />
2792 </xs:complexType>

```

2793 The <Obligation> element is of **ObligationType** complexType. See Section 7.14 for a  
 2794 description of how the set of **obligations** to be returned by the **PDP** is determined.

2795 The <Obligation> element contains the following elements and attributes:

2796 ObligationId [Required]

2797         **Obligation** identifier. The value of the **obligation** identifier SHALL be interpreted by the  
 2798         **PEP**.

2799 FulfillOn [Required]

2800         The **effect** for which this **obligation** must be fulfilled by the **PEP**.

2801 <AttributeAssignment> [Optional]

2802         **Obligation** arguments assignment. The values of the **obligation** arguments SHALL be  
 2803         interpreted by the **PEP**.

## 2804         **5.46. Element <AttributeAssignment>**

2805 The <AttributeAssignment> element is used for including arguments in **obligations**. It SHALL  
 2806 contain an **AttributeId** and the corresponding **attribute** value, by extending the  
 2807 **AttributeValueType** type definition. The <AttributeAssignment> element MAY be used in  
 2808 any way that is consistent with the schema syntax, which is a sequence of <xs:any> elements.  
 2809 The value specified SHALL be understood by the **PEP**, but it is not further specified by XACML.  
 2810 See Section 7.14. Section 4.2.4.3 provides a number of examples of arguments included in  
 2811 **obligations**.

```

2812 <xs:element name="AttributeAssignment" type="xacml:AttributeAssignmentType" />
2813 <xs:complexType name="AttributeAssignmentType" mixed="true">
2814   <xs:complexContent>
2815     <xs:extension base="xacml:AttributeValueType">
2816       <xs:attribute name="AttributeId" type="xs:anyURI" use="required" />
2817     </xs:extension>
2818   </xs:complexContent>
2819 </xs:complexType>

```

2820 The <AttributeAssignment> element is of **AttributeAssignmentType** complex type.

2821 The <AttributeAssignment> element contains the following attributes:

2822 AttributeId [Required]

2823         The **attribute** Identifier.



2824

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

2825

2826

### 6.1. Element <Request>

2827

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.

2828

2829

2830

2831

2832

2833

2834

The <Request> element contains <Subject>, <Resource>, <Action> and <Environment> elements. There may be multiple <Subject> elements and, under some conditions, multiple <Resource> elements<sup>2</sup>. 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.

2835

2836

2837

2838

2839

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

2840

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

2841

```
  <xs:sequence>
```

2842

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

2843

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

2844

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

2845

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

2846

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

2847

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

2848

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

2849

The <Request> element contains the following elements:

2850

<Subject> [One to Many]

2851

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.

2852

2853

2854

2855

2856

2857

2858

2859

2860

<Resource> [One to Many]

2861

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.

2862

2863

---

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

- 2864 <Action> [Required]
- 2865           Specifies the requested **action** to be performed on the **resource** by listing a set of
- 2866           <Attribute> elements associated with the **action**.
- 2867 <Environment> [Required]
- 2868           Contains a set of <Attribute> elements for the **environment**.

## 2869           **6.2. Element <Subject>**

2870 The <Subject> element specifies a **subject** by listing a sequence of <Attribute> elements

2871 associated with the **subject**.

```
2872 <xs:element name="Subject" type="xacml-context:SubjectType" />
2873 <xs:complexType name="SubjectType">
2874   <xs:sequence>
2875     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2876     maxOccurs="unbounded" />
2877   </xs:sequence>
2878   <xs:attribute name="SubjectCategory" type="xs:anyURI"
2879   default="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" />
2880 </xs:complexType>
```

2881 The <Subject> element is of **SubjectType** complex type.

2882 The <Subject> element contains the following elements and attributes:

2883 SubjectCategory [Optional]

2884           This attribute indicates the role that the parent <Subject> played in the formation of the

2885           **access** request. If this attribute is not present in a given <Subject> element, then the

2886           default value of “urn:oasis:names:tc:xacml:1.0:subject-category:access-subject” SHALL be

2887           used, indicating that the parent <Subject> element represents the entity ultimately

2888           responsible for initiating the **access** request.

2889           If more than one <Subject> element contains a "urn:oasis:names:tc:xacml:2.0:subject-

2890           category" attribute with the same value, then the PDP SHALL treat the contents of those

2891           elements as if they were contained in the same <Subject> element.

2892 <Attribute> [Any Number]

2893           A sequence of **attributes** that apply to the subject.

2894           Typically, a <Subject> element will contain an <Attribute> with an AttributeId of

2895           “urn:oasis:names:tc:xacml:1.0:subject:subject-id”, containing the identity of the **subject**.

2896           A <Subject> element MAY contain additional <Attribute> elements.

## 2897           **6.3. Element <Resource>**

2898 The <Resource> element specifies information about the **resource** to which **access** is requested,

2899 by listing a sequence of <Attribute> elements associated with the **resource**. It MAY include the

2900 **resource** content.

```
2901 <xs:element name="Resource" type="xacml-context:ResourceType" />
2902 <xs:complexType name="ResourceType">
2903   <xs:sequence>
2904     <xs:element ref="xacml-context:ResourceContent" minOccurs="0" />
```

```

2905     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2906     maxOccurs="unbounded" />
2907     </xs:sequence>
2908 </xs:complexType>

```

2909 The <Resource> element is of **ResourceType** complex type.

2910 The <Resource> element contains the following elements:

2911 <ResourceContent> [Optional]

2912       The **resource** content.

2913 <Attribute> [Any Number]

2914       A sequence of **resource attributes**.

2915       The <Resource> element MAY contain one or more <Attribute> elements with an  
2916       AttributeId of "urn:oasis:names:tc:xacml:2.0:resource:resource-id". Each such  
2917       <Attribute> SHALL be an absolute and fully-resolved representation of the identity of  
2918       the single **resource** to which access is being requested. If there is more than one such  
2919       absolute and fully-resolved representation, and if any <Attribute> with this  
2920       AttributeId is specified, then an <Attribute> for each such distinct representation of  
2921       the **resource** identity SHALL be specified. All such <Attribute> elements SHALL refer  
2922       to the same single **resource** instance. A Profile for a particular **resource** MAY specify a  
2923       single normative representation for instances of the **resource**; in this case, any  
2924       <Attribute> with this AttributeId SHALL use only this one representation.

2925       A <Resource> element MAY contain additional <Attribute> elements.

## 2926 **6.4. Element <ResourceContent>**

2927 The <ResourceContent> element is a notional placeholder for the content of the **resource**. If an  
2928 XACML **policy** references the contents of the **resource** by means of an <AttributeSelector>  
2929 element, then the <ResourceContent> element MUST be included in the  
2930 RequestContextPath string.

```

2931 <xs:complexType name="ResourceContentType" mixed="true">
2932   <xs:sequence>
2933     <xs:any namespace="##any" processContents="lax" minOccurs="0"
2934     maxOccurs="unbounded" />
2935   </xs:sequence>
2936   <xs:anyAttribute namespace="##any" processContents="lax" />
2937 </xs:complexType>

```

2938 The <ResourceContent> element is of **ResourceContentType** complex type.

2939 The <ResourceContent> element allows arbitrary elements and attributes.

## 2940 **6.5. Element <Action>**

2941 The <Action> element specifies the requested **action** on the **resource**, by listing a set of  
2942 <Attribute> elements associated with the **action**.

```

2943 <xs:element name="Action" type="xacml-context:ActionType" />
2944 <xs:complexType name="ActionType">
2945   <xs:sequence>
2946     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2947     maxOccurs="unbounded" />
2948   </xs:sequence>

```

2949 `</xs:complexType>`

2950 The `<Action>` element is of **ActionType** complex type.

2951 The `<Action>` element contains the following elements:

2952 `<Attribute>` [Any Number]

2953 List of **attributes** of the **action** to be performed on the **resource**.

## 2954 **6.6. Element `<Environment>`**

2955 The `<Environment>` element contains a set of **attributes** of the **environment**.

```
2956 <xs:element name="Environment" type="xacml-context:EnvironmentType" />
2957 <xs:complexType name="EnvironmentType">
2958   <xs:sequence>
2959     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2960     maxOccurs="unbounded" />
2961   </xs:sequence>
2962 </xs:complexType>
```

2963 The `<Environment>` element is of **EnvironmentType** complex type.

2964 The `<Environment>` element contains the following elements:

2965 `<Attribute>` [Any Number]

2966 A list of **environment attributes**. Environment **attributes** are **attributes** that are not  
2967 associated with either the **resource**, the **action** or any of the **subjects** of the **access**  
2968 request.

## 2969 **6.7. Element `<Attribute>`**

2970 The `<Attribute>` element is the central abstraction of the request **context**. It contains **attribute**  
2971 meta-data and one or more **attribute** values. The **attribute** meta-data comprises the **attribute**  
2972 identifier and the **attribute** issuer. `<AttributeDesignator>` and `<AttributeSelector>`  
2973 elements in the **policy** MAY refer to **attributes** by means of this meta-data.

```
2974 <xs:element name="Attribute" type="xacml-context:AttributeType" />
2975 <xs:complexType name="AttributeType">
2976   <xs:sequence>
2977     <xs:element ref="xacml-context:AttributeValue" maxOccurs="unbounded" />
2978   </xs:sequence>
2979   <xs:attribute name="AttributeId" type="xs:anyURI" use="required" />
2980   <xs:attribute name="DataType" type="xs:anyURI" use="required" />
2981   <xs:attribute name="Issuer" type="xs:string" use="optional" />
2982 </xs:complexType>
```

2983 The `<Attribute>` element is of **AttributeType** complex type.

2984 The `<Attribute>` element contains the following attributes and elements:

2985 **AttributeId** [Required]

2986 The **Attribute** identifier. A number of identifiers are reserved by XACML to denote  
2987 commonly used **attributes**. See Appendix B.

2988 **DataType** [Required]

2989 The data-type of the contents of the `<xacml-context:AttributeValue>` element.  
2990 This SHALL be either a primitive type defined by the XACML 2.0 specification or a type

2991 (primitive or structured) defined in a namespace declared in the `<xacml-context>`  
 2992 element.

2993 Issuer [Optional]

2994 The **Attribute** issuer. For example, this attribute value MAY be an x500Name that binds to  
 2995 a public key, or it may be some other identifier exchanged out-of-band by issuing and  
 2996 relying parties.

2997 `<xacml-context:AttributeValue>` [One to Many]

2998 One or more **attribute** values. Each **attribute** value MAY have contents that are empty,  
 2999 occur once or occur multiple times.

## 6.8. Element `<AttributeValue>`

3000 The `<xacml-context:AttributeValue>` element contains the value of an **attribute**.

```
3001 <xs:element name="AttributeValue" type="xacml-context:AttributeValueType" />
3002 <xs:complexType name="AttributeValueType" mixed="true">
3003   <xs:sequence>
3004     <xs:any namespace="##any" processContents="lax" minOccurs="0"
3005     maxOccurs="unbounded" />
3006   </xs:sequence>
3007   <xs:anyAttribute namespace="##any" processContents="lax" />
3008 </xs:complexType>
```

3010 The `<xacml-context:AttributeValue>` element is of **AttributeValueType** complex type.

3011 The data-type of the `<xacml-context:AttributeValue>` SHALL be specified by using the  
 3012 `DataType` attribute of the parent `<Attribute>` element.

## 6.9. Element `<Response>`

3014 The `<Response>` element is a top-level element in the XACML **context** schema. The  
 3015 `<Response>` element is an abstraction layer used by the **policy** language. Any proprietary  
 3016 system using the XACML specification MUST transform an XACML **context** `<Response>` element  
 3017 into the form of its **authorization decision**.

3018 The `<Response>` element encapsulates the **authorization decision** produced by the **PDP**. It includes  
 3019 a sequence of one or more results, with one `<Result>` element per requested **resource**. Multiple  
 3020 results MAY be returned by some implementations, in particular those that support the XACML  
 3021 Profile for Requests for Multiple Resources [MULT]. Support for multiple results is OPTIONAL.

```
3022 <xs:element name="Response" type="xacml-context:ResponseType" />
3023 <xs:complexType name="ResponseType">
3024   <xs:sequence>
3025     <xs:element ref="xacml-context:Result" maxOccurs="unbounded" />
3026   </xs:sequence>
3027 </xs:complexType>
```

3028 The `<Response>` element is of **ResponseType** complex type.

3029 The `<Response>` element contains the following elements:

3030 `<Result>` [One to Many]

3031 An authorization decision result. See Section 6.10.

## 3032 6.10. Element <Result>

3033 The <Result> element represents an **authorization decision** result for the **resource** specified by  
3034 the ResourceId **attribute**. It MAY include a set of **obligations** that MUST be fulfilled by the **PEP**.  
3035 If the **PEP** does not understand or cannot fulfill an **obligation**, then it MUST act as if the **PDP** had  
3036 denied **access** to the requested **resource**.

3037

```
3038 <xs:complexType name="ResultType">  
3039   <xs:sequence>  
3040     <xs:element ref="xacml-context:Decision"/>  
3041     <xs:element ref="xacml-context:Status" minOccurs="0"/>  
3042     <xs:element ref="xacml:Obligations" minOccurs="0"/>  
3043   </xs:sequence>  
3044   <xs:attribute name="ResourceId" type="xs:string" use="optional"/>  
3045 </xs:complexType>
```

3046 The <Result> element is of **ResultType** complex type.

3047 The <Result> element contains the following attributes and elements:

3048 ResourceId [Optional]

3049 The identifier of the requested **resource**. If this attribute is omitted, then the **resource**  
3050 identity is that specified by the "urn:oasis:names:tc:xacml:1.0:resource:resource-id"  
3051 **resource attribute** in the corresponding <Request> element.

3052 <Decision> [Required]

3053 The **authorization decision**: "Permit", "Deny", "Indeterminate" or "NotApplicable".

3054 <Status> [Optional]

3055 Indicates whether errors occurred during evaluation of the **decision request**, and  
3056 optionally, information about those errors. If the <Response> element contains <Result>  
3057 elements whose <Status> elements are all identical, and the <Response> element is  
3058 contained in a protocol wrapper that can convey status information, then the common  
3059 status information MAY be placed in the protocol wrapper and this <Status> element  
3060 MAY be omitted from all <Result> elements.

3061 <Obligations> [Optional]

3062 A list of **obligations** that MUST be fulfilled by the **PEP**. If the **PEP** does not understand or  
3063 cannot fulfill an **obligation**, then it MUST act as if the **PDP** had denied **access** to the  
3064 requested **resource**. See Section 7.14 for a description of how the set of **obligations** to  
3065 be returned by the PDP is determined.

## 3066 6.11. Element <Decision>

3067 The <Decision> element contains the result of **policy** evaluation.

```
3068 <xs:element name="Decision" type="xacml-context:DecisionType"/>  
3069 <xs:simpleType name="DecisionType">  
3070   <xs:restriction base="xs:string">  
3071     <xs:enumeration value="Permit"/>  
3072     <xs:enumeration value="Deny"/>  
3073     <xs:enumeration value="Indeterminate"/>  
3074     <xs:enumeration value="NotApplicable"/>  
3075   </xs:restriction>  
3076 </xs:simpleType>
```



- 3077 The <Decision> element is of **DecisionType** simple type.
- 3078 The values of the <Decision> element have the following meanings:
- 3079       “Permit”: the requested **access** is permitted.
- 3080       “Deny”: the requested **access** is denied.
- 3081       “Indeterminate”: the PDP is unable to evaluate the requested **access**. Reasons for such inability include: missing **attributes**, network errors while retrieving **policies**, division by zero during **policy** evaluation, syntax errors in the **decision request** or in the **policy**, etc..
- 3084       “NotApplicable”: the **PDP** does not have any **policy** that applies to this **decision request**.

## 3085 **6.12. Element <Status>**

3086 The <Status> element represents the status of the **authorization decision** result.

```
3087 <xs:element name="Status" type="xacml-context:StatusType" />
3088 <xs:complexType name="StatusType">
3089   <xs:sequence>
3090     <xs:element ref="xacml-context:StatusCode" />
3091     <xs:element ref="xacml-context:StatusMessage" minOccurs="0" />
3092     <xs:element ref="xacml-context:StatusDetail" minOccurs="0" />
3093   </xs:sequence>
3094 </xs:complexType>
```

3095 The <Status> element is of **StatusType** complex type.

3096 The <Status> element contains the following elements:

3097 <StatusCode> [Required]

3098       Status code.

3099 <StatusMessage> [Optional]

3100       A status message describing the status code.

3101 <StatusDetail> [Optional]

3102       Additional status information.

## 3103 **6.13. Element <StatusCode>**

3104 The <StatusCode> element contains a major status code value and an optional sequence of minor status codes.

```
3106 <xs:element name="StatusCode" type="xacml-context:StatusCodeType" />
3107 <xs:complexType name="StatusCodeType">
3108   <xs:sequence>
3109     <xs:element ref="xacml-context:StatusCode" minOccurs="0" />
3110   </xs:sequence>
3111   <xs:attribute name="Value" type="xs:anyURI" use="required" />
3112 </xs:complexType>
```

3113 The <StatusCode> element is of **StatusCodeType** complex type.

3114 The <StatusCode> element contains the following attributes and elements:

3115 Value [Required]

3116 See Section B.9 for a list of values.

3117 `<StatusCode>` [Any Number]

3118 Minor status code. This status code qualifies its parent status code.

## 3119 **6.14. Element `<StatusMessage>`**

3120 The `<StatusMessage>` element is a free-form description of the status code.

```
3121 <xs:element name="StatusMessage" type="xs:string"/>
```

3122 The `<StatusMessage>` element is of **xs:string** type.

## 3123 **6.15. Element `<StatusDetail>`**

3124 The `<StatusDetail>` element qualifies the `<Status>` element with additional information.

```
3125 <xs:element name="StatusDetail" type="xacml-context:StatusDetailType"/>
```

```
3126 <xs:complexType name="StatusDetailType">
```

```
3127 <xs:sequence>
```

```
3128 <xs:any namespace="##any" processContents="lax" minOccurs="0"
```

```
3129 maxOccurs="unbounded"/>
```

```
3130 </xs:sequence>
```

```
3131 </xs:complexType>
```

3132 The `<StatusDetail>` element is of **StatusDetailType** complex type.

3133 The `<StatusDetail>` element allows arbitrary XML content.

3134 Inclusion of a `<StatusDetail>` element is optional. However, if a **PDP** returns one of the  
3135 following XACML-defined `<StatusCode>` values and includes a `<StatusDetail>` element, then  
3136 the following rules apply.

3137 urn:oasis:names:tc:xacml:1.0:status:ok

3138 A **PDP** MUST NOT return a `<StatusDetail>` element in conjunction with the “ok” status value.

3139 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

3140 A **PDP** MAY choose not to return any `<StatusDetail>` information or MAY choose to return a  
3141 `<StatusDetail>` element containing one or more `<xacml-context:`  
3142 `MissingAttributeDetail>` elements.

3143 urn:oasis:names:tc:xacml:1.0:status:syntax-error

3144 A **PDP** MUST NOT return a `<StatusDetail>` element in conjunction with the “syntax-error” status  
3145 value. A syntax error may represent either a problem with the **policy** being used or with the  
3146 request **context**. The **PDP** MAY return a `<StatusMessage>` describing the problem.

3147 urn:oasis:names:tc:xacml:1.0:status:processing-error

3148 A **PDP** MUST NOT return `<StatusDetail>` element in conjunction with the “processing-error”  
3149 status value. This status code indicates an internal problem in the **PDP**. For security reasons, the  
3150 **PDP** MAY choose to return no further information to the **PEP**. In the case of a divide-by-zero error  
3151 or other computational error, the **PDP** MAY return a `<StatusMessage>` describing the nature of  
3152 the error.

3153

## 6.16. Element <MissingAttributeDetail>

3154 The <MissingAttributeDetail> element conveys information about **attributes** required for  
3155 **policy** evaluation that were missing from the request **context**.

```
3156 <xs:element name="MissingAttributeDetail" type="xacml -  
3157 context:MissingAttributeDetailType" />  
3158 <xs:complexType name="MissingAttributeDetailType">  
3159   <xs:sequence>  
3160     <xs:element ref="xacml-context:AttributeValue" minOccurs="0"  
3161     maxOccurs="unbounded" />  
3162   </xs:sequence>  
3163   <xs:attribute name="AttributeId" type="xs:anyURI" use="required" />  
3164   <xs:attribute name="DataType" type="xs:anyURI" use="required" />  
3165   <xs:attribute name="Issuer" type="xs:string" use="optional" />  
3166 </xs:complexType>
```

3167 The <MissingAttributeDetail> element is of **MissingAttributeDetailType** complex type.

3168 The <MissingAttributeDetail> element contains the following attributes and elements:

3169 AttributeValue [Optional]

3170       The required value of the missing **attribute**.

3171 <AttributeId> [Required]

3172       The identifier of the missing **attribute**.

3173 <DataType> [Required]

3174       The data-type of the missing **attribute**.

3175 Issuer [Optional]

3176       This attribute, if supplied, SHALL specify the required **Issuer** of the missing **attribute**.

3177 If the PDP includes <xacml-context:AttributeValue> elements in the <MissingAttributeDetail>  
3178 element, then this indicates the acceptable values for that attribute. If no <xacml-  
3179 context:AttributeValue> elements are included, then this indicates the names of attributes that the  
3180 PDP failed to resolve during its evaluation. The list of attributes may be partial or complete. There  
3181 is no guarantee by the PDP that supplying the missing values or attributes will be sufficient to  
3182 satisfy the policy.

---

## 3183 7. Functional requirements (normative)

3184 This section specifies certain functional requirements that are not directly associated with the  
3185 production or consumption of a particular XACML element.

### 3186 7.1. Policy enforcement point

3187 This section describes the requirements for the **PEP**.

3188 An application functions in the role of the **PEP** if it guards access to a set of **resources** and asks  
3189 the **PDP** for an **authorization decision**. The **PEP** MUST abide by the **authorization decision** as  
3190 described in one of the following sub-sections

3191

### 7.1.1. Base PEP

3192 If the **decision** is "Permit", then the **PEP** SHALL permit **access**. If **obligations** accompany the  
3193 **decision**, then the **PEP** SHALL permit **access** only if it understands and it can and will discharge  
3194 those **obligations**.

3195 If the **decision** is "Deny", then the **PEP** SHALL deny **access**. If **obligations** accompany the  
3196 **decision**, then the **PEP** shall deny **access** only if it understands, and it can and will discharge  
3197 those **obligations**.

3198 If the **decision** is "Not Applicable", then the **PEP's** behavior is undefined.

3199 If the **decision** is "Indeterminate", then the **PEP's** behavior is undefined.

3200

### 7.1.2. Deny-biased PEP

3201 If the **decision** is "Permit", then the **PEP** SHALL permit **access**. If **obligations** accompany the  
3202 **decision**, then the **PEP** SHALL permit **access** only if it understands and it can and will discharge  
3203 those **obligations**.

3204 All other **decisions** SHALL result in the denial of **access**.

3205 Note: other actions, e.g. consultation of additional **PDPs**, reformulation/resubmission of the  
3206 **decision request**, etc., are not prohibited.

3207

### 7.1.3. Permit-biased PEP

3208 If the **decision** is "Deny", then the **PEP** SHALL deny **access**. If **obligations** accompany the  
3209 **decision**, then the **PEP** shall deny **access** only if it understands, and it can and will discharge  
3210 those **obligations**.

3211 All other **decisions** SHALL result in the permission of **access**.

3212 Note: other actions, e.g. consultation of additional **PDPs**, reformulation/resubmission of the  
3213 **decision request**, etc., are not prohibited.

3214

## 7.2. Attribute evaluation

3215 **Attributes** are represented in the request **context** by the **context handler**, regardless of whether  
3216 or not they appeared in the original **decision request**, and are referred to in the **policy** by **subject**,  
3217 **resource**, **action** and **environment attribute** designators and **attribute** selectors. A **named**  
3218 **attribute** is the term used for the criteria that the specific **subject**, **resource**, **action** and  
3219 **environment attribute** designators and selectors use to refer to particular **attributes** in the  
3220 **subject**, **resource**, **action** and **environment** elements of the request **context**, respectively.

3221

### 7.2.1. Structured attributes

3222 <xacml:AttributeValue> and <xacml-context:AttributeValue> elements MAY contain  
3223 an instance of a structured XML data-type, for example <ds:KeyInfo>. XACML 2.0 supports  
3224 several ways for comparing the contents of such elements.

3225 1. In some cases, such elements MAY be compared using one of the XACML string functions,  
3226 such as "regexp-string-match", described below. This requires that the element be given  
3227 the data-type "<http://www.w3.org/2001/XMLSchema#string>". For example, a structured  
3228 data-type that is actually a ds:KeyInfo/KeyName would appear in the Context as:

```
3229 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
3230   &lt;ds:KeyName&gt;jhibbert-key&lt;/ds:KeyName&gt;
3231 </AttributeValue>
```

3232 In general, this method will not be adequate unless the structured data-type is quite simple.

3233 2. An `<AttributeSelector>` element MAY be used to select the contents of a leaf sub-  
3234 element of the structured data-type by means of an XPath expression. That value MAY  
3235 then be compared using one of the supported XACML functions appropriate for its primitive  
3236 data-type. This method requires support by the *PDP* for the optional XPath expressions  
3237 feature.

3238 3. An `<AttributeSelector>` element MAY be used to select any node in the structured  
3239 data-type by means of an XPath expression. This node MAY then be compared using one  
3240 of the XPath-based functions described in Section A.3. This method requires support by  
3241 the *PDP* for the optional XPath expressions and XPath functions features.

3242 See also Section 8.2.

### 3243 7.2.2. Attribute bags

3244 XACML defines implicit collections of its data-types. XACML refers to a collection of values that are  
3245 of a single data-type as a *bag*. *Bags* of data-types are needed because selections of nodes from  
3246 an XML *resource* or XACML request *context* may return more than one value.

3247 The `<AttributeSelector>` element uses an XPath expression to specify the selection of data  
3248 from an XML *resource*. The result of an XPath expression is termed a *node-set*, which contains all  
3249 the leaf nodes from the XML *resource* that match the predicate in the XPath expression. Based on  
3250 the various indexing functions provided in the XPath specification, it SHALL be implied that a  
3251 resultant node-set is the collection of the matching nodes. XACML also defines the  
3252 `<AttributeDesignator>` element to have the same matching methodology for *attributes* in the  
3253 XACML request *context*.

3254 The values in a *bag* are not ordered, and some of the values may be duplicates. There SHALL be  
3255 no notion of a *bag* containing *bags*, or a *bag* containing values of differing types. I.e. a *bag* in  
3256 XACML SHALL contain only values that are of the same data-type.

### 3257 7.2.3. Multivalued attributes

3258 If a single `<Attribute>` element in a request *context* contains multiple `<xacml-`  
3259 `context:AttributeValue>` child elements, then the *bag* of values resulting from evaluation of  
3260 the `<Attribute>` element MUST be identical to the *bag* of values that results from evaluating a  
3261 *context* in which each `<xacml-context:AttributeValue>` element appears in a separate  
3262 `<Attribute>` element, each carrying identical meta-data.

### 3263 7.2.4. Attribute Matching

3264 A *named attribute* includes specific criteria with which to match *attributes* in the *context*. An  
3265 *attribute* specifies an `AttributeId` and `DataType`, and a *named attribute* also specifies the  
3266 Issuer. A *named attribute* SHALL match an *attribute* if the values of their respective  
3267 `AttributeId`, `DataType` and optional Issuer attributes match within their particular element -  
3268 *subject*, *resource*, *action* or *environment* - of the *context*. The `AttributeId` of the *named*  
3269 *attribute* MUST match, by URI equality, the `AttributeId` of the corresponding *context attribute*.  
3270 The `DataType` of the *named attribute* MUST match, by URI equality, the `DataType` of the  
3271 corresponding *context attribute*. If Issuer is supplied in the *named attribute*, then it MUST

3272 match, using the `urn:oasis:names:tc:xacml:1.0:function:string-equal` function, the  
3273 Issuer of the corresponding **context attribute**. If Issuer is not supplied in the **named attribute**,  
3274 then the matching of the **context attribute** to the **named attribute** SHALL be governed by  
3275 AttributeId and DataType alone, regardless of the presence, absence, or actual value of  
3276 Issuer in the corresponding **context attribute**. In the case of an **attribute** selector, the matching  
3277 of the **attribute** to the **named attribute** SHALL be governed by the XPath expression and  
3278 DataType.

### 3279 **7.2.5. Attribute Retrieval**

3280 The **PDP** SHALL request the values of **attributes** in the request **context** from the **context handler**.  
3281 The **PDP** SHALL reference the **attributes** as if they were in a physical request **context** document,  
3282 but the **context handler** is responsible for obtaining and supplying the requested values by  
3283 whatever means it deems appropriate. The **context handler** SHALL return the values of  
3284 **attributes** that match the **attribute** designator or **attribute** selector and form them into a **bag** of  
3285 values with the specified data-type. If no **attributes** from the request **context** match, then the  
3286 **attribute** SHALL be considered missing. If the **attribute** is missing, then `MustBePresent`  
3287 governs whether the **attribute** designator or **attribute** selector returns an empty **bag** or an  
3288 "Indeterminate" result. If `MustBePresent` is "False" (default value), then a missing **attribute**  
3289 SHALL result in an empty **bag**. If `MustBePresent` is "True", then a missing **attribute** SHALL  
3290 result in "Indeterminate". This "Indeterminate" result SHALL be handled in accordance with the  
3291 specification of the encompassing expressions, **rules**, **policies** and **policy sets**. If the result is  
3292 "Indeterminate", then the `AttributeId`, `DataType` and `Issuer` of the **attribute** MAY be listed in  
3293 the **authorization decision** as described in Section 7.13. However, a **PDP** MAY choose not to  
3294 return such information for security reasons.

### 3295 **7.2.6. Environment Attributes**

3296 Standard **environment attributes** are listed in Section B.8. If a value for one of these **attributes** is  
3297 supplied in the **decision request**, then the **context handler** SHALL use that value. Otherwise, the  
3298 **context handler** SHALL supply a value. In the case of date and time **attributes**, the supplied  
3299 value SHALL have the semantics of the "date and time that apply to the **decision request**".

## 3300 **7.3. Expression evaluation**

3301 XACML specifies expressions in terms of the elements listed below, of which the `<Apply>` and  
3302 `<Condition>` elements recursively compose greater expressions. Valid expressions SHALL be  
3303 type correct, which means that the types of each of the elements contained within `<Apply>` and  
3304 `<Condition>` elements SHALL agree with the respective argument types of the function that is  
3305 named by the `FunctionId` attribute. The resultant type of the `<Apply>` or `<Condition>`  
3306 element SHALL be the resultant type of the function, which MAY be narrowed to a primitive data-  
3307 type, or a **bag** of a primitive data-type, by type-unification. XACML defines an evaluation result of  
3308 "Indeterminate", which is said to be the result of an invalid expression, or an operational error  
3309 occurring during the evaluation of the expression.

3310 XACML defines these elements to be in the substitution group of the `<Expression>` element:

- 3311 • `<xacml:AttributeValue>`
- 3312 • `<xacml:SubjectAttributeDesignator>`
- 3313 • `<xacml:ResourceAttributeDesignator>`
- 3314 • `<xacml:ActionAttributeDesignator>`



- 3315 • <xacml:EnvironmentAttributeDesignator>
- 3316 • <xacml:AttributeSelector>
- 3317 • <xacml:Apply>
- 3318 • <xacml:Condition>
- 3319 • <xacml:Function>
- 3320 • <xacml:VariableReference>

## 3321 **7.4. Arithmetic evaluation**

3322 IEEE 754 [IEEE 754] specifies how to evaluate arithmetic functions in a context, which specifies  
 3323 defaults for precision, rounding, etc. XACML SHALL use this specification for the evaluation of all  
 3324 integer and double functions relying on the *Extended Default Context*, enhanced with double  
 3325 precision:

- 3326 flags - all set to 0
- 3327 trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the “division-by-zero” trap  
 3328 enabler, which SHALL be set to 1
- 3329 precision - is set to the designated double precision
- 3330 rounding - is set to round-half-even (IEEE 854 §4.1)

## 3331 **7.5. Match evaluation**

3332 **Attribute** matching elements appear in the <Target> element of **rules**, **policies** and **policy sets**.  
 3333 They are the following:

- 3334 <SubjectMatch>
- 3335 <ResourceMatch>
- 3336 <ActionMatch>
- 3337 <EnvironmentMatch>

3338 These elements represent Boolean expressions over **attributes** of the **subject**, **resource**, **action**  
 3339 and **environment**, respectively. A matching element contains a MatchId attribute that specifies  
 3340 the function to be used in performing the match evaluation, an <xacml:AttributeValue> and an  
 3341 <AttributeDesignator> or <AttributeSelector> element that specifies the **attribute** in the  
 3342 **context** that is to be matched against the specified value.

3343 The MatchId attribute SHALL specify a function that compares two arguments, returning a result  
 3344 type of "http://www.w3.org/2001/XMLSchema#boolean". The **attribute** value specified in the  
 3345 matching element SHALL be supplied to the MatchId function as its first argument. An element of  
 3346 the **bag** returned by the <AttributeDesignator> or <AttributeSelector> element SHALL  
 3347 be supplied to the MatchId function as its second argument, as explained below. The DataType  
 3348 of the <xacml:AttributeValue> SHALL match the data-type of the first argument expected by  
 3349 the MatchId function. The DataType of the <AttributeDesignator> or  
 3350 <AttributeSelector> element SHALL match the data-type of the second argument expected  
 3351 by the MatchId function.

3352 The XACML standard functions that meet the requirements for use as a `MatchId` attribute value  
3353 are:

3354 `urn:oasis:names:tc:xacml:2.0:function:-type-equal`

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

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

3357 `urn:oasis:names:tc:xacml:2.0:function:-type-less-than`

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

3359 `urn:oasis:names:tc:xacml:2.0:function:-type-match`

3360 In addition, functions that are strictly within an extension to XACML MAY appear as a value for the  
3361 `MatchId` attribute, and those functions MAY use data-types that are also extensions, so long as  
3362 the extension function returns a Boolean result and takes two single base types as its inputs. The  
3363 function used as the value for the `MatchId` attribute SHOULD be easily indexable. Use of non-  
3364 indexable or complex functions may prevent efficient evaluation of **decision requests**.

3365 The evaluation semantics for a matching element is as follows. If an operational error were to  
3366 occur while evaluating the `<AttributeDesignator>` or `<AttributeSelector>` element, then  
3367 the result of the entire expression SHALL be "Indeterminate". If the `<AttributeDesignator>` or  
3368 `<AttributeSelector>` element were to evaluate to an empty **bag**, then the result of the  
3369 expression SHALL be "False". Otherwise, the `MatchId` function SHALL be applied between the  
3370 `<xacml:AttributeValue>` and each element of the **bag** returned from the  
3371 `<AttributeDesignator>` or `<AttributeSelector>` element. If at least one of those function  
3372 applications were to evaluate to "True", then the result of the entire expression SHALL be "True".  
3373 Otherwise, if at least one of the function applications results in "Indeterminate", then the result  
3374 SHALL be "Indeterminate". Finally, if all function applications evaluate to "False", then the result of  
3375 the entire expression SHALL be "False".

3376 It is also possible to express the semantics of a **target** matching element in a **condition**. For  
3377 instance, the **target** match expression that compares a "subject-name" starting with the name  
3378 "John" can be expressed as follows:

```
3379 <SubjectMatch  
3380 MatchId="urn:oasis:names:tc:xacml:1.0:function:regexp-string-match">  
3381   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3382     John.*  
3383   </AttributeValue>  
3384   <SubjectAttributeDesignator  
3385     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3386     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3387 </SubjectMatch>
```

3388 Alternatively, the same match semantics can be expressed as an `<Apply>` element in a **condition**  
3389 by using the "urn:oasis:names:tc:xacml:1.0:function:any-of" function, as follows:

```
3390 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">  
3391   <Function  
3392     FunctionId="urn:oasis:names:tc:xacml:1.0:function:regexp-string-match"/>  
3393   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3394     John.*  
3395   </AttributeValue>  
3396   <SubjectAttributeDesignator  
3397     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3398     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3399 </Apply>
```

3400

## 7.6. Target evaluation

3401 The **target** value SHALL be "Match" if the **subjects, resources, actions** and **environments**  
 3402 specified in the **target** all match values in the request **context**. If any one of the **subjects,**  
 3403 **resources, actions** and **environments** specified in the **target** are "Indeterminate", then the **target**  
 3404 SHALL be "Indeterminate". Otherwise, the **target** SHALL be "No match". The **target** match table is  
 3405 shown in Table 1.  
 3406

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"

3407

Table 1 - Target match table

3408 The **subjects, resources, actions** and **environments** SHALL match values in the request **context**  
 3409 if at least one of their <Subject>, <Resource>, <Action> or <Environment> elements,  
 3410 respectively, matches a value in the request **context**. The **subjects** match table is shown in Table  
 3411 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"

3412

Table 2 - Subjects match table

3413 A **subject, resource, action** or **environment** SHALL match a value in the request **context** if the  
 3414 value of all its <SubjectMatch>, <ResourceMatch>, <ActionMatch> or  
 3415 <EnvironmentMatch> elements, respectively, are "True".

3416 The **subject** match table is shown in Table 3. The **resource, action** and **environment** match  
 3417 tables are analogous.

3418

3419

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

3420

Table 3 - Subject match table

3421

## 7.7. VariableReference Evaluation

3422 The <VariableReference> element references a single <VariableDefinition> element  
 3423 contained within the same <Policy> element. A <VariableReference> that does not  
 3424 reference a particular <VariableDefinition> element within the encompassing <Policy>  
 3425 element is called an undefined reference. **Policies** with undefined references are invalid.

3426 In any place where a <VariableReference> occurs, it has the effect as if the text of the  
 3427 <Expression> element defined in the <VariableDefinition> element replaces the  
 3428 <VariableReference> element. Any evaluation scheme that preserves this semantic is  
 3429 acceptable. For instance, the expression in the <VariableDefinition> element may be  
 3430 evaluated to a particular value and cached for multiple references without consequence. (I.e. the  
 3431 value of an <Expression> element remains the same for the entire policy evaluation.) This  
 3432 characteristic is one of the benefits of XACML being a declarative language.

3433

## 7.8. Condition evaluation

3434 The **condition** value SHALL be "True" if the <Condition> element is absent, or if it evaluates to  
 3435 "True". Its value SHALL be "False" if the <Condition> element evaluates to "False". The  
 3436 **condition** value SHALL be "Indeterminate", if the expression contained in the <Condition>  
 3437 element evaluates to "Indeterminate".

3438

## 7.9. Rule evaluation

3439 A **rule** has a value that can be calculated by evaluating its contents. **Rule** evaluation involves  
 3440 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"

3441

Table 4 - Rule truth table

3442 If the **target** value is "No-match" or "Indeterminate" then the **rule** value SHALL be "NotApplicable"  
 3443 or "Indeterminate", respectively, regardless of the value of the **condition**. For these cases,  
 3444 therefore, the **condition** need not be evaluated.

3445 If the **target** value is "Match" and the **condition** value is "True", then the **effect** specified in the  
 3446 enclosing <Rule> element SHALL determine the **rule's** value.

## 3447 7.10. Policy evaluation

3448 The value of a **policy** SHALL be determined only by its contents, considered in relation to the  
 3449 contents of the request **context**. A **policy's** value SHALL be determined by evaluation of the  
 3450 **policy's target** and **rules**.

3451 The **policy's target** SHALL be evaluated to determine the applicability of the **policy**. If the **target**  
 3452 evaluates to "Match", then the value of the **policy** SHALL be determined by evaluation of the  
 3453 **policy's rules**, according to the specified **rule-combining algorithm**. If the **target** evaluates to  
 3454 "No-match", then the value of the **policy** SHALL be "NotApplicable". If the **target** evaluates to  
 3455 "Indeterminate", then the value of the **policy** SHALL be "Indeterminate".

3456 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 <b>rule-combining algorithm</b>
"Match"	All rule values are "NotApplicable"	"NotApplicable"
"Match"	At least one rule value is "Indeterminate"	Specified by the <b>rule-combining algorithm</b>
"No-match"	Don't care	"NotApplicable"
"Indeterminate"	Don't care	"Indeterminate"

3457 **Table 5 - Policy truth table**

3458 A **rules** value of "At least one rule value is its Effect" means either that the <Rule> element is  
 3459 absent, or one or more of the **rules** contained in the **policy** is applicable to the **decision request**  
 3460 (i.e., it returns the value of its "Effect"; see Section 7.9). A **rules** value of "All rule values are  
 3461 'NotApplicable'" SHALL be used if no **rule** contained in the **policy** is applicable to the request and if  
 3462 no **rule** contained in the **policy** returns a value of "Indeterminate". If no **rule** contained in the  
 3463 **policy** is applicable to the request, but one or more **rule** returns a value of "Indeterminate", then the  
 3464 **rules** SHALL evaluate to "At least one rule value is 'Indeterminate'".

3465 If the **target** value is "No-match" or "Indeterminate" then the **policy** value SHALL be  
 3466 "NotApplicable" or "Indeterminate", respectively, regardless of the value of the **rules**. For these  
 3467 cases, therefore, the **rules** need not be evaluated.

3468 If the **target** value is "Match" and the **rule** value is "At least one rule value is it's Effect" or "At least  
 3469 one rule value is 'Indeterminate'", then the **rule-combining algorithm** specified in the **policy**  
 3470 SHALL determine the **policy** value.

3471 Note that none of the **rule-combining algorithms** defined by XACML 2.0 take parameters.  
 3472 However, non-standard **combining algorithms** MAY take parameters. In such a case, the values  
 3473 of these parameters associated with the **rules**, MUST be taken into account when evaluating the  
 3474 **policy**. The parameters and their types should be defined in the specification of the **combining**  
 3475 **algorithm**. If the implementation supports combiner parameters and if combiner parameters are

3476 present in a **policy**, then the parameter values MUST be supplied to the **combining algorithm**  
 3477 implementation.

## 3478 7.11. Policy Set evaluation

3479 The value of a **policy set** SHALL be determined by its contents, considered in relation to the  
 3480 contents of the **request context**. A **policy set's** value SHALL be determined by evaluation of the  
 3481 **policy set's target, policies** and **policy sets**, according to the specified **policy-combining**  
 3482 **algorithm**.

3483 The **policy set's target** SHALL be evaluated to determine the applicability of the **policy set**. If the  
 3484 **target** evaluates to "Match" then the value of the **policy set** SHALL be determined by evaluation of  
 3485 the **policy set's policies** and **policy sets**, according to the specified **policy-combining algorithm**.  
 3486 If the **target** evaluates to "No-match", then the value of the **policy set** shall be "NotApplicable". If  
 3487 the **target** evaluates to "Indeterminate", then the value of the **policy set** SHALL be "Indeterminate".

3488 The **policy set** truth table is shown in Table 6.

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

3489 **Table 6 – Policy set truth table**

3490 A **policies** value of "At least one policy value is its **Decision**" SHALL be used if there are no  
 3491 contained or referenced **policies** or **policy sets**, or if one or more of the **policies** or **policy sets**  
 3492 contained in or referenced by the **policy set** is applicable to the **decision request** (i.e., returns a  
 3493 value determined by its **combining algorithm**) A **policies** value of "All policy values are  
 3494 'NotApplicable'" SHALL be used if no **policy** or **policy set** contained in or referenced by the **policy**  
 3495 **set** is applicable to the request and if no **policy** or **policy set** contained in or referenced by the  
 3496 **policy set** returns a value of "Indeterminate". If no **policy** or **policy set** contained in or referenced  
 3497 by the **policy set** is applicable to the request but one or more **policy** or **policy set** returns a value  
 3498 of "Indeterminate", then the **policies** SHALL evaluate to "At least one policy value is  
 3499 'Indeterminate'".

3500 If the **target** value is "No-match" or "Indeterminate" then the **policy set** value SHALL be  
 3501 "NotApplicable" or "Indeterminate", respectively, regardless of the value of the **policies**. For these  
 3502 cases, therefore, the **policies** need not be evaluated.

3503 If the **target** value is "Match" and the **policies** value is "At least one policy value is its **Decision**" or  
 3504 "At least one policy value is 'Indeterminate'", then the **policy-combining algorithm** specified in the  
 3505 **policy set** SHALL determine the **policy set** value.

3506 Note that none of the **policy-combining algorithms** defined by XACML 2.0 take parameters.  
 3507 However, non-standard **combining algorithms** MAY take parameters. In such a case, the values



3508 of these parameters associated with the **policies**, MUST be taken into account when evaluating the  
3509 **policy set**. The parameters and their types should be defined in the specification of the  
3510 **combining algorithm**. If the implementation supports combiner parameters and if combiner  
3511 parameters are present in a **policy**, then the parameter values MUST be supplied to the  
3512 **combining algorithm** implementation.

## 3513 7.12. Hierarchical resources

3514 It is often the case that a **resource** is organized as a hierarchy (e.g. file system, XML document).  
3515 XACML provides several optional mechanisms for supporting hierarchical resources. These are  
3516 described in the XACML Profile for Hierarchical Resources [HIER] and in the XACML Profile for  
3517 Requests for Multiple Resources [MULT].

## 3518 7.13. Authorization decision

3519 In relation to a particular **decision request**, the **PDP** is defined by a **policy-combining algorithm**  
3520 and a set of **policies** and/or **policy sets**. The **PDP** SHALL return a response **context** as if it had  
3521 evaluated a single **policy set** consisting of this **policy-combining algorithm** and the set of  
3522 **policies** and/or **policy sets**.

3523 The **PDP** MUST evaluate the **policy set** as specified in Sections 5 and 7. The **PDP** MUST return a  
3524 response **context**, with one <Decision> element of value "Permit", "Deny", "Indeterminate" or  
3525 "NotApplicable".

3526 If the **PDP** cannot make a decision, then an "Indeterminate" <Decision> element SHALL be  
3527 returned.

## 3528 7.14. Obligations

3529 A **policy** or **policy set** may contain one or more **obligations**. When such a **policy** or **policy set** is  
3530 evaluated, an **obligation** SHALL be passed up to the next level of evaluation (the enclosing or  
3531 referencing **policy**, **policy set** or **authorization decision**) only if the **effect** of the **policy** or **policy**  
3532 **set** being evaluated matches the value of the FulfillOn attribute of the **obligation**.

3533  
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  
3539 If the **PDP's** evaluation is viewed as a tree of **policy sets** and **policies**, each of which returns  
3540 "Permit" or "Deny", then the set of **obligations** returned by the **PDP** to the **PEP** will include only the  
3541 **obligations** associated with those paths where the **effect** at each level of evaluation is the same as  
3542 the **effect** being returned by the **PDP**. In situations where any lack of determinism is unacceptable,  
3543 a deterministic combining algorithm, such as ordered-deny-overrides, should be used.

3544 Also, see Section 7.1.

## 3545 7.15. Exception handling

3546 XACML specifies behaviour for the **PDP** in the following situations.

3547

### 7.15.1. Unsupported functionality

3548 If the **PDP** attempts to evaluate a **policy set** or **policy** that contains an optional element type or  
3549 function that the **PDP** does not support, then the **PDP** SHALL return a <Decision> value of  
3550 "Indeterminate". If a <StatusCode> element is also returned, then its value SHALL be  
3551 "urn:oasis:names:tc:xacml:1.0:status:syntax-error" in the case of an unsupported element type, and  
3552 "urn:oasis:names:tc:xacml:1.0:status:processing-error" in the case of an unsupported function.

3553

### 7.15.2. Syntax and type errors

3554 If a **policy** that contains invalid syntax is evaluated by the XACML **PDP** at the time a **decision**  
3555 **request** is received, then the result of that **policy** SHALL be "Indeterminate" with a StatusCode  
3556 value of "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

3557 If a **policy** that contains invalid static data-types is evaluated by the XACML **PDP** at the time a  
3558 **decision request** is received, then the result of that **policy** SHALL be "Indeterminate" with a  
3559 StatusCode value of "urn:oasis:names:tc:xacml:1.0:status:processing-error".

3560

### 7.15.3. Missing attributes

3561 The absence of matching **attributes** in the request **context** for any of the **attribute** designators or  
3562 selectors that are found in the **policy** SHALL result in a <Decision> element containing the  
3563 "Indeterminate" value, as described in Sections 5.37 and 5.42. If, in this case, and a status code is  
3564 supplied, then the value

3565 "urn:oasis:names:tc:xacml:1.0:status:missing-attribute"

3566 SHALL be used, to indicate that more information is needed in order for a definitive decision to be  
3567 rendered. In this case, the <Status> element MAY list the names and data-types of any  
3568 **attributes** of the **subjects**, **resource**, **action** or **environment** that are needed by the **PDP** to refine  
3569 its decision (see Section 6.16). A **PEP** MAY resubmit a refined request **context** in response to a  
3570 <Decision> element contents of "Indeterminate" with a status code of

3571 "urn:oasis:names:tc:xacml:1.0:missing-attribute"

3572 by adding **attribute** values for the **attribute** names that were listed in the previous response. When  
3573 the **PDP** returns a <Decision> element contents of "Indeterminate", with a status code of

3574 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3575 it MUST NOT list the names and data-types of any **attribute** of the **subject**, **resource**, **action** or  
3576 **environment** for which values were supplied in the original request. Note, this requirement forces  
3577 the **PDP** to eventually return an **authorization decision** of "Permit", "Deny" or "Indeterminate" with  
3578 some other status code, in response to successively-refined requests.

---

3579

## 8. XACML extensibility points (non-normative)

3580 This section describes the points within the XACML model and schema where extensions can be  
3581 added

## 3582 **8.1. Extensible XML attribute types**

3583 The following XML attributes have values that are URIs. These may be extended by the creation of  
3584 new URIs associated with new semantics for these attributes.

3585 `AttributeId`,

3586 `DataType`,

3587 `FunctionId`,

3588 `MatchId`,

3589 `ObligationId`,

3590 `PolicyCombiningAlgId`,

3591 `RuleCombiningAlgId`,

3592 `StatusCode`,

3593 `SubjectCategory`.

3594 See Section 5 for definitions of these attribute types.

## 3595 **8.2. Structured attributes**

3596 `<xacml:AttributeValue>` and `<xacml-context:AttributeValue>` elements MAY  
3597 contain an instance of a structured XML data-type. Section 7.2.1 describes a number of standard  
3598 techniques to identify data items within such a structured attribute. Listed here are some additional  
3599 techniques that require XACML extensions.

3600 1. For a given structured data-type, a community of XACML users MAY define new attribute  
3601 identifiers for each leaf sub-element of the structured data-type that has a type conformant  
3602 with one of the XACML-defined primitive data-types. Using these new attribute identifiers,  
3603 the **PEPs** or **context handlers** used by that community of users can flatten instances of  
3604 the structured data-type into a sequence of individual `<Attribute>` elements. Each such  
3605 `<Attribute>` element can be compared using the XACML-defined functions. Using this  
3606 method, the structured data-type itself never appears in an `<xacml-`  
3607 `context:AttributeValue>` element.

3608 2. A community of XACML users MAY define a new function that can be used to compare a  
3609 value of the structured data-type against some other value. This method may only be used  
3610 by **PDPs** that support the new function.

---

## 3611 **9. Security and privacy considerations (non-** 3612 **normative)**

3613 This section identifies possible security and privacy compromise scenarios that should be  
3614 considered when implementing an XACML-based system. The section is informative only. It is left  
3615 to the implementer to decide whether these compromise scenarios are practical in their  
3616 environment and to select appropriate safeguards.

3617

## 9.1. Threat model

3618 We assume here that the adversary has access to the communication channel between the  
3619 XACML actors and is able to interpret, insert, delete and modify messages or parts of messages.

3620 Additionally, an actor may use information from a former message maliciously in subsequent  
3621 transactions. It is further assumed that **rules** and **policies** are only as reliable as the actors that  
3622 create and use them. Thus it is incumbent on each actor to establish appropriate trust in the other  
3623 actors upon which it relies. Mechanisms for trust establishment are outside the scope of this  
3624 specification.

3625 The messages that are transmitted between the actors in the XACML model are susceptible to  
3626 attack by malicious third parties. Other points of vulnerability include the **PEP**, the **PDP** and the  
3627 **PAP**. While some of these entities are not strictly within the scope of this specification, their  
3628 compromise could lead to the compromise of **access control** enforced by the **PEP**.

3629 It should be noted that there are other components of a distributed system that may be  
3630 compromised, such as an operating system and the domain-name system (DNS) that are outside  
3631 the scope of this discussion of threat models. Compromise in these components may also lead to a  
3632 policy violation.

3633 The following sections detail specific compromise scenarios that may be relevant to an XACML  
3634 system.

3635

### 9.1.1. Unauthorized disclosure

3636 XACML does not specify any inherent mechanisms to protect the confidentiality of the messages  
3637 exchanged between actors. Therefore, an adversary could observe the messages in transit. Under  
3638 certain security policies, disclosure of this information is a violation. Disclosure of **attributes** or the  
3639 types of **decision requests** that a **subject** submits may be a breach of privacy policy. In the  
3640 commercial sector, the consequences of unauthorized disclosure of personal data may range from  
3641 embarrassment to the custodian to imprisonment and large fines in the case of medical or financial  
3642 data.

3643 Unauthorized disclosure is addressed by confidentiality safeguards.

3644

### 9.1.2. Message replay

3645 A message replay attack is one in which the adversary records and replays legitimate messages  
3646 between XACML actors. This attack may lead to denial of service, the use of out-of-date  
3647 information or impersonation.

3648 Prevention of replay attacks requires the use of message freshness safeguards.

3649 Note that encryption of the message does not mitigate a replay attack since the message is simply  
3650 replayed and does not have to be understood by the adversary.

3651

### 9.1.3. Message insertion

3652 A message insertion attack is one in which the adversary inserts messages in the sequence of  
3653 messages between XACML actors.

3654 The solution to a message insertion attack is to use mutual authentication and message sequence  
3655 integrity safeguards between the actors. It should be noted that just using SSL mutual  
3656 authentication is not sufficient. This only proves that the other party is the one identified by the

3657 subject of the X.509 certificate. In order to be effective, it is necessary to confirm that the certificate  
3658 subject is authorized to send the message.

#### 3659 **9.1.4. Message deletion**

3660 A message deletion attack is one in which the adversary deletes messages in the sequence of  
3661 messages between XACML actors. Message deletion may lead to denial of service. However, a  
3662 properly designed XACML system should not render an incorrect authorization decision as a result  
3663 of a message deletion attack.

3664 The solution to a message deletion attack is to use message sequence integrity safeguards  
3665 between the actors.

#### 3666 **9.1.5. Message modification**

3667 If an adversary can intercept a message and change its contents, then they may be able to alter an  
3668 **authorization decision**. A message integrity safeguard can prevent a successful message  
3669 modification attack.

#### 3670 **9.1.6. NotApplicable results**

3671 A result of "NotApplicable" means that the **PDP** could not locate a **policy** whose **target** matched  
3672 the information in the **decision request**. In general, it is highly recommended that a "Deny" **effect**  
3673 **policy** be used, so that when a **PDP** would have returned "NotApplicable", a result of "Deny" is  
3674 returned instead.

3675 In some security models, however, such as those found in many Web Servers, an **authorization**  
3676 **decision** of "NotApplicable" is treated as equivalent to "Permit". There are particular security  
3677 considerations that must be taken into account for this to be safe. These are explained in the  
3678 following paragraphs.

3679 If "NotApplicable" is to be treated as "Permit", it is vital that the matching algorithms used by the  
3680 **policy** to match elements in the **decision request** be closely aligned with the data syntax used by  
3681 the applications that will be submitting the **decision request**. A failure to match will result in  
3682 "NotApplicable" and be treated as "Permit". So an unintended failure to match may allow  
3683 unintended **access**.

3684 Commercial http responders allow a variety of syntaxes to be treated equivalently. The "%" can be  
3685 used to represent characters by hex value. The URL path "/../" provides multiple ways of specifying  
3686 the same value. Multiple character sets may be permitted and, in some cases, the same printed  
3687 character can be represented by different binary values. Unless the matching algorithm used by  
3688 the policy is sophisticated enough to catch these variations, unintended access may be permitted.

3689 It may be safe to treat "NotApplicable" as "Permit" only in a closed environment where all  
3690 applications that formulate a **decision request** can be guaranteed to use the exact syntax  
3691 expected by the **policies**. In a more open environment, where **decision requests** may be received  
3692 from applications that use any legal syntax, it is strongly recommended that "NotApplicable" NOT  
3693 be treated as "Permit" unless matching rules have been very carefully designed to match all  
3694 possible applicable inputs, regardless of syntax or type variations. Note, however, that according to  
3695 Section 7.1, a **PEP** must deny **access** unless it receives an explicit "Permit" **authorization**  
3696 **decision**.

#### 3697 **9.1.7. Negative rules**

3698 A negative **rule** is one that is based on a **predicate** not being "True". If not used with care,  
3699 negative **rules** can lead to a policy violation, therefore some authorities recommend that they not

3700 be used. However, negative **rules** can be extremely efficient in certain cases, so XACML has  
3701 chosen to include them. Nevertheless, it is recommended that they be used with care and avoided  
3702 if possible.

3703 A common use for negative **rules** is to deny **access** to an individual or subgroup when their  
3704 membership in a larger group would otherwise permit them access. For example, we might want to  
3705 write a **rule** that allows all Vice Presidents to see the unpublished financial data, except for Joe,  
3706 who is only a Ceremonial Vice President and can be indiscreet in his communications. If we have  
3707 complete control over the administration of **subject attributes**, a superior approach would be to  
3708 define "Vice President" and "Ceremonial Vice President" as distinct groups and then define **rules**  
3709 accordingly. However, in some environments this approach may not be feasible. (It is worth noting  
3710 in passing that, generally speaking, referring to individuals in **rules** does not scale well. Generally,  
3711 shared **attributes** are preferred.)

3712 If not used with care, negative **rules** can lead to policy violation in two common cases. They are:  
3713 when **attributes** are suppressed and when the base group changes. An example of suppressed  
3714 **attributes** would be if we have a policy that **access** should be permitted, *unless* the **subject** is a  
3715 credit risk. If it is possible that the **attribute** of being a credit risk may be unknown to the **PDP** for  
3716 some reason, then unauthorized **access** may result. In some environments, the **subject** may be  
3717 able to suppress the publication of **attributes** by the application of privacy controls, or the server or  
3718 repository that contains the information may be unavailable for accidental or intentional reasons.

3719 An example of a changing base group would be if there is a policy that everyone in the engineering  
3720 department may change software source code, except for secretaries. Suppose now that the  
3721 department was to merge with another engineering department and the intent is to maintain the  
3722 same policy. However, the new department also includes individuals identified as administrative  
3723 assistants, who ought to be treated in the same way as secretaries. Unless the policy is altered,  
3724 they will unintentionally be permitted to change software source code. Problems of this type are  
3725 easy to avoid when one individual administers all **policies**, but when administration is distributed,  
3726 as XACML allows, this type of situation must be explicitly guarded against.

## 3727 **9.2. Safeguards**

### 3728 **9.2.1. Authentication**

3729 Authentication provides the means for one party in a transaction to determine the identity of the  
3730 other party in the transaction. Authentication may be in one direction, or it may be bilateral.

3731 Given the sensitive nature of **access control** systems, it is important for a **PEP** to authenticate the  
3732 identity of the **PDP** to which it sends **decision requests**. Otherwise, there is a risk that an  
3733 adversary could provide false or invalid **authorization decisions**, leading to a policy violation.

3734 It is equally important for a **PDP** to authenticate the identity of the **PEP** and assess the level of trust  
3735 to determine what, if any, sensitive data should be passed. One should keep in mind that even  
3736 simple "Permit" or "Deny" responses could be exploited if an adversary were allowed to make  
3737 unlimited requests to a **PDP**.

3738 Many different techniques may be used to provide authentication, such as co-located code, a  
3739 private network, a VPN or digital signatures. Authentication may also be performed as part of the  
3740 communication protocol used to exchange the **contexts**. In this case, authentication may be  
3741 performed either at the message level or at the session level.

### 3742 **9.2.2. Policy administration**

3743 If the contents of **policies** are exposed outside of the **access control** system, potential **subjects**  
3744 may use this information to determine how to gain unauthorized **access**.



3745 To prevent this threat, the repository used for the storage of **policies** may itself require **access**  
3746 **control**. In addition, the <Status> element should be used to return values of missing **attributes**  
3747 only when exposure of the identities of those **attributes** will not compromise security.

### 3748 **9.2.3. Confidentiality**

3749 Confidentiality mechanisms ensure that the contents of a message can be read only by the desired  
3750 recipients and not by anyone else who encounters the message while it is in transit. There are two  
3751 areas in which confidentiality should be considered: one is confidentiality during transmission; the  
3752 other is confidentiality within a <Policy> element.

#### 3753 **9.2.3.1. Communication confidentiality**

3754 In some environments it is deemed good practice to treat all data within an **access control** system  
3755 as confidential. In other environments, **policies** may be made freely available for distribution,  
3756 inspection and audit. The idea behind keeping **policy** information secret is to make it more difficult  
3757 for an adversary to know what steps might be sufficient to obtain unauthorized **access**. Regardless  
3758 of the approach chosen, the security of the **access control** system should not depend on the  
3759 secrecy of the **policy**.

3760 Any security considerations related to transmitting or exchanging XACML <Policy> elements are  
3761 outside the scope of the XACML standard. While it is often important to ensure that the integrity  
3762 and confidentiality of <Policy> elements is maintained when they are exchanged between two  
3763 parties, it is left to the implementers to determine the appropriate mechanisms for their  
3764 environment.

3765 Communications confidentiality can be provided by a confidentiality mechanism, such as SSL.  
3766 Using a point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points  
3767 is compromised.

#### 3768 **9.2.3.2. Statement level confidentiality**

3769 In some cases, an implementation may want to encrypt only parts of an XACML <Policy>  
3770 element.

3771 The XML Encryption Syntax and Processing Candidate Recommendation from W3C can be used  
3772 to encrypt all or parts of an XML document. This specification is recommended for use with  
3773 XACML.

3774 It should go without saying that if a repository is used to facilitate the communication of cleartext  
3775 (i.e., unencrypted) **policy** between the **PAP** and **PDP**, then a secure repository should be used to  
3776 store this sensitive data.

### 3777 **9.2.4. Policy integrity**

3778 The XACML **policy**, used by the **PDP** to evaluate the request **context**, is the heart of the system.  
3779 Therefore, maintaining its integrity is essential. There are two aspects to maintaining the integrity of  
3780 the **policy**. One is to ensure that <Policy> elements have not been altered since they were  
3781 originally created by the **PAP**. The other is to ensure that <Policy> elements have not been  
3782 inserted or deleted from the set of **policies**.

3783 In many cases, both aspects can be achieved by ensuring the integrity of the actors and  
3784 implementing session-level mechanisms to secure the communication between actors. The  
3785 selection of the appropriate mechanisms is left to the implementers. However, when **policy** is  
3786 distributed between organizations to be acted on at a later time, or when the **policy** travels with the

3787 protected resource, it would be useful to sign the **policy**. In these cases, the XML Signature  
3788 Syntax and Processing standard from W3C is recommended to be used with XACML.

3789 Digital signatures should only be used to ensure the integrity of the statements. Digital signatures  
3790 should not be used as a method of selecting or evaluating **policy**. That is, the **PDP** should not  
3791 request a **policy** based on who signed it or whether or not it has been signed (as such a basis for  
3792 selection would, itself, be a matter of policy). However, the **PDP** must verify that the key used to  
3793 sign the **policy** is one controlled by the purported issuer of the **policy**. The means to do this are  
3794 dependent on the specific signature technology chosen and are outside the scope of this document.

### 3795 **9.2.5. Policy identifiers**

3796 Since **policies** can be referenced by their identifiers, it is the responsibility of the **PAP** to ensure  
3797 that these are unique. Confusion between identifiers could lead to misidentification of the  
3798 **applicable policy**. This specification is silent on whether a **PAP** must generate a new identifier  
3799 when a **policy** is modified or may use the same identifier in the modified **policy**. This is a matter of  
3800 administrative practice. However, care must be taken in either case. If the identifier is reused,  
3801 there is a danger that other **policies** or **policy sets** that reference it may be adversely affected.  
3802 Conversely, if a new identifier is used, these other **policies** may continue to use the prior **policy**,  
3803 unless it is deleted. In either case the results may not be what the **policy** administrator intends.

### 3804 **9.2.6. Trust model**

3805 Discussions of authentication, integrity and confidentiality safeguards necessarily assume an  
3806 underlying trust model: how can one actor come to believe that a given key is uniquely associated  
3807 with a specific, identified actor so that the key can be used to encrypt data for that actor or verify  
3808 signatures (or other integrity structures) from that actor? Many different types of trust model exist,  
3809 including strict hierarchies, distributed authorities, the Web, the bridge and so on.

3810 It is worth considering the relationships between the various actors of the **access control** system in  
3811 terms of the interdependencies that do and do not exist.

- 3812 • None of the entities of the authorization system are dependent on the **PEP**. They may  
3813 collect data from it, for example authentication data, but are responsible for verifying it  
3814 themselves.
- 3815 • The correct operation of the system depends on the ability of the **PEP** to actually enforce  
3816 **policy** decisions.
- 3817 • The **PEP** depends on the **PDP** to correctly evaluate **policies**. This in turn implies that the  
3818 **PDP** is supplied with the correct inputs. Other than that, the **PDP** does not depend on the  
3819 **PEP**.
- 3820 • The **PDP** depends on the **PAP** to supply appropriate policies. The **PAP** is not dependent  
3821 on other components.

### 3822 **9.2.7. Privacy**

3823 It is important to be aware that any transactions that occur with respect to **access control** may  
3824 reveal private information about the actors. For example, if an XACML **policy** states that certain  
3825 data may only be read by **subjects** with "Gold Card Member" status, then any transaction in which  
3826 a **subject** is permitted **access** to that data leaks information to an adversary about the **subject's**  
3827 status. Privacy considerations may therefore lead to encryption and/or to access control  
3828 requirements surrounding the enforcement of XACML **policy** instances themselves: confidentiality-  
3829 protected channels for the request/response protocol messages, protection of **subject attributes** in  
3830 storage and in transit, and so on.

3831 Selection and use of privacy mechanisms appropriate to a given environment are outside the scope  
3832 of XACML. The decision regarding whether, how and when to deploy such mechanisms is left to  
3833 the implementers associated with the environment.

---

## 3834 10. Conformance (normative)

### 3835 10.1. Introduction

3836 The XACML specification addresses the following aspect of conformance:

3837 The XACML specification defines a number of functions, etc. that have somewhat special  
3838 application, therefore they are not required to be implemented in an implementation that claims to  
3839 conform with the OASIS standard.

### 3840 10.2. Conformance tables

3841 This section lists those portions of the specification that **MUST** be included in an implementation of  
3842 a **PDP** that claims to conform with XACML v2.0. A set of test cases has been created to assist in  
3843 this process. These test cases are hosted by Sun Microsystems and can be located from the  
3844 XACML Web page. The site hosting the test cases contains a full description of the test cases and  
3845 how to execute them.

3846 Note: "M" means mandatory-to-implement. "O" means optional.

#### 3847 10.2.1. Schema elements

3848 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

3849 **10.2.2. Identifier Prefixes**

3850 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

3851 **10.2.3. Algorithms**

3852 The implementation MUST include the rule- and policy-combining algorithms associated with the  
 3853 following identifiers that are marked "M".

3854

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

3855 **10.2.4. Status Codes**

3856 Implementation support for the <StatusCode> element is optional, but if the element is supported,  
3857 then the following status codes must be supported and must be used in the way XACML has  
3858 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

3859 **10.2.5. Attributes**

3860 The implementation MUST support the **attributes** associated with the following identifiers as  
3861 specified by XACML. If values for these **attributes** are not present in the **decision request**, then  
3862 their values MUST be supplied by the **context handler**. So, unlike most other **attributes**, their  
3863 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

3864 **10.2.6. Identifiers**

3865 The implementation MUST use the **attributes** associated with the following identifiers in the way  
3866 XACML has defined. This requirement pertains primarily to implementations of a **PAP** or **PEP** that  
3867 uses XACML, since the semantics of the attributes are transparent to the **PDP**.

3868  
3869  
3870  
3871

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

3872 **10.2.7. Data-types**

3873 The implementation MUST support the data-types associated with the following identifiers marked  
3874 "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

3875 **10.2.8. Functions**

3876 The implementation MUST properly process those functions associated with the identifiers marked  
3877 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

3878

## 3879 11. References

3880 [DS] D. Eastlake et al., *XML-Signature Syntax and Processing*,  
3881 <http://www.w3.org/TR/xmldsig-core/>, World Wide Web Consortium.

3882 [Hancock] Hancock, "Polymorphic Type Checking", in Simon L. Peyton Jones,  
3883 "Implementation of Functional Programming Languages", Section 8,  
3884 Prentice-Hall International, 1987

3885 [Haskell] Haskell, a purely functional language. Available at  
3886 <http://www.haskell.org/>

3887 [Hier] Anderson A, ed., *The XACML Profile for Hierarchical Resources*, OASIS  
3888 Access Control TC, Committee Draft 01, 16 Sep 2004, <http://www.oasis-open.org/committees/xacml>

3890 [Hinton94] Hinton, H, M, Lee,, E, S, The Compatibility of Policies, Proceedings 2nd  
3891 ACM Conference on Computer and Communications Security, Nov 1994,  
3892 Fairfax, Virginia, USA.

3893	<b>[IEEE754]</b>	IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-7653-8, IEEE Product No. SH10116-TBR
3894		
3895	<b>[ISO10181-3]</b>	ISO/IEC 10181-3:1996 Information technology – Open Systems Interconnection -- Security frameworks for open systems: Access control framework.
3896		
3897		
3898	<b>[Kudo00]</b>	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.
3899		
3900		
3901	<b>[LDAP-1]</b>	RFC2256, A summary of the X500(96) User Schema for use with LDAPv3, Section 5, M Wahl, December 1997 <a href="http://www.ietf.org/rfc/rfc2798.txt">http://www.ietf.org/rfc/rfc2798.txt</a>
3902		
3903	<b>[LDAP-2]</b>	RFC2798, Definition of the inetOrgPerson, M. Smith, April 2000
3904		<a href="http://www.ietf.org/rfc/rfc2798.txt">http://www.ietf.org/rfc/rfc2798.txt</a>
3905	<b>[MathML]</b>	Mathematical Markup Language (MathML), Version 2.0, W3C Recommendation, 21 February 2001. Available at:
3906		<a href="http://www.w3.org/TR/MathML2/">http://www.w3.org/TR/MathML2/</a>
3907		
3908	<b>[Multi]</b>	Anderson A, ed., <i>The XACML Profile for Requests for Multiple Resources</i> , OASIS Access Control TC, Working Draft 02, 4 June 2004,
3909		<a href="http://www.oasis-open.org/committees/xacml">http://www.oasis-open.org/committees/xacml</a>
3910		
3911	<b>[Perritt93]</b>	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:
3912		<a href="http://www.ifla.org/documents/infopol/copyright/perh2.txt">http://www.ifla.org/documents/infopol/copyright/perh2.txt</a>
3913		
3914		
3915	<b>[RBAC]</b>	Role-Based Access Controls, David Ferraiolo and Richard Kuhn, 15th National Computer Security Conference, 1992. Available at:
3916		<a href="http://csrc.nist.gov/rbac">http://csrc.nist.gov/rbac</a>
3917		
3918	<b>[RegEx]</b>	XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, Appendix D. Available at: <a href="http://www.w3.org/TR/xmlschema-0/">http://www.w3.org/TR/xmlschema-0/</a>
3919		
3920	<b>[RFC2119]</b>	S. Bradner, <i>Key words for use in RFCs to Indicate Requirement Levels</i> , <a href="http://www.ietf.org/rfc/rfc2119.txt">http://www.ietf.org/rfc/rfc2119.txt</a> , IETF RFC 2119, March 1997
3921		
3922	<b>[RFC2396]</b>	Berners-Lee T, Fielding R, Masinter L, Uniform Resource Identifiers (URI): Generic Syntax. Available at: <a href="http://www.ietf.org/rfc/rfc2396.txt">http://www.ietf.org/rfc/rfc2396.txt</a>
3923		
3924	<b>[RFC2732]</b>	Hinden R, Carpenter B, Masinter L, Format for Literal IPv6 Addresses in URL's. Available at: <a href="http://www.ietf.org/rfc/rfc2732.txt">http://www.ietf.org/rfc/rfc2732.txt</a>
3925		
3926	<b>[RFC3198]</b>	IETF RFC 3198: Terminology for Policy-Based Management, November 2001. <a href="http://www.ietf.org/rfc/rfc3198.txt">http://www.ietf.org/rfc/rfc3198.txt</a>
3927		
3928	<b>[SAML]</b>	Security Assertion Markup Language available from <a href="http://www.oasis-open.org/committees/security/#documents">http://www.oasis-open.org/committees/security/#documents</a>
3929		
3930	<b>[Sloman94]</b>	Sloman, M. Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Volume 2, part 4. Plenum Press. 1994.
3931		
3932		
3933	<b>[XACMLv1.0]</b>	Extensible access control markup language (XACML) Version 1.0. OASIS Standard. 18 February 2003. Available at: <a href="http://www.oasis-open.org/apps/org/workgroup/xacml/download.php/940/oasis-xacml-1.0.pdf">http://www.oasis-open.org/apps/org/workgroup/xacml/download.php/940/oasis-xacml-1.0.pdf</a>
3934		
3935		
3936		
3937	<b>[XACMLv1.1]</b>	Extensible access control markup language (XACML) Version 1.1. OASIS Committee Specification. 7 August 2003. Available at: <a href="http://www.oasis-open.org/apps/org/workgroup/xacml/download.php/4104/cs-xacml-specification-1.1.pdf">http://www.oasis-open.org/apps/org/workgroup/xacml/download.php/4104/cs-xacml-specification-1.1.pdf</a>
3938		
3939		
3940		

3941	<b>[XF]</b>	XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft
3942		16 August 2002. Available at: <a href="http://www.w3.org/TR/2002/WD-xquery-operators-20020816">http://www.w3.org/TR/2002/WD-xquery-operators-20020816</a>
3943		
3944	<b>[XS]</b>	XML Schema, parts 1 and 2. Available at:
3945		<a href="http://www.w3.org/TR/xmlschema-1/">http://www.w3.org/TR/xmlschema-1/</a> and
3946		<a href="http://www.w3.org/TR/xmlschema-2/">http://www.w3.org/TR/xmlschema-2/</a>
3947	<b>[XPath]</b>	XML Path Language (XPath), Version 1.0, W3C Recommendation 16
3948		November 1999. Available at: <a href="http://www.w3.org/TR/xpath">http://www.w3.org/TR/xpath</a>
3949	<b>[XSLT]</b>	XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16
3950		November 1999. Available at: <a href="http://www.w3.org/TR/xslt">http://www.w3.org/TR/xslt</a>
3951		



---

## 3952 Appendix A. Data-types and functions (normative)

### 3953 A.1. Introduction

3954 This section specifies the data-types and functions used in XACML to create *predicates* for  
3955 *conditions* and *target* matches.

3956 This specification combines the various standards set forth by IEEE and ANSI for string  
3957 representation of numeric values, as well as the evaluation of arithmetic functions. It describes the  
3958 primitive data-types and *bags*. The standard functions are named and their operational semantics  
3959 are described.

### 3960 A.2. Data-types

3961 Although XML instances represent all data-types as strings, an XACML *PDP* must reason about  
3962 types of data that, while they have string representations, are not just strings. Types such as  
3963 Boolean, integer and double **MUST** be converted from their XML string representations to values  
3964 that can be compared with values in their domain of discourse, such as numbers. The following  
3965 primitive data-types are specified for use with XACML and have explicit data representations:

- 3966 • <http://www.w3.org/2001/XMLSchema#string>
- 3967 • <http://www.w3.org/2001/XMLSchema#boolean>
- 3968 • <http://www.w3.org/2001/XMLSchema#integer>
- 3969 • <http://www.w3.org/2001/XMLSchema#double>
- 3970 • <http://www.w3.org/2001/XMLSchema#time>
- 3971 • <http://www.w3.org/2001/XMLSchema#date>
- 3972 • <http://www.w3.org/2001/XMLSchema#dateTime>
- 3973 • <http://www.w3.org/2001/XMLSchema#anyURI>
- 3974 • <http://www.w3.org/2001/XMLSchema#hexBinary>
- 3975 • <http://www.w3.org/2001/XMLSchema#base64Binary>
- 3976 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration>
- 3977 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration>
- 3978 • <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
- 3979 • <urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name>
- 3980 • <urn:oasis:names:tc:xacml:1.0:data-type:ipAddress>
- 3981 • <urn:oasis:names:tc:xacml:1.0:data-type:dnsName>

3982 For the sake of improved interoperability, it is RECOMMENDED that all time references be in UTC  
3983 time.

3984 An XACML **PDP** SHALL be capable of converting string representations into various primitive data-  
3985 types. For integers and doubles, XACML SHALL use the conversions described in [IEEE754].

3986 XACML defines three data-types; these are:

3987 "urn:oasis:names:tc:xacml:1.0:data-type:x500Name",

3988 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"

3989 "urn:oasis:names:tc:xacml:1.0:data-type:ipAddress"

3990 "urn:oasis:names:tc:xacml:1.0:data-type:dnsName" and

3991 These types represent identifiers for subjects or resources and appear in several standard  
3992 applications, such as TLS/SSL and electronic mail.

### 3993 **X.500 directory name**

3994 The "urn:oasis:names:tc:xacml:1.0:data-type:x500Name" primitive type represents an ITU-T Rec.  
3995 X.520 Distinguished Name. The valid syntax for such a name is described in IETF RFC 2253  
3996 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names"

### 3997 **RFC 822 name**

3998 The "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" primitive type represents an electronic  
3999 mail address. The valid syntax for such a name is described in IETF RFC 2821, Section 4.1.2,  
4000 Command Argument Syntax, under the term "Mailbox".

### 4001 **IP address**

4002 The "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" primitive type represents an IPv4 or IPv6  
4003 network address, with optional mask and optional port or port range. The syntax SHALL be:

4004  
4005 ipAddress = address [ "/" mask ] [ ":" [ portrange ] ]

4006  
4007 For an IPv4 address, the address and mask are formatted in accordance with the syntax for a  
4008 "host" in IETF RFC 2396 "Uniform Resource Identifiers (URI): Generic Syntax", section 3.2.  
4009 For an IPv6 address, the address and mask are formatted in accordance with the syntax for an  
4010 "ipv6reference" in IETF RFC 2732 "Format for Literal IPv6 Addresses in URL's". (Note that an IPv6  
4011 address or mask, in this syntax, is enclosed in literal "[" "]" brackets.)

4012

### 4013 **DNS name**

4014 The "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" primitive type represents a Domain Name  
4015 Service (DNS) host name, with optional port or port range. The syntax SHALL be:

4016  
4017 dnsName = hostname [ ":" portrange ]

4018

4019 The hostname is formatted in accordance with IETF RFC 2396 "Uniform Resource Identifiers (URI):  
4020 Generic Syntax", section 3.2, except that a wildcard "\*" may be used in the left-most component of  
4021 the hostname to indicate "any subdomain" under the domain specified to its right.

4022

4023 For both the "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" and  
4024 "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" data-types, the port or port range syntax  
4025 SHALL be

4026

4027 portrange = portnumber | "-"portnumber | portnumber-"[portnumber]

4028

4029 where "portnumber" is a decimal port number. If the port number is of the form "-x", where "x" is a  
4030 port number, then the range is all ports numbered "x" and below. If the port number is of the form

4031 "x-", then the range is all ports numbered "x" and above. [This syntax is taken from the Java  
4032 SocketPermission.]

## 4033 **A.3. Functions**

4034 XACML specifies the following functions. If an argument of one of these functions were to evaluate  
4035 to "Indeterminate", then the function SHALL be set to "Indeterminate".

### 4036 **A.3.1 Equality predicates**

4037 The following functions are the *equality* functions for the various primitive types. Each function for a  
4038 particular data-type follows a specified standard convention for that data-type.

- 4039 • urn:oasis:names:tc:xacml:1.0:function:string-equal

4040 This function SHALL take two arguments of data-type  
4041 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4042 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and  
4043 only if the value of both of its arguments are of equal length and each string is determined  
4044 to be equal byte-by-byte according to the function "integer-equal". Otherwise, it SHALL  
4045 return "False".

- 4046 • urn:oasis:names:tc:xacml:1.0:function:boolean-equal

4047 This function SHALL take two arguments of data-type  
4048 "http://www.w3.org/2001/XMLSchema#boolean" and SHALL return an  
4049 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and  
4050 only if the arguments are equal. Otherwise, it SHALL return "False".

- 4051 • urn:oasis:names:tc:xacml:1.0:function:integer-equal

4052 This function SHALL take two arguments of data-type  
4053 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return an  
4054 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on  
4055 integers according to IEEE 754 [IEEE 754].

- 4056 • urn:oasis:names:tc:xacml:1.0:function:double-equal

4057 This function SHALL take two arguments of data-type  
4058 "http://www.w3.org/2001/XMLSchema#double" and SHALL return an  
4059 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on  
4060 doubles according to IEEE 754 [IEEE 754].

- 4061 • urn:oasis:names:tc:xacml:1.0:function:date-equal

4062 This function SHALL take two arguments of data-type  
4063 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
4064 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation  
4065 according to the "op:date-equal" function [XF Section 8.3.11].

- 4066 • urn:oasis:names:tc:xacml:1.0:function:time-equal

4067 This function SHALL take two arguments of data-type  
4068 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
4069 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation  
4070 according to the "op:time-equal" function [XF Section 8.3.14].

- 4071 • urn:oasis:names:tc:xacml:1.0:function:dateTime-equal
- 4072 This function SHALL take two arguments of data-type  
 4073 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
 4074 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation  
 4075 according to the "op:dateTime-equal" function [XF Section 8.3.8].
- 4076 • urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-equal
- 4077 This function SHALL take two arguments of data-type "http://www.w3.org/TR/2002/WD-  
 4078 xquery-operators-20020816#dayTimeDuration" and SHALL return an  
 4079 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation  
 4080 according to the "op:dayTimeDuration-equal" function [XF Section 8.3.5]. Note that the  
 4081 lexical representation of each argument MUST be converted to a value expressed in  
 4082 fractional seconds [XF Section 8.2.2].
- 4083 • urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-equal
- 4084 This function SHALL take two arguments of data-type "http://www.w3.org/TR/2002/WD-  
 4085 xquery-operators-20020816#yearMonthDuration" and SHALL return an  
 4086 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation  
 4087 according to the "op:yearMonthDuration-equal" function [XF Section 8.3.2]. Note that the  
 4088 lexical representation of each argument MUST be converted to a value expressed in  
 4089 integer months [XF Section 8.2.1].
- 4090 • urn:oasis:names:tc:xacml:1.0:function:anyURI-equal
- 4091 This function SHALL take two arguments of data-type  
 4092 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return an  
 4093 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation  
 4094 according to the "op:anyURI-equal" function [XF Section 10.2.1].
- 4095 • urn:oasis:names:tc:xacml:1.0:function:x500Name-equal
- 4096 This function SHALL take two arguments of "urn:oasis:names:tc:xacml:1.0:data-  
 4097 type:x500Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It  
 4098 SHALL return "True" if and only if each Relative Distinguished Name (RDN) in the two  
 4099 arguments matches. Otherwise, it SHALL return "False". Two RDNs shall be said to  
 4100 match if and only if the result of the following operations is "True"<sup>3</sup>.
- 4101 1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory  
 4102 Access Protocol (v3): UTF-8 String Representation of Distinguished Names".
  - 4103 2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute  
 4104 ValuePairs in that RDN in ascending order when compared as octet strings  
 4105 (described in ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").
  - 4106 3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key  
 4107 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section  
 4108 4.1.2.4 "Issuer".
- 4109 • urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal
- 4110 This function SHALL take two arguments of data-type "urn:oasis:names:tc:xacml:1.0:data-  
 4111 type:rfc822Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean".  
 4112 It SHALL return "True" if and only if the two arguments are equal. Otherwise, it SHALL

---

<sup>3</sup> 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.

- 4113 return "False". An RFC822 name consists of a *local-part* followed by "@" followed by a  
 4114 *domain-part*. The *local-part* is case-sensitive, while the *domain-part* (which is usually a  
 4115 DNS host name) is not case-sensitive. Perform the following operations:
- 4116 1. Normalize the *domain-part* of each argument to lower case
  - 4117 2. Compare the expressions by applying the function  
 4118 "urn:oasis:names:tc:xacml:1.0:function:string-equal" to the normalized arguments.
- 4119 • urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal  
 4120 This function SHALL take two arguments of data-type  
 4121 "http://www.w3.org/2001/XMLSchema#hexBinary" and SHALL return an  
 4122 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the octet  
 4123 sequences represented by the value of both arguments have equal length and are equal in  
 4124 a conjunctive, point-wise, comparison using the  
 4125 "urn:oasis:names:tc:xacml:1.0:function:integer-equal" function. Otherwise, it SHALL return  
 4126 "False". The conversion from the string representation to an octet sequence SHALL be as  
 4127 specified in [XS Section 8.2.15].
  - 4128 • urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal  
 4129 This function SHALL take two arguments of data-type  
 4130 "http://www.w3.org/2001/XMLSchema#base64Binary" and SHALL return an  
 4131 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the octet  
 4132 sequences represented by the value of both arguments have equal length and are equal in  
 4133 a conjunctive, point-wise, comparison using the  
 4134 "urn:oasis:names:tc:xacml:1.0:function:integer-equal" function. Otherwise, it SHALL return  
 4135 "False". The conversion from the string representation to an octet sequence SHALL be as  
 4136 specified in [XS Section 8.2.16].

### 4137 **A.3.2 Arithmetic functions**

4138 All of the following functions SHALL take two arguments of the specified *data-type*, integer or  
 4139 double, and SHALL return an element of integer or double data-type, respectively. However, the  
 4140 "add" functions MAY take more than two arguments. Each function evaluation SHALL proceed as  
 4141 specified by their logical counterparts in IEEE 754 [IEEE 754]. In an expression that contains any  
 4142 of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to  
 4143 "Indeterminate". In the case of the divide functions, if the divisor is zero, then the function SHALL  
 4144 evaluate to "Indeterminate".

- 4145 • urn:oasis:names:tc:xacml:1.0:function:integer-add  
 4146 This function MAY have two or more arguments.
- 4147 • urn:oasis:names:tc:xacml:1.0:function:double-add  
 4148 This function MAY have two or more arguments.
- 4149 • urn:oasis:names:tc:xacml:1.0:function:integer-subtract
- 4150 • urn:oasis:names:tc:xacml:1.0:function:double-subtract
- 4151 • urn:oasis:names:tc:xacml:1.0:function:integer-multiply
- 4152 • urn:oasis:names:tc:xacml:1.0:function:double-multiply
- 4153 • urn:oasis:names:tc:xacml:1.0:function:integer-divide
- 4154 • urn:oasis:names:tc:xacml:1.0:function:double-divide

- 4155
- urn:oasis:names:tc:xacml:1.0:function:integer-mod

4156 The following functions SHALL take a single argument of the specified *data-type*. The round and  
4157 floor functions SHALL take a single argument of data-type  
4158 “http://www.w3.org/2001/XMLSchema#double” and return a value of the data-type  
4159 “http://www.w3.org/2001/XMLSchema#double”.

- 4160
- urn:oasis:names:tc:xacml:1.0:function:integer-abs
- 4161
- urn:oasis:names:tc:xacml:1.0:function:double-abs
- 4162
- urn:oasis:names:tc:xacml:1.0:function:round
- 4163
- urn:oasis:names:tc:xacml:1.0:function:floor

### 4164 **A.3.3 String conversion functions**

4165 The following functions convert between values of the data-type  
4166 “http://www.w3.org/2001/XMLSchema#string” primitive types.

- 4167
- urn:oasis:names:tc:xacml:1.0:function:string-normalize-space

4168 This function SHALL take one argument of data-type  
4169 “http://www.w3.org/2001/XMLSchema#string” and SHALL normalize the value by stripping  
4170 off all leading and trailing white space characters.

- 4171
- urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case

4172 This function SHALL take one argument of data-type  
4173 “http://www.w3.org/2001/XMLSchema#string” and SHALL normalize the value by  
4174 converting each upper case character to its lower case equivalent.

### 4175 **A.3.4 Numeric data-type conversion functions**

4176 The following functions convert between the data-type  
4177 “http://www.w3.org/2001/XMLSchema#integer” and “http://www.w3.org/2001/XMLSchema#double”  
4178 primitive types.

- 4179
- urn:oasis:names:tc:xacml:1.0:function:double-to-integer

4180 This function SHALL take one argument of data-type  
4181 “http://www.w3.org/2001/XMLSchema#double” and SHALL truncate its numeric value to a  
4182 whole number and return an element of data-type  
4183 “http://www.w3.org/2001/XMLSchema#integer”.

- 4184
- urn:oasis:names:tc:xacml:1.0:function:integer-to-double

4185 This function SHALL take one argument of data-type  
4186 “http://www.w3.org/2001/XMLSchema#integer” and SHALL promote its value to an element  
4187 of data-type “http://www.w3.org/2001/XMLSchema#double” with the same numeric value.

### 4188 **A.3.5 Logical functions**

4189 This section contains the specification for logical functions that operate on arguments of data-type  
4190 “http://www.w3.org/2001/XMLSchema#boolean”.

- 4191
- urn:oasis:names:tc:xacml:1.0:function:or



4192 This function SHALL return "False" if it has no arguments and SHALL return "True" if at  
4193 least one of its arguments evaluates to "True". The order of evaluation SHALL be from first  
4194 argument to last. The evaluation SHALL stop with a result of "True" if any argument  
4195 evaluates to "True", leaving the rest of the arguments unevaluated.

- 4196 • urn:oasis:names:tc:xacml:1.0:function:and

4197 This function SHALL return "True" if it has no arguments and SHALL return "False" if one of  
4198 its arguments evaluates to "False". The order of evaluation SHALL be from first argument  
4199 to last. The evaluation SHALL stop with a result of "False" if any argument evaluates to  
4200 "False", leaving the rest of the arguments unevaluated.

- 4201 • urn:oasis:names:tc:xacml:1.0:function:n-of

4202 The first argument to this function SHALL be of data-type  
4203 <http://www.w3.org/2001/XMLSchema#integer>. The remaining arguments SHALL be of  
4204 data-type <http://www.w3.org/2001/XMLSchema#boolean>. The first argument specifies the  
4205 minimum number of the remaining arguments that MUST evaluate to "True" for the  
4206 expression to be considered "True". If the first argument is 0, the result SHALL be "True".  
4207 If the number of arguments after the first one is less than the value of the first argument,  
4208 then the expression SHALL result in "Indeterminate". The order of evaluation SHALL be:  
4209 first evaluate the integer value, then evaluate each subsequent argument. The evaluation  
4210 SHALL stop and return "True" if the specified number of arguments evaluate to "True". The  
4211 evaluation of arguments SHALL stop if it is determined that evaluating the remaining  
4212 arguments will not satisfy the requirement.

- 4213 • urn:oasis:names:tc:xacml:1.0:function:not

4214 This function SHALL take one argument of data-type  
4215 "<http://www.w3.org/2001/XMLSchema#boolean>". If the argument evaluates to "True", then  
4216 the result of the expression SHALL be "False". If the argument evaluates to "False", then  
4217 the result of the expression SHALL be "True".

4218 Note: When evaluating and, or, or n-of, it MAY NOT be necessary to attempt a full evaluation of  
4219 each argument in order to determine whether the evaluation of the argument would result in  
4220 "Indeterminate". Analysis of the argument regarding the availability of its attributes, or other  
4221 analysis regarding errors, such as "divide-by-zero", may render the argument error free. Such  
4222 arguments occurring in the expression in a position after the evaluation is stated to stop need not  
4223 be processed.

### 4224 **A.3.6 Numeric comparison functions**

4225 These functions form a minimal set for comparing two numbers, yielding a Boolean result. They  
4226 SHALL comply with the rules governed by IEEE 754 [IEEE 754].

- 4227 • urn:oasis:names:tc:xacml:1.0:function:integer-greater-than
- 4228 • urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal
- 4229 • urn:oasis:names:tc:xacml:1.0:function:integer-less-than
- 4230 • urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal
- 4231 • urn:oasis:names:tc:xacml:1.0:function:double-greater-than
- 4232 • urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal
- 4233 • urn:oasis:names:tc:xacml:1.0:function:double-less-than

- 4234 • urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal

### 4235 **A.3.7 Date and time arithmetic functions**

4236 These functions perform arithmetic operations with date and time.

- 4237 • urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration

4238 This function SHALL take two arguments, the first SHALL be of data-type  
4239 “http://www.w3.org/2001/XMLSchema#dateTime” and the second SHALL be of data-type  
4240 “http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration”. It SHALL  
4241 return a result of “http://www.w3.org/2001/XMLSchema#dateTime”. This function SHALL  
4242 return the value by adding the second argument to the first argument according to the  
4243 specification of adding durations to date and time [XS Appendix E].

- 4244 • urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration

4245 This function SHALL take two arguments, the first SHALL be a  
4246 “http://www.w3.org/2001/XMLSchema#dateTime” and the second SHALL be a  
4247 “http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration”. It  
4248 SHALL return a result of “http://www.w3.org/2001/XMLSchema#dateTime”. This function  
4249 SHALL return the value by adding the second argument to the first argument according to  
4250 the specification of adding durations to date and time [XS Appendix E].

- 4251 • urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-dayTimeDuration

4252 This function SHALL take two arguments, the first SHALL be a  
4253 “http://www.w3.org/2001/XMLSchema#dateTime” and the second SHALL be a  
4254 “http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration”. It SHALL  
4255 return a result of “http://www.w3.org/2001/XMLSchema#dateTime”. If the second argument  
4256 is a positive duration, then this function SHALL return the value by adding the  
4257 corresponding negative duration, as per the specification [XS Appendix E]. If the second  
4258 argument is a negative duration, then the result SHALL be as if the function  
4259 “urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration” had been applied  
4260 to the corresponding positive duration.

- 4261 • urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-yearMonthDuration

4262 This function SHALL take two arguments, the first SHALL be a  
4263 “http://www.w3.org/2001/XMLSchema#dateTime” and the second SHALL be a  
4264 “http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration”. It  
4265 SHALL return a result of “http://www.w3.org/2001/XMLSchema#dateTime”. If the second  
4266 argument is a positive duration, then this function SHALL return the value by adding the  
4267 corresponding negative duration, as per the specification [XS Appendix E]. If the second  
4268 argument is a negative duration, then the result SHALL be as if the function  
4269 “urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration” had been  
4270 applied to the corresponding positive duration.

- 4271 • urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration

4272 This function SHALL take two arguments, the first SHALL be a  
4273 “http://www.w3.org/2001/XMLSchema#date” and the second SHALL be a  
4274 “http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration”. It  
4275 SHALL return a result of “http://www.w3.org/2001/XMLSchema#date”. This function  
4276 SHALL return the value by adding the second argument to the first argument according to  
4277 the specification of adding duration to date [XS Appendix E].

- 4278 • urn:oasis:names:tc:xacml:1.0:function:date-subtract-yearMonthDuration

4279 This function SHALL take two arguments, the first SHALL be a  
4280 "http://www.w3.org/2001/XMLSchema#date" and the second SHALL be a  
4281 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It  
4282 SHALL return a result of "http://www.w3.org/2001/XMLSchema#date". If the second  
4283 argument is a positive duration, then this function SHALL return the value by adding the  
4284 corresponding negative duration, as per the specification [XS Appendix E]. If the second  
4285 argument is a negative duration, then the result SHALL be as if the function  
4286 "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" had been applied to  
4287 the corresponding positive duration.

### 4288 **A.3.8 Non-numeric comparison functions**

4289 These functions perform comparison operations on two arguments of non-numerical types.

- 4290 • urn:oasis:names:tc:xacml:1.0:function:string-greater-than

4291 This function SHALL take two arguments of data-type  
4292 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4293 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
4294 arguments are compared byte by byte and, after an initial prefix of corresponding bytes  
4295 from both arguments that are considered equal by  
4296 "urn:oasis:names:tc:xacml:1.0:function:integer-equal", the next byte by byte comparison is  
4297 such that the byte from the first argument is greater than the byte from the second  
4298 argument by the use of the function "urn:oasis:names:tc:xacml:2.0:function:integer-greater-  
4299 then". Otherwise, it SHALL return "False".

- 4300 • urn:oasis:names:tc:xacml:1.0:function:string-greater-than-or-equal

4301 This function SHALL take two arguments of data-type  
4302 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4303 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return a result as if evaluated  
4304 with the logical function "urn:oasis:names:tc:xacml:1.0:function:or" with two arguments  
4305 containing the functions "urn:oasis:names:tc:xacml:1.0:function:string-greater-than" and  
4306 "urn:oasis:names:tc:xacml:1.0:function:string-equal" containing the original arguments

- 4307 • urn:oasis:names:tc:xacml:1.0:function:string-less-than

4308 This function SHALL take two arguments of data-type  
4309 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4310 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
4311 arguments are compared byte by byte and, after an initial prefix of corresponding bytes  
4312 from both arguments that are considered equal by  
4313 "urn:oasis:names:tc:xacml:1.0:function:integer-equal", the next byte by byte comparison is  
4314 such that the byte from the first argument is less than the byte from the second argument  
4315 by the use of the function "urn:oasis:names:tc:xacml:1.0:function:integer-less-than".  
4316 Otherwise, it SHALL return "False".

- 4317 • urn:oasis:names:tc:xacml:1.0:function:string-less-than-or-equal

4318 This function SHALL take two arguments of data-type  
4319 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4320 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return a result as if evaluated  
4321 with the function "urn:oasis:names:tc:xacml:1.0:function:or" with two arguments containing  
4322 the functions "urn:oasis:names:tc:xacml:1.0:function:string-less-than" and  
4323 "urn:oasis:names:tc:xacml:1.0:function:string-equal" containing the original arguments.

- 4324 • urn:oasis:names:tc:xacml:1.0:function:time-greater-than

4325 This function SHALL take two arguments of data-type  
4326 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
4327 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
4328 first argument is greater than the second argument according to the order relation specified  
4329 for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8]. Otherwise, it SHALL  
4330 return "False". Note: it is illegal to compare a time that includes a time-zone value with one  
4331 that does not. In such cases, the time-in-range function should be used.

- 4332 • urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal

4333 This function SHALL take two arguments of data-type  
4334 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
4335 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
4336 first argument is greater than or equal to the second argument according to the order  
4337 relation specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].  
4338 Otherwise, it SHALL return "False". Note: it is illegal to compare a time that includes a  
4339 time-zone value with one that does not. In such cases, the time-in-range function should  
4340 be used.

- 4341 • urn:oasis:names:tc:xacml:1.0:function:time-less-than

4342 This function SHALL take two arguments of data-type  
4343 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
4344 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
4345 first argument is less than the second argument according to the order relation specified for  
4346 "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8]. Otherwise, it SHALL  
4347 return "False". Note: it is illegal to compare a time that includes a time-zone value with one  
4348 that does not. In such cases, the time-in-range function should be used.

- 4349 • urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal

4350 This function SHALL take two arguments of data-type  
4351 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
4352 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
4353 first argument is less than or equal to the second argument according to the order relation  
4354 specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8]. Otherwise, it  
4355 SHALL return "False". Note: it is illegal to compare a time that includes a time-zone value  
4356 with one that does not. In such cases, the time-in-range function should be used.

- 4357 • urn:oasis:names:tc:xacml:1.0:function:time-in-range

4358 This function SHALL take three arguments of data-type  
4359 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
4360 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
4361 argument falls in the range defined inclusively by the second and third arguments.  
4362 Otherwise, it SHALL return "False". Regardless of its value, the third argument SHALL be  
4363 interpreted as a time that is equal to, or later than by less than twenty-four hours, the  
4364 second argument. If no time zone is provided for the first argument, it SHALL use the  
4365 default time zone at the context handler. If no time zone is provided for the second or third  
4366 arguments, then they SHALL use the time zone from the first argument.

- 4367 • urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than

4368 This function SHALL take two arguments of data-type  
4369 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
4370 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
4371 first argument is greater than the second argument according to the order relation specified  
4372 for "http://www.w3.org/2001/XMLSchema#dateTime" by [XF Section 3.2.7]. Otherwise, it

- 4373 SHALL return "False". Note: if a dateTime value does not include a time-zone value, then  
4374 an implicit time-zone value SHALL be assigned, as described in [XF].
- 4375 • urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal
- 4376 This function SHALL take two arguments of data-type  
4377 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
4378 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
4379 first argument is greater than or equal to the second argument according to the order  
4380 relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" by [XF Section  
4381 3.2.7]. Otherwise, it SHALL return "False". Note: if a dateTime value does not include a  
4382 time-zone value, then an implicit time-zone value SHALL be assigned, as described in  
4383 [XF].
- 4384 • urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than
- 4385 This function SHALL take two arguments of data-type  
4386 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
4387 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
4388 first argument is less than the second argument according to the order relation specified for  
4389 "http://www.w3.org/2001/XMLSchema#dateTime" by [XF Section 3.2.7]. Otherwise, it  
4390 SHALL return "False". Note: if a dateTime value does not include a time-zone value, then  
4391 an implicit time-zone value SHALL be assigned, as described in [XF].
- 4392 • urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal
- 4393 This function SHALL take two arguments of data-type  
4394 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
4395 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
4396 first argument is less than or equal to the second argument according to the order relation  
4397 specified for "http://www.w3.org/2001/XMLSchema#dateTime" by [XF Section 3.2.7].  
4398 Otherwise, it SHALL return "False". Note: if a dateTime value does not include a time-zone  
4399 value, then an implicit time-zone value SHALL be assigned, as described in [XF].
- 4400 • urn:oasis:names:tc:xacml:1.0:function:date-greater-than
- 4401 This function SHALL take two arguments of data-type  
4402 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
4403 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
4404 first argument is greater than the second argument according to the order relation specified  
4405 for "http://www.w3.org/2001/XMLSchema#date" by [XF Section 3.2.9]. Otherwise, it SHALL  
4406 return "False". Note: if a date value does not include a time-zone value, then an implicit  
4407 time-zone value SHALL be assigned, as described in [XF].
- 4408 • urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal
- 4409 This function SHALL take two arguments of data-type  
4410 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
4411 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
4412 first argument is greater than or equal to the second argument according to the order  
4413 relation specified for "http://www.w3.org/2001/XMLSchema#date" by [XF Section 3.2.9].  
4414 Otherwise, it SHALL return "False". Note: if a date value does not include a time-zone  
4415 value, then an implicit time-zone value SHALL be assigned, as described in [XF].
- 4416 • urn:oasis:names:tc:xacml:1.0:function:date-less-than
- 4417 This function SHALL take two arguments of data-type  
4418 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
4419 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the



4420 first argument is less than the second argument according to the order relation specified for  
4421 "http://www.w3.org/2001/XMLSchema#date" by [XF Section 3.2.9]. Otherwise, it SHALL  
4422 return "False". Note: if a date value does not include a time-zone value, then an implicit  
4423 time-zone value SHALL be assigned, as described in [XF].

- 4424 • urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal

4425 This function SHALL take two arguments of data-type  
4426 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
4427 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
4428 first argument is less than or equal to the second argument according to the order relation  
4429 specified for "http://www.w3.org/2001/XMLSchema#date" by [XF Section 3.2.9]. Otherwise,  
4430 it SHALL return "False". Note: if a date value does not include a time-zone value, then an  
4431 implicit time-zone value SHALL be assigned, as described in [XF].

### 4432 A.3.9 String functions

4433 The following functions operate on strings and URIs.

- 4434 • urn:oasis:names:tc:xacml:2.0:function:string-concatenate

4435

4436 This function SHALL take two or more arguments of data-type  
4437 "http://www.w3.org/2001/XMLSchema#string" and SHALL return a  
4438 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the concatenation, in  
4439 order, of the arguments.

- 4440 • urn:oasis:names:tc:xacml:2.0:function:url-string-concatenate

4441 This function SHALL take one argument of data-type  
4442 "http://www.w3.org/2001/XMLSchema#anyURI" and one or more arguments of type  
4443 "http://www.w3.org/2001/XMLSchema#string", and SHALL return a  
4444 "http://www.w3.org/2001/XMLSchema#anyURI". The result SHALL be the URI constructed  
4445 by appending, in order, the "string" arguments to the "anyURI" argument.

### 4446 A.3.10 Bag functions

4447 These functions operate on a **bag** of 'type' values, where *type* is one of the primitive data-types.  
4448 Some additional conditions defined for each function below SHALL cause the expression to  
4449 evaluate to "Indeterminate".

- 4450 • urn:oasis:names:tc:xacml:1.0:function:type-one-and-only

4451 This function SHALL take a **bag** of 'type' values as an argument and SHALL return a value  
4452 of '-type'. It SHALL return the only value in the **bag**. If the **bag** does not have one and only  
4453 one value, then the expression SHALL evaluate to "Indeterminate".

- 4454 • urn:oasis:names:tc:xacml:1.0:function:type-bag-size

4455 This function SHALL take a **bag** of 'type' values as an argument and SHALL return an  
4456 "http://www.w3.org/2001/XMLSchema#integer" indicating the number of values in the **bag**.

- 4457 • urn:oasis:names:tc:xacml:1.0:function:type-is-in

4458 This function SHALL take an argument of 'type' as the first argument and a **bag** of *type*  
4459 values as the second argument and SHALL return an  
4460 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL evaluate to "True" if



4461 and only if the first argument matches by the "urn:oasis:names:tc:xacml:x.x:function:type-  
4462 equal" any value in the **bag**. Otherwise, it SHALL return "False".

- 4463 • urn:oasis:names:tc:xacml:1.0:function:type-bag

4464 This function SHALL take any number of arguments of 'type' and return a **bag** of 'type'  
4465 values containing the values of the arguments. An application of this function to zero  
4466 arguments SHALL produce an empty **bag** of the specified data-type.

### 4467 **A.3.11 Set functions**

4468 These functions operate on **bags** mimicking sets by eliminating duplicate elements from a **bag**.

- 4469 • urn:oasis:names:tc:xacml:1.0:function:type-intersection

4470 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL  
4471 return a **bag** of 'type' values such that it contains only elements that are common between  
4472 the two **bags**, which is determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal".  
4473 No duplicates, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal",  
4474 SHALL exist in the result.

- 4475 • urn:oasis:names:tc:xacml:1.0:function:type-at-least-one-member-of

4476 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL  
4477 return a "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL evaluate to  
4478 "True" if and only if at least one element of the first argument is contained in the second  
4479 argument as determined by "urn:oasis:names:tc:xacml:x.x:function:type-is-in".

- 4480 • urn:oasis:names:tc:xacml:1.0:function:type-union

4481 This function SHALL take two arguments that are both a **bag** of 'type' values. The  
4482 expression SHALL return a **bag** of 'type' such that it contains all elements of both **bags**.  
4483 No duplicates, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal",  
4484 SHALL exist in the result.

- 4485 • urn:oasis:names:tc:xacml:1.0:function:type-subset

4486 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL  
4487 return a "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and  
4488 only if the first argument is a subset of the second argument. Each argument SHALL be  
4489 considered to have had its duplicates removed, as determined by  
4490 "urn:oasis:names:tc:xacml:x.x:function:type-equal", before the subset calculation.

- 4491 • urn:oasis:names:tc:xacml:1.0:function:type-set-equals

4492 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL  
4493 return a "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return the result of  
4494 applying "urn:oasis:names:tc:xacml:1.0:function:and" to the application of  
4495 "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the first and second arguments and  
4496 the application of "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the second and  
4497 first arguments.

### 4498 **A.3.12 Higher-order bag functions**

4499 This section describes functions in XACML that perform operations on **bags** such that functions  
4500 may be applied to the **bags** in general.

4501 In this section, a general-purpose functional language called Haskell [**Haskell**] is used to formally  
4502 specify the semantics of these functions. Although the English description is adequate, a formal  
4503 specification of the semantics is helpful.

4504 For a quick summary, in the following Haskell notation, a function definition takes the form of  
4505 clauses that are applied to patterns of structures, namely lists. The symbol “[]” denotes the empty  
4506 list, whereas the expression “(x:xs)” matches against an argument of a non-empty list of which “x”  
4507 represents the first element of the list, and “xs” is the rest of the list, which may be an empty list.  
4508 We use the Haskell notion of a list, which is an ordered collection of elements, to model the XACML  
4509 **bags** of values.

4510 A simple Haskell definition of a familiar function “urn:oasis:names:tc:xacml:1.0:function:and” that  
4511 takes a list of values of type Boolean is defined as follows:

4512       and:: [Bool] -> Bool

4513       and []       = "True"

4514       and (x:xs) = x && (and xs)

4515 The first definition line denoted by a “::” formally describes the data-type of the function, which takes  
4516 a list of Booleans, denoted by “[Bool]”, and returns a Boolean, denoted by “Bool”. The second  
4517 definition line is a clause that states that the function “and” applied to the empty list is “True”. The  
4518 third definition line is a clause that states that for a non-empty list, such that the first element is “x”,  
4519 which is a value of data-type Bool, the function “and” applied to x SHALL be combined with, using  
4520 the logical conjunction function, which is denoted by the infix symbol “&&”, the result of recursively  
4521 applying the function “and” to the rest of the list. Of course, an application of the “and” function is  
4522 “True” if and only if the list to which it is applied is empty or every element of the list is “True”. For  
4523 example, the evaluation of the following Haskell expressions,

4524       (and []), (and ["True"]), (and ["True","True"]), (and ["True","True","False"])

4525 evaluate to "True", "True", "True", and "False", respectively.

4526 • urn:oasis:names:tc:xacml:1.0:function:any-of

4527       This function applies a Boolean function between a specific primitive value and a **bag** of  
4528 values, and SHALL return "True" if and only if the predicate is "True" for at least one  
4529 element of the **bag**.

4530       This function SHALL take three arguments. The first argument SHALL be an  
4531 <xacml:Function> element that names a Boolean function that takes two arguments of  
4532 primitive types. The second argument SHALL be a value of a primitive data-type. The third  
4533 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated  
4534 as if the function named in the <xacml:Function> argument were applied to the second  
4535 argument and each element of the third argument (the **bag**) and the results are combined  
4536 with “urn:oasis:names:tc:xacml:1.0:function:or”.

4537       In Haskell, the semantics of this operation are as follows:

4538       any\_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool

4539       any\_of f a []       = "False"

4540       any\_of f a (x:xs) = (f a x) || (any\_of f a xs)

4541       In the above notation, “f” is the function to be applied, “a” is the primitive value, and “(x:xs)”  
4542 represents the first element of the list as “x” and the rest of the list as “xs”.

4543       For example, the following expression SHALL return "True":

```

4544 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
4545   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
4546   <AttributeValue
4547     DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4548   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4549     <AttributeValue
4550       DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4551     <AttributeValue
4552       DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4553     <AttributeValue
4554       DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4555     <AttributeValue
4556       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4557   </Apply>
4558 </Apply>

```

4559       This expression is "True" because the first argument is equal to at least one of the  
4560       elements of the **bag**, according to the function.

4561     • urn:oasis:names:tc:xacml:1.0:function:all-of

4562       This function applies a Boolean function between a specific primitive value and a **bag** of  
4563       values, and returns "True" if and only if the predicate is "True" for every element of the **bag**.

4564       This function SHALL take three arguments. The first argument SHALL be an  
4565       <xacml:Function> element that names a Boolean function that takes two arguments of  
4566       primitive types. The second argument SHALL be a value of a primitive data-type. The third  
4567       argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated  
4568       as if the function named in the <xacml:Function> argument were applied to the second  
4569       argument and each element of the third argument (the **bag**) and the results were combined  
4570       using "urn:oasis:names:tc:xacml:1.0:function:and".

4571       In Haskell, the semantics of this operation are as follows:

```

4572             all_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
4573             all_of f a [] = "False"
4574             all_of f a (x:xs) = (f a x) && (all_of f a xs)

```

4575       In the above notation, "f" is the function to be applied, "a" is the primitive value, and "(x:xs)"  
4576       represents the first element of the list as "x" and the rest of the list as "xs".

4577       For example, the following expression SHALL evaluate to "True":

```

4578 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">
4579   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater"/>
4580   <AttributeValue
4581     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4582   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4583     <AttributeValue
4584       DataType="http://www.w3.org/2001/XMLSchema#integer">9</AttributeValue>
4585     <AttributeValue
4586       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4587     <AttributeValue
4588       DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4589     <AttributeValue
4590       DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4591   </Apply>
4592 </Apply>

```

4593       This expression is "True" because the first argument (10) is greater than *all* of the elements  
4594       of the **bag** (9,3,4 and 2).

4595     • urn:oasis:names:tc:xacml:1.0:function:any-of-any

4596 This function applies a Boolean function between each element of a **bag** of values and  
4597 each element of another **bag** of values, and returns "True" if and only if the predicate is  
4598 "True" for at least one comparison.

4599 This function SHALL take three arguments. The first argument SHALL be an  
4600 `<xacml:Function>` element that names a Boolean function that takes two arguments of  
4601 primitive types. The second argument SHALL be a **bag** of a primitive data-type. The third  
4602 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated  
4603 as if the function named in the `<xacml:Function>` argument were applied between  
4604 every element of the second argument and every element of the third argument and the  
4605 results were combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics  
4606 are that the result of the expression SHALL be "True" if and only if the applied predicate is  
4607 "True" for *at least one* comparison of elements from the two **bags**.

4608 In Haskell, taking advantage of the "any\_of" function defined above, the semantics of the  
4609 "any\_of\_any" function are as follows:

```
4610 any_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ]-> Bool
4611 any_of_any f [] ys = "False"
4612 any_of_any f (x:xs) ys = (any_of f x ys) || (any_of_any f xs ys)
```

4613 In the above notation, "f" is the function to be applied and "(x:xs)" represents the first  
4614 element of the list as "x" and the rest of the list as "xs".

4615 For example, the following expression SHALL evaluate to "True":

```
4616 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">
4617   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
4618   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4619     <AttributeValue
4620       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4621     <AttributeValue
4622       DataType="http://www.w3.org/2001/XMLSchema#string">Mary</AttributeValue>
4623   </Apply>
4624   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4625     <AttributeValue
4626       DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4627     <AttributeValue
4628       DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4629     <AttributeValue
4630       DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4631     <AttributeValue
4632       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4633   </Apply>
4634 </Apply>
```

4635 This expression is "True" because at least one of the elements of the first **bag**, namely  
4636 "Ringo", is equal to at least one of the elements of the second **bag**.

4637 • urn:oasis:names:tc:xacml:1.0:function:all-of-any

4638 This function applies a Boolean function between the elements of two **bags**. The  
4639 expression SHALL be "True" if and only if the supplied predicate is 'True' between each  
4640 element of the first **bag** and any element of the second **bag**.

4641 This function SHALL take three arguments. The first argument SHALL be an  
4642 `<xacml:Function>` element that names a Boolean function that takes two arguments of  
4643 primitive types. The second argument SHALL be a **bag** of a primitive data-type. The third  
4644 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated  
4645 as if the "urn:oasis:names:tc:xacml:1.0:function:any-of" function had been applied to each

4646 value of the first **bag** and the whole of the second **bag** using the supplied xacml:Function,  
4647 and the results were then combined using “urn:oasis:names:tc:xacml:1.0:function:and”.

4648 In Haskell, taking advantage of the “any\_of” function defined in Haskell above, the  
4649 semantics of the “all\_of\_any” function are as follows:

```
4650 all_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4651 all_of_any f [] ys = "False"
4652 all_of_any f (x:xs) ys = (any_of f x ys) && (all_of_any f xs ys)
```

4653 In the above notation, “f” is the function to be applied and “(x:xs)” represents the first  
4654 element of the list as “x” and the rest of the list as “xs”.

4655 For example, the following expression SHALL evaluate to "True":

```
4656 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-any">
4657   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater"/>
4658   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4659     <AttributeValue
4660       DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4661     <AttributeValue
4662       DataType="http://www.w3.org/2001/XMLSchema#integer">20</AttributeValue>
4663   </Apply>
4664   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4665     <AttributeValue
4666       DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4667     <AttributeValue
4668       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4669     <AttributeValue
4670       DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4671     <AttributeValue
4672       DataType="http://www.w3.org/2001/XMLSchema#integer">19</AttributeValue>
4673   </Apply>
4674 </Apply>
```

4675 This expression is “True” because each of the elements of the first **bag** is greater than at  
4676 least one of the elements of the second **bag**.

4677 • urn:oasis:names:tc:xacml:1.0:function:any-of-all

4678 This function applies a Boolean function between the elements of two **bags**. The  
4679 expression SHALL be “True” if and only if the supplied predicate is “True” between each  
4680 element of the second **bag** and any element of the first **bag**.

4681 This function SHALL take three arguments. The first argument SHALL be an  
4682 <xacml:Function> element that names a Boolean function that takes two arguments of  
4683 primitive types. The second argument SHALL be a **bag** of a primitive data-type. The third  
4684 argument SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated  
4685 as if the “urn:oasis:names:tc:xacml:1.0:function:any-of” function had been applied to each  
4686 value of the second **bag** and the whole of the first **bag** using the supplied xacml:Function,  
4687 and the results were then combined using “urn:oasis:names:tc:xacml:1.0:function:and”.

4688 In Haskell, taking advantage of the “all\_of” function defined in Haskell above, the semantics  
4689 of the “any\_of\_all” function are as follows:

```
4690 any_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4691 any_of_all f [] ys = "False"
4692 any_of_all f (x:xs) ys = (all_of f x ys) || ( any_of_all f xs ys)
```

4693 In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first  
4694 element of the list as “x” and the rest of the list as “xs”.

4695 For example, the following expression SHALL evaluate to "True":

```

4696 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-all">
4697   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater"/>
4698   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4699     <AttributeValue
4700     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4701     <AttributeValue
4702     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4703   </Apply>
4704   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4705     <AttributeValue
4706     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4707     <AttributeValue
4708     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4709     <AttributeValue
4710     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4711     <AttributeValue
4712     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4713   </Apply>
4714 </Apply>

```

4715       This expression is "True" because, for all of the values in the second **bag**, there is a value  
4716       in the first **bag** that is greater.

4717 • urn:oasis:names:tc:xacml:1.0:function:all-of-all

4718       This function applies a Boolean function between the elements of two **bags**. The  
4719       expression SHALL be "True" if and only if the supplied predicate is "True" between each  
4720       and every element of the first **bag** collectively against all the elements of the second **bag**.

4721       This function SHALL take three arguments. The first argument SHALL be an  
4722       <xacml:Function> element that names a Boolean function that takes two arguments of  
4723       primitive types. The second argument SHALL be a **bag** of a primitive data-type. The third  
4724       argument SHALL be a **bag** of a primitive data-type. The expression is evaluated as if the  
4725       function named in the <xacml:Function> element were applied between every element  
4726       of the second argument and every element of the third argument and the results were  
4727       combined using "urn:oasis:names:tc:xacml:1.0:function:and". The semantics are that the  
4728       result of the expression is "True" if and only if the applied predicate is "True" for *all*  
4729       elements of the first **bag** compared to *all* the elements of the second **bag**.

4730       In Haskell, taking advantage of the "all\_of" function defined in Haskell above, the semantics  
4731       of the "all\_of\_all" function is as follows:

```

4732               all_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4733               all_of_all f [] ys = "False"
4734               all_of_all f (x:xs) ys = (all_of f x ys) && (all_of_all f xs ys)

```

4735       In the above notation, "f" is the function to be applied and "(x:xs)" represents the first  
4736       element of the list as "x" and the rest of the list as "xs".

4737       For example, the following expression SHALL evaluate to "True":



```

4738 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-all">
4739   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater"/>
4740   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4741     <AttributeValue
4742       DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>
4743     <AttributeValue
4744       DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4745   </Apply>
4746   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4747     <AttributeValue
4748       DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4749     <AttributeValue
4750       DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4751     <AttributeValue
4752       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4753     <AttributeValue
4754       DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4755   </Apply>
4756 </Apply>

```

4757 This expression is "True" because all elements of the first **bag**, "5" and "6", are each  
4758 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4759 • urn:oasis:names:tc:xacml:1.0:function:map

4760 This function converts a **bag** of values to another **bag** of values.

4761 This function SHALL take two arguments. The first function SHALL be an  
4762 <xacml:Function> element naming a function that takes a single argument of a primitive  
4763 data-type and returns a value of a primitive data-type. The second argument SHALL be a  
4764 **bag** of a primitive data-type. The expression SHALL be evaluated as if the function named  
4765 in the <xacml:Function> element were applied to each element in the **bag** resulting in a  
4766 **bag** of the converted value. The result SHALL be a **bag** of the primitive data-type that is  
4767 returned by the function named in the <xacml:Function> element.

4768 In Haskell, this function is defined as follows:

```

4769     map:: (a -> b) -> [a] -> [b]
4770     map f [] = []
4771     map f (x:xs) = (f x) : (map f xs)

```

4772 In the above notation, "f" is the function to be applied and "(x:xs)" represents the first  
4773 element of the list as "x" and the rest of the list as "xs".

4774 For example, the following expression,

```

4775 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:map">
4776   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-normalize-
4777     to-lower-case">
4778     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4779       <AttributeValue
4780         DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>
4781       <AttributeValue
4782         DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>
4783     </Apply>
4784   </Apply>

```

4785 evaluates to a **bag** containing "hello" and "world!".

4786

### A.3.13 Regular-expression-based functions

4787 These functions operate on various types using regular expressions and evaluate to  
4788 "http://www.w3.org/2001/XMLSchema#boolean".

- 4789 • urn:oasis:names:tc:xacml:1.0:function:regexp-string-match

4790 This function decides a regular expression match. It SHALL take two arguments of  
4791 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4792 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
4793 expression and the second argument SHALL be a general string. The function  
4794 specification SHALL be that of the "xf:matches" function with the arguments reversed [XF  
4795 Section 6.3.15].

- 4796 • urn:oasis:names:tc:xacml:1.0:function:regexp-uri-match

4797 This function decides a regular expression match. It SHALL take two arguments; the first is  
4798 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type  
4799 "http://www.w3.org/2001/XMLSchema#anyURI". It SHALL return an  
4800 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
4801 expression and the second argument SHALL be a URI. The function SHALL convert the  
4802 second argument to type "http://www.w3.org/2001/XMLSchema#string", then apply  
4803 "urn:oasis:names:tc:xacml:1.0:function:regexp-string-match".

- 4804 • urn:oasis:names:tc:xacml:1.0:function:regexp-ipAddress-match

4805 This function decides a regular expression match. It SHALL take two arguments; the first is  
4806 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type  
4807 "urn:oasis:names:tc:xacml:1.0:data-type:ipAddress". It SHALL return an  
4808 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
4809 expression and the second argument SHALL be an IPv4 or IPv6 address. The function  
4810 SHALL convert the second argument to type "http://www.w3.org/2001/XMLSchema#string",  
4811 then apply "urn:oasis:names:tc:xacml:1.0:function:regexp-string-match".

- 4812 • urn:oasis:names:tc:xacml:1.0:function:regexp-dnsName-match

4813 This function decides a regular expression match. It SHALL take two arguments; the first is  
4814 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type  
4815 "urn:oasis:names:tc:xacml:1.0:data-type:dnsName". It SHALL return an  
4816 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
4817 expression and the second argument SHALL be a DNS name. The function SHALL  
4818 convert the second argument to type "http://www.w3.org/2001/XMLSchema#string", then  
4819 apply "urn:oasis:names:tc:xacml:1.0:function:regexp-string-match".

- 4820 • urn:oasis:names:tc:xacml:1.0:function:regexp-rfc822Name-match

4821 This function decides a regular expression match. It SHALL take two arguments; the first is  
4822 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type  
4823 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name". It SHALL return an  
4824 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
4825 expression and the second argument SHALL be an RFC 822 name. The function SHALL  
4826 convert the second argument to type "http://www.w3.org/2001/XMLSchema#string", then  
4827 apply "urn:oasis:names:tc:xacml:1.0:function:regexp-string-match".

- 4828 • urn:oasis:names:tc:xacml:1.0:function:regexp-x500Name-match

4829 This function decides a regular expression match. It SHALL take two arguments; the first is  
4830 of type "http://www.w3.org/2001/XMLSchema#string" and the second is of type  
4831 "urn:oasis:names:tc:xacml:1.0:data-type:x500Name". It SHALL return an

4832 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
4833 expression and the second argument SHALL be an X.500 directory name. The function  
4834 SHALL convert the second argument to type "http://www.w3.org/2001/XMLSchema#string",  
4835 then apply "urn:oasis:names:tc:xacml:1.0:function:regexp-string-match".

### 4836 **A.3.14 Special match functions**

4837 These functions operate on various types and evaluate to  
4838 "http://www.w3.org/2001/XMLSchema#boolean" based on the specified standard matching  
4839 algorithm.

- 4840 • urn:oasis:names:tc:xacml:1.0:function:x500Name-match

4841 This function shall take two arguments of "urn:oasis:names:tc:xacml:2.0:data-  
4842 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It  
4843 shall return "True" if and only if the first argument matches some terminal sequence of  
4844 RDNs from the second argument when compared using x500Name-equal.

- 4845 • urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match

4846 This function SHALL take two arguments, the first is of data-type  
4847 "http://www.w3.org/2001/XMLSchema#string" and the second is of data-type  
4848 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an  
4849 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if  
4850 the first argument matches the second argument according to the following specification.

4851 An RFC822 name consists of a local-part followed by "@" followed by a domain-part. The  
4852 local-part is case-sensitive, while the domain-part (which is usually a DNS name) is not  
4853 case-sensitive.<sup>4</sup>

4854 The second argument contains a complete rfc822Name. The first argument is a complete  
4855 or partial rfc822Name used to select appropriate values in the second argument as follows.

4856 In order to match a particular address in the second argument, the first argument must  
4857 specify the complete mail address to be matched. For example, if the first argument is  
4858 "Anderson@sun.com", this matches a value in the second argument of  
4859 "Anderson@sun.com" and "Anderson@SUN.COM", but not "Anne.Anderson@sun.com",  
4860 "anderson@sun.com" or "Anderson@east.sun.com".

4861 In order to match any address at a particular domain in the second argument, the first  
4862 argument must specify only a domain name (usually a DNS name). For example, if the first  
4863 argument is "sun.com", this matches a value in the first argument of "Anderson@sun.com"  
4864 or "Baxter@SUN.COM", but not "Anderson@east.sun.com".

4865 In order to match any address in a particular domain in the second argument, the first  
4866 argument must specify the desired domain-part with a leading ".". For example, if the first  
4867 argument is ".east.sun.com", this matches a value in the second argument of  
4868 "Anderson@east.sun.com" and "anne.anderson@ISRG.EAST.SUN.COM" but not  
4869 "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.

4870

### A.3.15 XPath-based functions

4871 This section specifies functions that take XPath expressions for arguments. An XPath expression  
4872 evaluates to a *node-set*, which is a set of XML nodes that match the expression. A node or node-  
4873 set is not in the formal data-type system of XACML. All comparison or other operations on node-  
4874 sets are performed in isolation of the particular function specified. That is, the XPath expressions in  
4875 these functions are restricted to the XACML request **context**. The `<xacml-context:Request>`  
4876 element is the context node for every XPath expression. The following functions are defined:

- 4877 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-count

4878 This function SHALL take an "http://www.w3.org/2001/XMLSchema#string" as an  
4879 argument, which SHALL be interpreted as an XPath expression, and evaluates to an  
4880 "http://www.w3.org/2001/XMLSchema#integer". The value returned from the function  
4881 SHALL be the count of the nodes within the node-set that match the given XPath  
4882 expression.

- 4883 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal

4884 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,  
4885 which SHALL be interpreted as XPath expressions, and SHALL return an  
4886 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if any  
4887 of the XML nodes in the node-set matched by the first argument equals, according to the  
4888 "op:node-equal" function [XF Section 13.1.6], any of the XML nodes in the node-set  
4889 matched by the second argument.

- 4890 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-match

4891 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,  
4892 which SHALL be interpreted as XPath expressions and SHALL return an  
4893 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if  
4894 one of the following two conditions is satisfied: (1) Any of the XML nodes in the node-set  
4895 matched by the first argument is equal, according to "op:node-equal" [XF Section 13.1.6],  
4896 to any of the XML nodes in the node-set matched by the second argument; (2) any attribute  
4897 and element node below any of the XML nodes in the node-set matched by the first  
4898 argument is equal, according to "op:node-equal" [XF Section 13.1.6], to any of the XML  
4899 nodes in the node-set matched by the second argument.

4900 NOTE: The first condition is equivalent to "xpath-node-equal", and guarantees that "xpath-node-  
4901 equal" is a special case of "xpath-node-match".

### 4902 A.3.16 Extension functions and primitive types

4903 Functions and primitive types are specified by string identifiers allowing for the introduction of  
4904 functions in addition to those specified by XACML. This approach allows one to extend the XACML  
4905 module with special functions and special primitive data-types.

4906 In order to preserve the integrity of the XACML evaluation strategy, the result of an extension  
4907 function SHALL depend only on the values of its arguments. Global and hidden parameters SHALL  
4908 NOT affect the evaluation of an expression. Functions SHALL NOT have side effects, as  
4909 evaluation order cannot be guaranteed in a standard way.

---

## 4910 Appendix B. XACML identifiers (normative)

4911 This section defines standard identifiers for commonly used entities.

### 4912 B.1. XACML namespaces

4913 There are currently two defined XACML namespaces.

4914 Policies are defined using this identifier.

4915 `urn:oasis:names:tc:xacml:2.0:policy:schema:cd`

4916 Request and response **contexts** are defined using this identifier.

4917 `urn:oasis:names:tc:xacml:2.0:context:schema:cd`

### 4918 B.2. Access subject categories

4919 This identifier indicates the system entity that initiated the **access** request. That is, the initial entity  
4920 in a request chain. If **subject** category is not specified, this is the default value.

4921 `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

4922 This identifier indicates the system entity that will receive the results of the request (used when it is  
4923 distinct from the access-subject).

4924 `urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject`

4925 This identifier indicates a system entity through which the **access** request was passed. There may  
4926 be more than one. No means is provided to specify the order in which they passed the message.

4927 `urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject`

4928 This identifier indicates a system entity associated with a local or remote codebase that generated  
4929 the request. Corresponding **subject attributes** might include the URL from which it was loaded  
4930 and/or the identity of the code-signer. There may be more than one. No means is provided to  
4931 specify the order in which they processed the request.

4932 `urn:oasis:names:tc:xacml:1.0:subject-category:codebase`

4933 This identifier indicates a system entity associated with the computer that initiated the **access**  
4934 request. An example would be an IPsec identity.

4935 `urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine`

### 4936 B.3. Data-types

4937 The following identifiers indicate data-types that are defined in Section A.2.

4938 `urn:oasis:names:tc:xacml:1.0:data-type:x500Name.`

4939 `urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name`

4940 `urn:oasis:names:tc:xacml:1.0:data-type:ipAddress`

4941 `urn:oasis:names:tc:xacml:1.0:data-type:dnsName`

4942 The following data-type identifiers are defined by XML Schema [XS].

4943 `http://www.w3.org/2001/XMLSchema#string`

4944 `http://www.w3.org/2001/XMLSchema#boolean`

4945 `http://www.w3.org/2001/XMLSchema#integer`



4946 <http://www.w3.org/2001/XMLSchema#double>  
4947 <http://www.w3.org/2001/XMLSchema#time>  
4948 <http://www.w3.org/2001/XMLSchema#date>  
4949 <http://www.w3.org/2001/XMLSchema#dateTime>  
4950 <http://www.w3.org/2001/XMLSchema#anyURI>  
4951 <http://www.w3.org/2001/XMLSchema#hexBinary>  
4952 <http://www.w3.org/2001/XMLSchema#base64Binary>  
4953 The following data-type identifiers correspond to the `dayTimeDuration` and `yearMonthDuration`  
4954 data-types defined in [XF Sections 8.2.2 and 8.2.1, respectively].  
4955 <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration>  
4956 <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration>

## 4957 B.4. Subject attributes

4958 These identifiers indicate *attributes* of a *subject*. When used, they SHALL appear within a  
4959 `<Subject>` element of the request *context*. They SHALL be accessed by means of a  
4960 `<SubjectAttributeDesignator>` element, or an `<AttributeSelector>` element that points  
4961 into a `<Subject>` element of the request *context*.

4962 At most one of each of these attributes is associated with each subject. Each attribute associated  
4963 with authentication included within a single `<Subject>` element relates to the same authentication  
4964 event.

4965 This identifier indicates the name of the *subject*. The default format is  
4966 “<http://www.w3.org/2001/XMLSchema#string>”. To indicate other formats, use the `Data Type`  
4967 attributes listed in B.3

4968 `urn:oasis:names:tc:xacml:1.0:subject:subject-id`

4969 This identifier indicates the *subject* category. “access-subject” is the default value.

4970 `urn:oasis:names:tc:xacml:1.0:subject:category`

4971 This identifier indicates the security domain of the *subject*. It identifies the administrator and policy  
4972 that manages the name-space in which the *subject* id is administered.

4973 `urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier`

4974 This identifier indicates a public key used to confirm the *subject's* identity.

4975 `urn:oasis:names:tc:xacml:1.0:subject:key-info`

4976 This identifier indicates the time at which the *subject* was authenticated.

4977 `urn:oasis:names:tc:xacml:1.0:subject:authentication-time`

4978 This identifier indicates the method used to authenticate the *subject*.

4979 `urn:oasis:names:tc:xacml:1.0:subject:authn-locality:authentication-method`

4980 This identifier indicates the time at which the *subject* initiated the *access* request, according to the  
4981 *PEP*.

4982 `urn:oasis:names:tc:xacml:1.0:subject:request-time`

4983 This identifier indicates the time at which the *subject's* current session began, according to the  
4984 *PEP*.

4985 `urn:oasis:names:tc:xacml:1.0:subject:session-start-time`

4986 The following identifiers indicate the location where authentication credentials were activated. They  
4987 are intended to support the corresponding entities from the SAML authentication statement  
4988 [SAML].

4989 This identifier indicates that the location is expressed as an IP address.



4990 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address  
4991 The corresponding attribute SHALL be of data-type "http://www.w3.org/2001/XMLSchema#string".  
4992 This identifier indicates that the location is expressed as a DNS name.  
4993 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name  
4994 The corresponding attribute SHALL be of data-type "http://www.w3.org/2001/XMLSchema#string".  
4995 Where a suitable attribute is already defined in LDAP [LDAP-1, LDAP-2], the XACML identifier  
4996 SHALL be formed by adding the **attribute** name to the URI of the LDAP specification. For  
4997 example, the **attribute** name for the userPassword defined in the RFC 2256 SHALL be:  
4998 http://www.ietf.org/rfc/rfc2256.txt#userPassword

## 4999 B.6. Resource attributes

5000 These identifiers indicate **attributes** of the **resource**. The corresponding **attributes** MAY appear in  
5001 the <Resource> element of the request **context** and be accessed by means of a  
5002 <ResourceAttributeDesignator> element, or by an <AttributeSelector> element that  
5003 points into the <Resource> element of the request **context**.

5004 This **attribute** identifies the **resource** to which access is requested. If an <xacml-  
5005 context:ResourceContent> element is provided, then the resource to which access is  
5006 requested SHALL be all or a portion of the resource supplied in the <xacml-  
5007 context:ResourceContent> element.

5008 urn:oasis:names:tc:xacml:1.0:resource:resource-id

5009 This **attribute** identifies the namespace of the top element of the contents of the <xacml-  
5010 context:ResourceContent> element. In the case where the **resource** content is supplied in the  
5011 request **context** and the **resource** namespace is defined in the **resource**, the PDP SHALL confirm  
5012 that the namespace defined by this **attribute** is the same as that defined in the **resource**. The type  
5013 of the corresponding **attribute** SHALL be "http://www.w3.org/2001/XMLSchema#anyURI".

5014 urn:oasis:names:tc:xacml:2.0:resource:target-namespace

## 5015 B.7. Action attributes

5016 These identifiers indicate **attributes** of the **action** being requested. When used, they SHALL  
5017 appear within the <Action> element of the request **context**. They SHALL be accessed by means  
5018 of an <ActionAttributeDesignator> element, or an <AttributeSelector> element that  
5019 points into the <Action> element of the request **context**.

5020 This **attribute** identifies the **action** for which **access** is requested.

5021 urn:oasis:names:tc:xacml:1.0:action:action-id

5022 Where the **action** is implicit, the value of the action-id **attribute** SHALL be

5023 urn:oasis:names:tc:xacml:1.0:action:implied-action

5024 This **attribute** identifies the namespace in which the action-id **attribute** is defined.

5025 urn:oasis:names:tc:xacml:1.0:action:action-namespace

## 5026 B.8. Environment attributes

5027 These identifiers indicate *attributes* of the *environment* within which the *decision request* is to be  
5028 evaluated. When used in the *decision request*, they SHALL appear in the <Environment>  
5029 element of the request *context*. They SHALL be accessed by means of an  
5030 <EnvironmentAttributeDesignator> element, or an <AttributeSelector> element that  
5031 points into the <Environment> element of the request *context*.

5032 This identifier indicates the current time at the *context handler*. In practice it is the time at which  
5033 the request *context* was created. For this reason, if these identifiers appear in multiple places  
5034 within a <Policy> or <PolicySet>, then the same value SHALL be assigned to each occurrence  
5035 in the evaluation procedure, regardless of how much time elapses between the processing of the  
5036 occurrences.

5037 urn:oasis:names:tc:xacml:1.0:environment:current-time

5038 The corresponding *attribute* SHALL be of data-type  
5039 "http://www.w3.org/2001/XMLSchema#time".

5040 urn:oasis:names:tc:xacml:1.0:environment:current-date

5041 The corresponding *attribute* SHALL be of data-type  
5042 "http://www.w3.org/2001/XMLSchema#date".

5043 urn:oasis:names:tc:xacml:1.0:environment:current-dateTime

5044 The corresponding *attribute* SHALL be of data-type  
5045 "http://www.w3.org/2001/XMLSchema#dateTime".

## 5046 B.9. Status codes

5047 The following status code values are defined.

5048 This identifier indicates success.

5049 urn:oasis:names:tc:xacml:1.0:status:ok

5050 This identifier indicates that all the attributes necessary to make a policy decision were not available  
5051 (see Section 6.16).

5052 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

5053 This identifier indicates that some attribute value contained a syntax error, such as a letter in a  
5054 numeric field.

5055 urn:oasis:names:tc:xacml:1.0:status:syntax-error

5056 This identifier indicates that an error occurred during policy evaluation. An example would be  
5057 division by zero.

5058 urn:oasis:names:tc:xacml:1.0:status:processing-error

## 5059 B.10. Combining algorithms

5060 The deny-overrides rule-combining algorithm has the following value for the  
5061 ruleCombiningAlgId attribute:

5062 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

5063 The deny-overrides policy-combining algorithm has the following value for the  
5064 policyCombiningAlgId attribute:  
5065 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides  
5066 The permit-overrides rule-combining algorithm has the following value for the  
5067 ruleCombiningAlgId attribute:  
5068 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides  
5069 The permit-overrides policy-combining algorithm has the following value for the  
5070 policyCombiningAlgId attribute:  
5071 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides  
5072 The first-applicable rule-combining algorithm has the following value for the  
5073 ruleCombiningAlgId attribute:  
5074 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable  
5075 The first-applicable policy-combining algorithm has the following value for the  
5076 policyCombiningAlgId attribute:  
5077 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable  
5078 The only-one-applicable-policy policy-combining algorithm has the following value for the  
5079 policyCombiningAlgId attribute:  
5080 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-  
5081 applicable  
5082 The ordered-deny-overrides rule-combining algorithm has the following value for the  
5083 ruleCombiningAlgId attribute:  
5084 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-  
5085 overrides  
5086 The ordered-deny-overrides policy-combining algorithm has the following value for the  
5087 policyCombiningAlgId attribute:  
5088 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-  
5089 overrides  
5090 The ordered-permit-overrides rule-combining algorithm has the following value for the  
5091 ruleCombiningAlgId attribute:  
5092 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-  
5093 overrides  
5094 The ordered-permit-overrides policy-combining algorithm has the following value for the  
5095 policyCombiningAlgId attribute:  
5096 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-  
5097 overrides

---

## 5098 Appendix C. Combining algorithms (normative)

5099 This section contains a description of the *rule-* and *policy-combining algorithms* specified by  
5100 XACML.

### 5101 C.1. Deny-overrides

5102 The following specification defines the "Deny-overrides" *rule-combining algorithm* of a *policy*.

5103 In the entire set of *rules* in the *policy*, if any *rule* evaluates to "Deny", then the result of the  
5104 *rule* combination SHALL be "Deny". If any *rule* evaluates to "Permit" and all other *rules*  
5105 evaluate to "NotApplicable", then the result of the *rule* combination SHALL be "Permit". In  
5106 other words, "Deny" takes precedence, regardless of the result of evaluating any of the  
5107 other *rules* in the combination. If all *rules* are found to be "NotApplicable" to the *decision*  
5108 *request*, then the *rule* combination SHALL evaluate to "NotApplicable".

5109 If an error occurs while evaluating the *target* or *condition* of a *rule* that contains an *effect*  
5110 value of "Deny" then the evaluation SHALL continue to evaluate subsequent *rules*, looking  
5111 for a result of "Deny". If no other *rule* evaluates to "Deny", then the combination SHALL  
5112 evaluate to "Indeterminate", with the appropriate error status.

5113 If at least one *rule* evaluates to "Permit", all other *rules* that do not have evaluation errors  
5114 evaluate to "Permit" or "NotApplicable" and all *rules* that do have evaluation errors contain  
5115 *effects* of "Permit", then the result of the combination SHALL be "Permit".

5116 The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

```
5117 Decision denyOverridesRuleCombiningAlgorithm(Rule rule[])
5118 {
5119     Boolean atLeastOneError = false;
5120     Boolean potentialDeny = false;
5121     Boolean atLeastOnePermit = false;
5122     for( i=0 ; i < lengthOf(rules) ; i++ )
5123     {
5124         Decision decision = evaluate(rule[i]);
5125         if (decision == Deny)
5126         {
5127             return Deny;
5128         }
5129         if (decision == Permit)
5130         {
5131             atLeastOnePermit = true;
5132             continue;
5133         }
5134         if (decision == NotApplicable)
5135         {
5136             continue;
5137         }
5138         if (decision == Indeterminate)
5139         {
5140             atLeastOneError = true;
5141
5142             if (effect(rule[i]) == Deny)
5143             {
5144                 potentialDeny = true;
5145             }
5146             continue;

```

```

5147     }
5148   }
5149   if (potentialDeny)
5150   {
5151     return Indeterminate;
5152   }
5153   if (atLeastOnePermit)
5154   {
5155     return Permit;
5156   }
5157   if (atLeastOneError)
5158   {
5159     return Indeterminate;
5160   }
5161   return NotApplicable;
5162 }

```

5163 The following specification defines the “Deny-overrides” **policy-combining algorithm** of a **policy**  
5164 **set**.

5165 In the entire set of **policies** in the **policy set**, if any **policy** evaluates to "Deny", then the  
5166 result of the **policy** combination SHALL be "Deny". In other words, "Deny" takes  
5167 precedence, regardless of the result of evaluating any of the other **policies** in the **policy**  
5168 **set**. If all **policies** are found to be "NotApplicable" to the **decision request**, then the  
5169 **policy set** SHALL evaluate to "NotApplicable".

5170 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is  
5171 considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set**  
5172 SHALL evaluate to "Deny".

5173 The following pseudo-code represents the evaluation strategy of this **policy-combining algorithm**.

```

5174 Decision denyOverridesPolicyCombiningAlgorithm(Policy policy[])
5175 {
5176   Boolean atLeastOnePermit = false;
5177   for( i=0 ; i < lengthOf(policy) ; i++ )
5178   {
5179     Decision decision = evaluate(policy[i]);
5180     if (decision == Deny)
5181     {
5182       return Deny;
5183     }
5184     if (decision == Permit)
5185     {
5186       atLeastOnePermit = true;
5187       continue;
5188     }
5189     if (decision == NotApplicable)
5190     {
5191       continue;
5192     }
5193     if (decision == Indeterminate)
5194     {
5195       return Deny;
5196     }
5197   }
5198   if (atLeastOnePermit)
5199   {
5200     return Permit;
5201   }
5202   return NotApplicable;
5203 }

```

5204 **Obligations** of the individual **policies** shall be combined as described in Section 7.14.

## 5205 C.2. Ordered-deny-overrides

5206 The following specification defines the "Ordered-deny-overrides" *rule-combining algorithm* of a  
5207 *policy*.

5208 The behavior of this algorithm is identical to that of the Deny-overrides *rule-combining*  
5209 *algorithm* with one exception. The order in which the collection of *rules* is evaluated SHALL  
5210 match the order as listed in the *policy*.

5211 The following specification defines the "Ordered-deny-overrides" *policy-combining algorithm* of a  
5212 *policy set*.

5213 The behavior of this algorithm is identical to that of the Deny-overrides *policy-combining*  
5214 *algorithm* with one exception. The order in which the collection of *policies* is evaluated  
5215 SHALL match the order as listed in the *policy set*.

## 5216 C.3. Permit-overrides

5217 The following specification defines the "Permit-overrides" *rule-combining algorithm* of a *policy*.

5218 In the entire set of *rules* in the *policy*, if any *rule* evaluates to "Permit", then the result of  
5219 the *rule* combination SHALL be "Permit". If any *rule* evaluates to "Deny" and all other  
5220 *rules* evaluate to "NotApplicable", then the *policy* SHALL evaluate to "Deny". In other  
5221 words, "Permit" takes precedence, regardless of the result of evaluating any of the other  
5222 *rules* in the *policy*. If all *rules* are found to be "NotApplicable" to the *decision request*,  
5223 then the *policy* SHALL evaluate to "NotApplicable".

5224 If an error occurs while evaluating the *target* or *condition* of a *rule* that contains an *effect*  
5225 of "Permit" then the evaluation SHALL continue looking for a result of "Permit". If no other  
5226 *rule* evaluates to "Permit", then the *policy* SHALL evaluate to "Indeterminate", with the  
5227 appropriate error status.

5228 If at least one *rule* evaluates to "Deny", all other *rules* that do not have evaluation errors  
5229 evaluate to "Deny" or "NotApplicable" and all *rules* that do have evaluation errors contain  
5230 an *effect* value of "Deny", then the *policy* SHALL evaluate to "Deny".

5231 The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

```
5232 Decision permitOverridesRuleCombiningAlgorithm(Rule rule[])
5233 {
5234     Boolean atLeastOneError = false;
5235     Boolean potentialPermit = false;
5236     Boolean atLeastOneDeny = false;
5237     for( i=0 ; i < lengthOf(rule) ; i++ )
5238     {
5239         Decision decision = evaluate(rule[i]);
5240         if (decision == Deny)
5241         {
5242             atLeastOneDeny = true;
5243             continue;
5244         }
5245         if (decision == Permit)
5246         {
5247             return Permit;
5248         }
5249         if (decision == NotApplicable)
5250         {
5251             continue;
```



```

5252     }
5253     if (decision == Indeterminate)
5254     {
5255         atLeastOneError = true;
5256
5257         if (effect(rule[i]) == Permit)
5258         {
5259             potentialPermit = true;
5260         }
5261         continue;
5262     }
5263 }
5264 if (potentialPermit)
5265 {
5266     return Indeterminate;
5267 }
5268 if (atLeastOneDeny)
5269 {
5270     return Deny;
5271 }
5272 if (atLeastOneError)
5273 {
5274     return Indeterminate;
5275 }
5276 return NotApplicable;
5277 }

```

5278 The following specification defines the "Permit-overrides" **policy-combining algorithm** of a **policy**  
5279 **set**.

5280 In the entire set of **policies** in the **policy set**, if any **policy** evaluates to "Permit", then the  
5281 result of the **policy** combination SHALL be "Permit". In other words, "Permit" takes  
5282 precedence, regardless of the result of evaluating any of the other **policies** in the **policy**  
5283 **set**. If all **policies** are found to be "NotApplicable" to the **decision request**, then the  
5284 **policy set** SHALL evaluate to "NotApplicable".

5285 If an error occurs while evaluating the **target** of a **policy**, a reference to a **policy** is  
5286 considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set**  
5287 SHALL evaluate to "Indeterminate", with the appropriate error status, provided no other  
5288 **policies** evaluate to "Permit" or "Deny".

5289 The following pseudo-code represents the evaluation strategy of this **policy-combining algorithm**.

```

5290 Decision permitOverridesPolicyCombiningAlgorithm(Policy policy[])
5291 {
5292     Boolean atLeastOneError = false;
5293     Boolean atLeastOneDeny = false;
5294     for( i=0 ; i < lengthOf(policy) ; i++ )
5295     {
5296         Decision decision = evaluate(policy[i]);
5297         if (decision == Deny)
5298         {
5299             atLeastOneDeny = true;
5300             continue;
5301         }
5302         if (decision == Permit)
5303         {
5304             return Permit;
5305         }
5306         if (decision == NotApplicable)
5307         {
5308             continue;
5309         }

```

```

5310     if (decision == Indeterminate)
5311     {
5312         atLeastOneError = true;
5313         continue;
5314     }
5315 }
5316 if (atLeastOneDeny)
5317 {
5318     return Deny;
5319 }
5320 if (atLeastOneError)
5321 {
5322     return Indeterminate;
5323 }
5324 return NotApplicable;
5325 }

```

5326 **Obligations** of the individual **policies** shall be combined as described in Section 7.14.

## 5327 C.4. Ordered-permit-overrides

5328 The following specification defines the "Ordered-permit-overrides" **rule-combining algorithm** of a  
5329 **policy**.

5330 The behavior of this algorithm is identical to that of the Permit-overrides **rule-combining**  
5331 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL  
5332 match the order as listed in the **policy**.

5333 The following specification defines the "Ordered-permit-overrides" **policy-combining algorithm** of  
5334 a **policy set**.

5335 The behavior of this algorithm is identical to that of the Permit-overrides **policy-combining**  
5336 **algorithm** with one exception. The order in which the collection of **policies** is evaluated  
5337 SHALL match the order as listed in the **policy set**.

## 5338 C.5. First-applicable

5339 The following specification defines the "First-Applicable " **rule-combining algorithm** of a **policy**.

5340 Each **rule** SHALL be evaluated in the order in which it is listed in the **policy**. For a  
5341 particular **rule**, if the **target** matches and the **condition** evaluates to "True", then the  
5342 evaluation of the **policy** SHALL halt and the corresponding **effect** of the **rule** SHALL be the  
5343 result of the evaluation of the **policy** (i.e. "Permit" or "Deny"). For a particular **rule** selected  
5344 in the evaluation, if the **target** evaluates to "False" or the **condition** evaluates to "False",  
5345 then the next **rule** in the order SHALL be evaluated. If no further **rule** in the order exists,  
5346 then the **policy** SHALL evaluate to "NotApplicable".

5347 If an error occurs while evaluating the **target** or **condition** of a **rule**, then the evaluation  
5348 SHALL halt, and the **policy** shall evaluate to "Indeterminate", with the appropriate error  
5349 status.

5350 The following pseudo-code represents the evaluation strategy of this **rule-combining algorithm**.

5351

5352

```

5353 Decision firstApplicableEffectRuleCombiningAlgorithm(Rule rule[])
5354 {
5355     for( i = 0 ; i < lengthOf(rule) ; i++ )
5356     {
5357         Decision decision = evaluate(rule[i]);
5358         if (decision == Deny)
5359         {
5360             return Deny;
5361         }
5362         if (decision == Permit)
5363         {
5364             return Permit;
5365         }
5366         if (decision == NotApplicable)
5367         {
5368             continue;
5369         }
5370         if (decision == Indeterminate)
5371         {
5372             return Indeterminate;
5373         }
5374     }
5375     return NotApplicable;
5376 }

```

5377 The following specification defines the "First-applicable" **policy-combining algorithm** of a **policy**  
5378 **set**.

5379 Each **policy** is evaluated in the order that it appears in the **policy set**. For a particular  
5380 **policy**, if the **target** evaluates to "True" and the **policy** evaluates to a determinate value of  
5381 "Permit" or "Deny", then the evaluation SHALL halt and the **policy set** SHALL evaluate to  
5382 the **effect** value of that **policy**. For a particular **policy**, if the **target** evaluate to "False", or  
5383 the **policy** evaluates to "NotApplicable", then the next **policy** in the order SHALL be  
5384 evaluated. If no further **policy** exists in the order, then the **policy set** SHALL evaluate to  
5385 "NotApplicable".

5386 If an error were to occur when evaluating the **target**, or when evaluating a specific **policy**,  
5387 the reference to the **policy** is considered invalid, or the **policy** itself evaluates to  
5388 "Indeterminate", then the evaluation of the **policy-combining algorithm** shall halt, and the  
5389 **policy set** shall evaluate to "Indeterminate" with an appropriate error status.

5390 The following pseudo-code represents the evaluation strategy of this **policy-combination**  
5391 **algorithm**.

```

5392 Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy policy[])
5393 {
5394     for( i = 0 ; i < lengthOf(policy) ; i++ )
5395     {
5396         Decision decision = evaluate(policy[i]);
5397         if(decision == Deny)
5398         {
5399             return Deny;
5400         }
5401         if(decision == Permit)
5402         {
5403             return Permit;
5404         }
5405         if (decision == NotApplicable)
5406         {
5407             continue;
5408         }
5409         if (decision == Indeterminate)
5410         {

```

```

5411         return Indeterminate;
5412     }
5413 }
5414     return NotApplicable;
5415 }

```

5416 **Obligations** of the individual **policies** shall be combined as described in Section 7.14.

## 5417 C.6. Only-one-applicable

5418 The following specification defines the "Only-one-applicable" **policy-combining algorithm** of a  
5419 **policy set**.

5420 In the entire set of **policies** in the **policy set**, if no **policy** is considered applicable by virtue  
5421 of its **target**, then the result of the **policy** combination algorithm SHALL be "NotApplicable".  
5422 If more than one **policy** is considered applicable by virtue of its **target**, then the result of  
5423 the **policy** combination algorithm SHALL be "Indeterminate".

5424 If only one **policy** is considered applicable by evaluation of its **target**, then the result of the  
5425 **policy-combining algorithm** SHALL be the result of evaluating the **policy**.

5426 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is  
5427 considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set**  
5428 SHALL evaluate to "Indeterminate", with the appropriate error status.

5429 The following pseudo-code represents the evaluation strategy of this policy combining algorithm.

```

5430 Decision onlyOneApplicablePolicyPolicyCombiningAlgoithm(Policy policy[])
5431 {
5432     Boolean          atLeastOne      = false;
5433     Policy           selectedPolicy = null;
5434     ApplicableResult appResult;
5435
5436     for ( i = 0; i < lengthOf(policy) ; i++ )
5437     {
5438         appResult = isApplicable(policy[i]);
5439
5440         if ( appResult == Indeterminate )
5441         {
5442             return Indeterminate;
5443         }
5444         if( appResult == Applicable )
5445         {
5446             if ( atLeastOne )
5447             {
5448                 return Indeterminate;
5449             }
5450             else
5451             {
5452                 atLeastOne      = true;
5453                 selectedPolicy = policy[i];
5454             }
5455         }
5456         if ( appResult == NotApplicable )
5457         {
5458             continue;
5459         }
5460     }
5461     if ( atLeastOne )
5462     {

```

```
5463     return evaluate(selectedPolicy);
5464 }
5465 else
5466 {
5467     return NotApplicable;
5468 }
5469 }
5470
```

5471

---

## Appendix D. Acknowledgments

5472

The following individuals contributed to the development of the specification:

5473

Anne Anderson

5474

Anthony Nadalin

5475

Bill Parducci

5476

Carlisle Adams

5477

Daniel Engovatov

5478

Don Flinn

5479

Ed Coyne

5480

Ernesto Damiani

5481

Frank Siebenlist

5482

Gerald Brose

5483

Hal Lockhart

5484

James MacLean

5485

John Merrells

5486

Ken Yagen

5487

Konstantin Beznosov

5488

Michiharu Kudo

5489

Michael McIntosh

5490

Pierangela Samarati

5491

Pirasenna Velandai Thiyagarajan

5492

Polar Humenn

5493

Rebekah Metz

5494

Ron Jacobson

5495

Satoshi Hada

5496

Sekhar Vajjhala

5497

Seth Proctor

5498

Simon Godik

5499

Steve Anderson

5500

Steve Crocker

5501

Suresh Damodaran

5502

Tim Moses

5503

Von Welch

5504



5505

---

## Appendix E. Revision history

Rev	Date	By whom	What
CD 01	16 Sep 2004	Access Control TC	First committee draft
CD 02	30 Sep 2004	Access Control TC	Updated list of editors

5506

---

5507

## Appendix F. Notices

5508 OASIS takes no position regarding the validity or scope of any intellectual property or other rights  
5509 that might be claimed to pertain to the implementation or use of the technology described in this  
5510 document or the extent to which any license under such rights might or might not be available;  
5511 neither does it represent that it has made any effort to identify any such rights. Information on  
5512 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS  
5513 website. Copies of claims of rights made available for publication and any assurances of licenses to  
5514 be made available, or the result of an attempt made to obtain a general license or permission for  
5515 the use of such proprietary rights by implementers or users of this specification, can be obtained  
5516 from the OASIS Executive Director.

5517 OASIS has been notified of intellectual property rights claimed in regard to some or all of the  
5518 contents of this specification. For more information consult the online list of claimed rights.

5519 OASIS invites any interested party to bring to its attention any copyrights, patents or patent  
5520 applications, or other proprietary rights which may cover technology that may be required to  
5521 implement this specification. Please address the information to the OASIS Executive Director.

5522 Copyright (C) OASIS Open 2004. All Rights Reserved.

5523 This document and translations of it may be copied and furnished to others, and derivative works  
5524 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
5525 published and distributed, in whole or in part, without restriction of any kind, provided that the above  
5526 copyright notice and this paragraph are included on all such copies and derivative works. However,  
5527 this document itself may not be modified in any way, such as by removing the copyright notice or  
5528 references to OASIS, except as needed for the purpose of developing OASIS specifications, in  
5529 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights  
5530 document must be followed, or as required to translate it into languages other than English.

5531 The limited permissions granted above are perpetual and will not be revoked by OASIS or its  
5532 successors or assigns.

5533 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
5534 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
5535 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY  
5536 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
5537 PARTICULAR PURPOSE.