



1

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

4 **Committee Specification 1.0, 6 November**
5 **2002**

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

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

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

9 Editors:

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

12 Contributors:

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

32 Abstract:

33 This specification defines an XML schema for an extensible access-control policy
34 language.

35 Status:

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

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

42

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

| | | |
|----|---|----|
| 44 | Table of contents | |
| 45 | 1. Introduction (non-normative) | 9 |
| 46 | 1.1. Glossary | 9 |
| 47 | 1.1.1 Preferred terms | 9 |
| 48 | 1.1.2 Related terms | 10 |
| 49 | 1.2. Notation | 10 |
| 50 | 1.3. Schema organization and namespaces | 11 |
| 51 | 2. Background (non-normative) | 11 |
| 52 | 2.1. Requirements | 12 |
| 53 | 2.2. Rule and policy combining | 13 |
| 54 | 2.3. Combining algorithms | 13 |
| 55 | 2.4. Multiple subjects | 14 |
| 56 | 2.5. Policies based on subject and resource attributes | 14 |
| 57 | 2.6. Multi-valued attributes | 14 |
| 58 | 2.7. Policies based on resource contents | 15 |
| 59 | 2.8. Operators | 15 |
| 60 | 2.9. Policy distribution | 16 |
| 61 | 2.10. Policy indexing | 16 |
| 62 | 2.11. Abstraction layer | 16 |
| 63 | 2.12. Actions performed in conjunction with enforcement | 17 |
| 64 | 3. Models (non-normative) | 17 |
| 65 | 3.1. Data-flow model | 17 |
| 66 | 3.2. XACML context | 19 |
| 67 | 3.3. Policy language model | 19 |
| 68 | 3.3.1 Rule | 20 |
| 69 | 3.3.2 Policy | 22 |
| 70 | 3.3.3 Policy set | 23 |
| 71 | 4. Examples (non-normative) | 23 |
| 72 | 4.1. Example one | 24 |
| 73 | 4.1.1 Example policy | 24 |
| 74 | 4.1.2 Example request context | 26 |
| 75 | 4.1.3 Example response context | 27 |
| 76 | 4.2. Example two | 28 |
| 77 | 4.2.1 Example medical record instance | 28 |
| 78 | 4.2.2 Example request context | 29 |
| 79 | 4.2.3 Example plain-language rules | 31 |

| | | | |
|-----|-------|---|----|
| 80 | 4.2.4 | Example XACML rule instances | 31 |
| 81 | 5. | Policy syntax (normative, with the exception of the schema fragments) | 45 |
| 82 | 5.1. | Element <PolicySet> | 45 |
| 83 | 5.2. | Element <Description> | 46 |
| 84 | 5.3. | Element <PolicySetDefaults> | 46 |
| 85 | 5.4. | Element <XPathVersion> | 47 |
| 86 | 5.5. | Element <Target> | 47 |
| 87 | 5.6. | Element <Subjects> | 48 |
| 88 | 5.7. | Element <Subject> | 48 |
| 89 | 5.8. | Element <AnySubject> | 48 |
| 90 | 5.9. | Element <SubjectMatch> | 48 |
| 91 | 5.10. | Element <Resources> | 49 |
| 92 | 5.11. | Element <Resource> | 49 |
| 93 | 5.12. | Element <AnyResource> | 50 |
| 94 | 5.13. | Element <ResourceMatch> | 50 |
| 95 | 5.14. | Element <Actions> | 51 |
| 96 | 5.15. | Element <Action> | 51 |
| 97 | 5.16. | Element <AnyAction> | 51 |
| 98 | 5.17. | Element <ActionMatch> | 52 |
| 99 | 5.18. | Element <PolicySetIdReference> | 52 |
| 100 | 5.19. | Element <PolicyIdReference> | 52 |
| 101 | 5.20. | Element <Policy> | 53 |
| 102 | 5.21. | Element <Rule> | 54 |
| 103 | 5.22. | Simple type EffectType | 55 |
| 104 | 5.23. | Element <Condition> | 55 |
| 105 | 5.24. | Element <Apply> | 55 |
| 106 | 5.25. | Element <Function> | 56 |
| 107 | 5.26. | Complex type AttributeDesignatorType | 57 |
| 108 | 5.27. | Element <ResourceAttributeDesignator> | 57 |
| 109 | 5.28. | Element <ActionAttributeDesignator> | 59 |
| 110 | 5.29. | Element <EnvironmentAttributeDesignator> | 60 |
| 111 | 5.30. | Element <ResourceAttributesPresent> | 61 |
| 112 | 5.31. | Element <ActionAttributesPresent> | 62 |
| 113 | 5.32. | Element <EnvironmentAttributesPresent> | 63 |
| 114 | 5.33. | Complex type SubjectAttributeDesignatorType | 64 |
| 115 | 5.34. | Element <SubjectAttributesPresent> | 65 |
| 116 | 5.35. | Element <AttributeSelector> | 66 |

| | | | |
|-----|--------|---|----|
| 117 | 5.36. | Element <AttributeValue> | 67 |
| 118 | 5.37. | Element <Obligations> | 67 |
| 119 | 5.38. | Element <Obligation> | 67 |
| 120 | 5.39. | Element <AttributeAssignment> | 68 |
| 121 | 6. | Context syntax (normative with the exception of the schema fragments) | 69 |
| 122 | 6.1. | Element <Request> | 69 |
| 123 | 6.2. | Element <Subject> | 70 |
| 124 | 6.3. | Element <Resource> | 70 |
| 125 | 6.4. | Element <ResourceContent> | 71 |
| 126 | 6.5. | Element <Action> | 71 |
| 127 | 6.6. | Element <Environment> | 71 |
| 128 | 6.7. | Element <Attribute> | 72 |
| 129 | 6.8. | Element <AttributeValue> | 72 |
| 130 | 6.9. | Element <Response> | 73 |
| 131 | 6.10. | Element <Result> | 73 |
| 132 | 6.11. | Element <Decision> | 74 |
| 133 | 6.12. | Element <Status> | 74 |
| 134 | 6.13. | Element <StatusCode> | 75 |
| 135 | 6.14. | Element <StatusMessage> | 75 |
| 136 | 6.15. | Element <StatusDetail> | 75 |
| 137 | 7. | Functional requirements (normative) | 76 |
| 138 | 7.1. | Policy enforcement point | 76 |
| 139 | 7.2. | Base policy | 77 |
| 140 | 7.3. | Target evaluation | 77 |
| 141 | 7.4. | Condition evaluation | 77 |
| 142 | 7.5. | Rule evaluation | 77 |
| 143 | 7.6. | Policy evaluation | 78 |
| 144 | 7.7. | Policy Set evaluation | 78 |
| 145 | 7.8. | Hierarchical resources | 79 |
| 146 | 7.9. | Attributes | 80 |
| 147 | 7.9.1. | Attribute Matching | 80 |
| 148 | 7.9.2. | Attribute Retrieval | 80 |
| 149 | 7.9.3. | Environment Attributes | 81 |
| 150 | 7.9.4. | Subject Attributes | 81 |
| 151 | 7.10. | Authorization decision | 81 |
| 152 | 7.11. | Obligations | 81 |
| 153 | 8. | XACML extensibility points (non-normative) | 82 |

| | | | |
|-----|-------------|--|----|
| 154 | 8.1. | Extensible XML attribute types | 82 |
| 155 | 8.2. | Extensible XACML attribute types | 82 |
| 156 | 8.3. | Structured attributes | 83 |
| 157 | 9. | Security and privacy considerations (non-normative) | 83 |
| 158 | 9.1. | Threat model | 83 |
| 159 | 9.1.1. | Unauthorized disclosure | 84 |
| 160 | 9.1.2. | Message replay | 84 |
| 161 | 9.1.3. | Message insertion | 84 |
| 162 | 9.1.4. | Message deletion | 84 |
| 163 | 9.1.5. | Message modification | 84 |
| 164 | 9.1.6. | Not-applicable results | 84 |
| 165 | 9.1.7. | Negative rules | 85 |
| 166 | 9.2. | Safeguards | 86 |
| 167 | 9.2.1. | Authentication | 86 |
| 168 | 9.2.2. | Policy administration | 86 |
| 169 | 9.2.3. | Confidentiality | 86 |
| 170 | 9.2.4. | Policy integrity | 87 |
| 171 | 9.2.5. | Policy identifiers | 87 |
| 172 | 9.2.6. | Trust model | 88 |
| 173 | 9.2.7. | Privacy | 88 |
| 174 | 10. | Conformance (normative) | 88 |
| 175 | 10.1. | Introduction | 88 |
| 176 | 10.2. | Attestation | 89 |
| 177 | 10.3. | Conformance tables | 89 |
| 178 | 10.3.1. | Schema elements | 89 |
| 179 | 10.3.2. | Identifier Prefixes | 90 |
| 180 | 10.3.3. | Algorithms | 90 |
| 181 | 10.3.4. | Status Codes | 90 |
| 182 | 10.3.5. | Attributes | 91 |
| 183 | 10.3.6. | Identifiers | 91 |
| 184 | 10.3.7. | Data Types | 91 |
| 185 | 10.3.8. | Functions | 92 |
| 186 | 11. | References | 94 |
| 187 | Appendix A. | Standard data types, functions and their semantics (normative) | 96 |
| 188 | A.1. | Introduction | 96 |
| 189 | A.2. | Primitive types | 96 |
| 190 | A.3. | Structured types | 97 |

| | | | |
|-----|--|--|-----|
| 191 | A.4. | Representations | 97 |
| 192 | A.5. | Bags | 98 |
| 193 | A.6. | Expressions | 98 |
| 194 | A.7. | Element <AttributeValue> | 99 |
| 195 | A.8. | Elements <AttributeDesignator> and <AttributeSelector> | 99 |
| 196 | A.9. | Element <Apply> | 99 |
| 197 | A.10. | Element <Condition> | 99 |
| 198 | A.11. | Element <Function> | 100 |
| 199 | A.12. | Matching elements | 100 |
| 200 | A.13. | Arithmetic evaluation | 101 |
| 201 | A.14. | XACML standard functions | 102 |
| 202 | A14.1 | Equality predicates | 102 |
| 203 | A14.2 | Arithmetic functions | 104 |
| 204 | A14.3 | String conversion functions | 105 |
| 205 | A14.4 | Numeric type conversion functions | 105 |
| 206 | A14.5 | Logical functions | 105 |
| 207 | A14.6 | Arithmetic comparison functions | 106 |
| 208 | A14.7 | Date and time arithmetic functions | 107 |
| 209 | A14.8 | Non-numeric comparison functions | 108 |
| 210 | A14.9 | Bag functions | 110 |
| 211 | A14.10 | Set functions | 111 |
| 212 | A14.11 | Higher-order bag functions | 111 |
| 213 | A14.12 | Special match functions | 118 |
| 214 | A14.13 | XPath-based functions | 119 |
| 215 | A14.14 | Extension functions and primitive types | 119 |
| 216 | Appendix B. XACML identifiers (normative) | | 120 |
| 217 | B.1. | XACML namespaces | 120 |
| 218 | B.2. | Access subject categories | 120 |
| 219 | B.3. | XACML functions | 120 |
| 220 | B.4. | Data types | 121 |
| 221 | B.5. | Subject attributes | 121 |
| 222 | B.6. | Resource attributes | 122 |
| 223 | B.7. | Action attributes | 123 |
| 224 | B.8. | Environment attributes | 123 |
| 225 | B.9. | Status codes | 123 |
| 226 | B.10. | Combining algorithms | 124 |
| 227 | Appendix C. Combining algorithms (normative) | | 125 |

| | | |
|-----|------------------------------|-----|
| 228 | C.1. Deny-overrides | 125 |
| 229 | C.2. Permit-overrides | 127 |
| 230 | C.3. First-applicable | 129 |
| 231 | C.4. Only-one-applicable | 130 |
| 232 | Appendix D. Acknowledgments | 132 |
| 233 | Appendix E. Revision history | 133 |
| 234 | Appendix F. Notices | 134 |
| 235 | | |

237 1. Introduction (non-normative)

238 1.1. Glossary

239 1.1.1 Preferred terms

240 **Access** - Performing an *action*

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

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

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

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

247 **Authorization decision** - The result of evaluating *applicable policy*, returned by the *PDP* to the
248 *PEP*. A function that evaluates to "Permit", "Deny", "Indeterminate" or "Not-applicable", and
249 (optionally) a set of *obligations*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

285 1.1.2 Related terms

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

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

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

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

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

293 1.2. Notation

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

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

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

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

305 Listings of XACML schemas appear like this.

306
307 Example code listings appear like this.

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

- 311 • The prefix `saml:` stands for the SAML assertion namespace [SAML].
- 312 • The prefix `ds:` stands for the W3C XML Signature namespace [DS].
- 313 • The prefix `xs:` stands for the W3C XML Schema namespace [XS].
- 314 • The prefix `xf:` stands for the XPath query and function specification namespace [XF].

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

318 1.3. Schema organization and namespaces

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

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

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

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

323 XACML data-types are defined in the following XML namespace:

324 `urn:oasis:names:tc:xacml:1.0:data-type`

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

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

328 2. Background (non-normative)

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

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

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

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

355 2.1. Requirements

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

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

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

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

380 2.2. Rule and policy combining

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

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

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

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

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

400 2.3. Combining algorithms

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

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

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

416 combining algorithm, the combined result is the same as the result of evaluating the first <Rule>,
417 <Policy> or <PolicySet> element in the list of **rules** whose **target** is applicable to the **decision**
418 **request**. The "Only-one-applicable" **policy-combining algorithm** only applies to **policies**. The
419 result of this combining algorithm ensures that one and only one **policy** or **policy set** is applicable
420 by virtue of their **targets**. If no **policy** or **policy set** applies, then the result is "Not-applicable", but
421 if more than one **policy** or **policy set** is applicable, then the result is "Indeterminate". When exactly
422 one **policy** or **policy set** is applicable, the result of the combining algorithm is the result of
423 evaluating the single **applicable policy** or **policy set**.

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

425 **2.4. Multiple subjects**

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

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

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

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

449 **2.6. Multi-valued attributes**

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

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

458 2.7. Policies based on resource contents

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

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

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

469 2.8. Operators

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

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

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

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

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

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

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

499 2.9. Policy distribution

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

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

511 2.10. Policy indexing

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

517 Two approaches are supported:

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

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

528 2.11. Abstraction layer

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

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

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

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

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

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

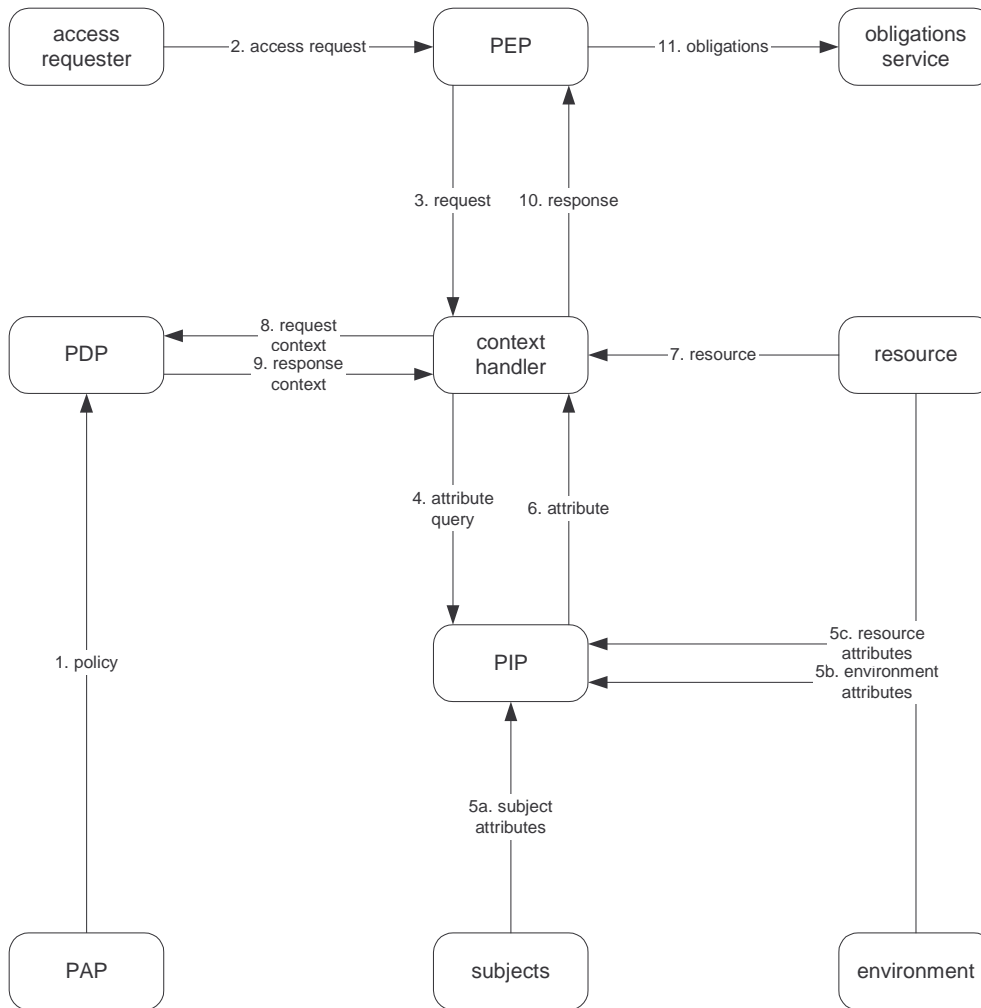
552 In many applications, policies specify actions that MUST be performed, either instead of, or in
553 addition to, actions that MAY be performed. This idea was described by Sloman [Sloman94].
554 XACML provides facilities to specify actions that MUST be performed in conjunction with policy
555 evaluation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

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

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

563 **3.1. Data-flow model**

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



565

566

Figure 1 - Data-flow diagram

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

572 The model operates by the following steps.

573 1. **PAPs** write **policies** and make them available to the **PDP**. Its **policies** represent the complete
 574 policy for a specified **target**.

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

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

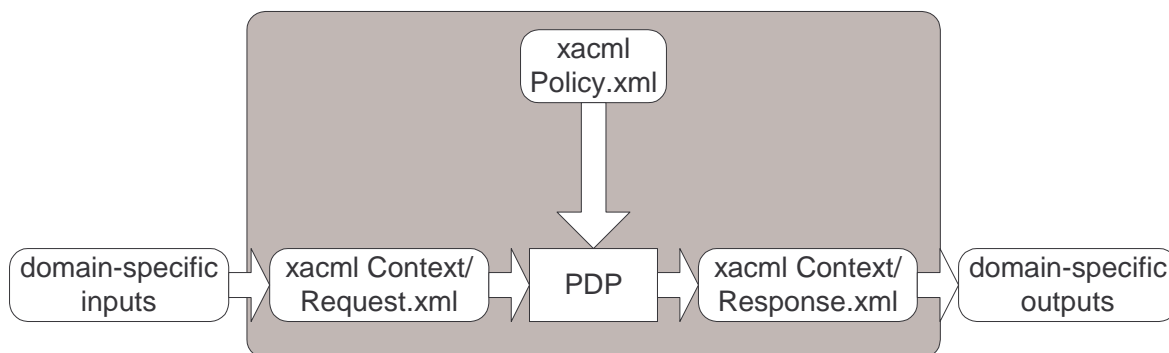
579 4. **Subject**, **resource** and **environment attributes** may be requested from a **PIP**.

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

- 581 6. The **PIP** returns the requested **attributes** to the **context handler**.
- 582 7. Optionally, the **context handler** includes the **resource** in the **context**.
- 583 8. The **context handler** makes information about the request **context** available to the **PDP**. The
- 584 **PDP** identifies the **policy** applicable to the request **context**. The **PDP** evaluates the **policy**.
- 585 9. The **PDP** returns the response **context** (including the **authorization decision**) to the **context**
- 586 **handler**.
- 587 10. The **context handler** translates the response **context** to the native response format of the
- 588 **PEP**. The **context handler** returns the response to the **PEP**.
- 589 11. The **PEP** fulfills the **obligations**.
- 590 12. (Not shown) If **access** is permitted, then the **PEP** permits **access** to the **resource**; otherwise, it
- 591 denies **access**.

592 3.2. XACML context

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



604

605

Figure 2 - XACML context

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

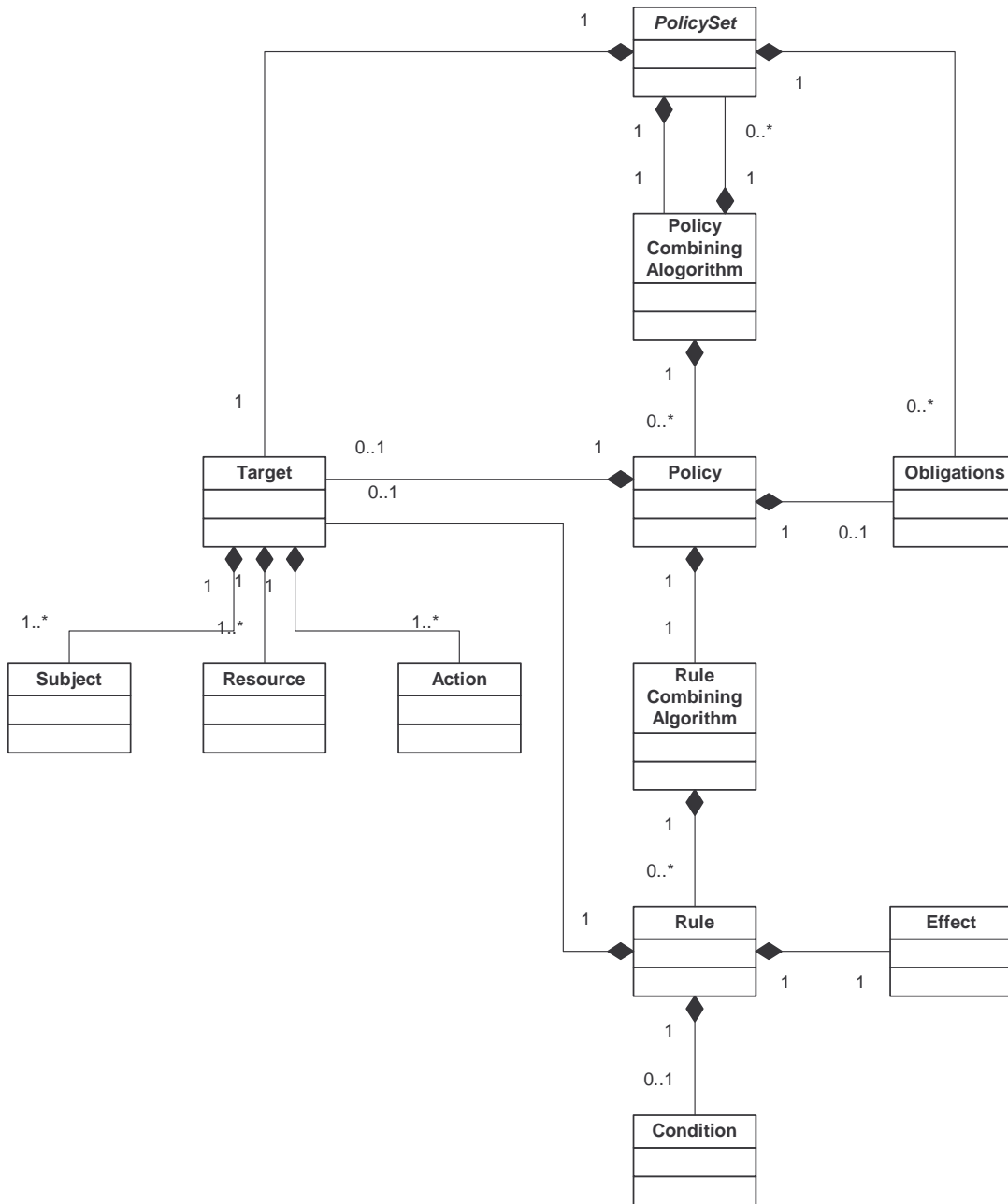
607 See Section 7.9 for a more detailed discussion of the request **context**.

608 3.3. Policy language model

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

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

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



614

615

Figure 3 - Policy language model

616

3.3.1 Rule

617 The main components of a *rule* are:

- 618 • a *target*,

- 619 • an *effect*; and
620 • a *condition*.

621 These are discussed in the following sub-sections.

622 3.3.1.1. Rule target

623 The *target* defines the set of:

- 624 • *resources*;
625 • *subjects*; and
626 • *actions*

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

634 The `<Target>` element may be absent from a `<Rule>`. In this case, the `<Rule>` inherits its *target*
635 from the parent `<Policy>` element.

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

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

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

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

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

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

660 All three options are supported in XACML.

661

3.3.1.2. Effect

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

664

3.3.1.3. Condition

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

667

3.3.2 Policy

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

- 670 • a **target**;
- 671 • a **rule-combining algorithm**-identifier;
- 672 • a set of **rules**; and
- 673 • **obligations**.

674 **Rules** are described above. The remaining components are described in the following sub-
675 sections.

676

3.3.2.1. Policy target

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

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

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

698 3.3.2.2. Rule-combining algorithm

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

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

704 3.3.2.3. Obligations

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

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

709 3.3.3 Policy set

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

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

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

717 3.3.3.1. Policy-combining algorithm

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

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

723 3.3.3.2. Obligations

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

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

728 4. Examples (non-normative)

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

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

733 4.1. Example one

734 4.1.1 Example policy

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

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

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

741 The header for this policy is

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

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

744 Line [p02] introduces the XACML Policy itself.

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

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

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

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

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

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

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



```
[p20] </Actions>
[p21] </Target>
```

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

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

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

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

764 Line [p24] says what **effect** this **rule** has if the **rule** evaluates to "True". **Rules** can have an **effect**
765 of either "Permit" or "Deny". In this case, the rule will evaluate to "Permit", meaning that, as far as
766 this one **rule** is concerned, the requested **access** should be permitted. If a **rule** evaluates to
767 "False", then it returns a result of "Not-applicable". If an error occurs when evaluating the **rule**, the
768 **rule** returns a result of "Indeterminate". As mentioned above, the **rule-combining algorithm** for
769 the **policy** tells how various **rule** values are combined into a single **policy** value.

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

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

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

771 Line [p29] introduces the **target** of the **rule**. As described above for the **target** of a policy, the
772 **target** of a **rule** describes the **decision requests** to which this **rule** applies. If the **subject**,
773 **resource** and **action** in a **decision request** do not match the values specified in the **rule target**,
774 then the remainder of the **rule** does not need to be evaluated, and a value of "Not-applicable" is
775 returned to the **policy** evaluation.

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

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

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

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

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

789 4.1.2 Example request context

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

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

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

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

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

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

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

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

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

```
[c18] <Action>  
[c19] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"  
[c20]   DataType="http://www.w3.org/2001/XMLSchema#string">  
[c21]   <AttributeValue>read</AttributeValue>  
[c22] </Attribute>  
[c23] </Action>
```

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

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

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

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

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

821 4.1.3 Example response context

822 As a result, there is no **rule** in this **policy** that returns a "Permit" result for this request. The **rule-**
823 **combining algorithm** for the **policy** specifies that, in this case, a result of "Not-applicable" should
824 be returned. The response **context** looks as follows:

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

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

```
[r05] <Result>  
[r06] <Decision>Not-applicable</Decision>  
[r07] </Result>
```

827 The <Result> element in lines [r05-r07] contains the result of evaluating the **decision request**
828 against the **policy**. In this case, the result is `"Not-applicable"`. A **policy** can return `"Permit"`, `"Deny"`,
829 `"Not-applicable"` or `"Indeterminate"`.

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

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

831

4.2. Example two

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

835

4.2.1 Example medical record instance

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

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

```

888     <zip>12345</zip>
889     <phone>555.123.9012</phone>
890     <fax>555.123.9013</fax>
891     <email/>
892   </physicianContact>
893   <registrationID>ABC123</registrationID>
894 </primaryCarePhysician>
895 <insurer>
896   <name>Blue Cross</name>
897   <street>1234 Main St</street>
898   <city>Springfield</city>
899   <state>MA</state>
900   <zip>12345</zip>
901   <phone>555.123.5678</phone>
902   <fax>555.123.5679</fax>
903   <email/>
904 </insurer>
905 <medical>
906   <treatment>
907     <drug>
908       <name>methylphenidate hydrochloride</name>
909       <dailyDosage>30mgs</dailyDosage>
910       <startDate>1999-01-12</startDate>
911     </drug>
912     <comment>patient exhibits side-effects of skin coloration and carpal
913 degeneration</comment>
914   </treatment>
915   <result>
916     <test>blood pressure</test>
917     <value>120/80</value>
918     <date>2001-06-09</date>
919     <performedBy>Nurse Betty</performedBy>
920   </result>
921 </medical>
922 </record>

```

923 4.2.2 Example request context

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

```

927 [01] <?xml version="1.0" encoding="UTF-8"?>
928 [02] <Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
929 [03] xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy"
930 [04] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
931 [05] <Subject>
932 [06]   <Attribute AttributeId=
933 [07]     "urn:oasis:names:tc:xacml:1.0:subject:subject-category"
934 [08]     DataType="http://www.w3.org/2001/XMLSchema#string"
935 [09]     Issuer="www.medico.com"
936 [10]     IssueInstant="2001-12-17T09:30:47-05:00">
937 [11]   <AttributeValue>
938 [12]     urn:oasis:names:tc:xacml:1.0:subject:category:access-subject
939 [13]   </AttributeValue>
940 [14] </Attribute>
941 [15] <Attribute AttributeId=
942 [16]   "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
943 [17]   DataType=
944 [18]   "urn:oasis:names:tc:xacml:1.0:data-type:x500name"
945 [19]   Issuer="www.medico.com"
946 [20]   IssueInstant="2001-12-17T09:30:47-05:00">

```

```

947 [21]     <AttributeValue>CN=Julius Hibbert</AttributeValue>
948 [22]   </Attribute>
949 [23]   <Attribute AttributeId=
950 [24]     "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
951 [25]     DataType="http://www.w3.org/2001/XMLSchema#string"
952 [26]     Issuer="www.medico.com"
953 [27]     IssueInstant="2001-12-17T09:30:47-05:00">
954 [28]   <AttributeValue>physician</AttributeValue>
955 [29] </Attribute>
956 [30]   <Attribute AttributeId=
957 [31]     "urn:oasis:names:tc:xacml:1.0:example:attribute:physician-id"
958 [32]     DataType="http://www.w3.org/2001/XMLSchema#string"
959 [33]     Issuer="www.medico.com"
960 [34]     IssueInstant="2001-12-17T09:30:47-05:00">
961 [35]   <AttributeValue>jh1234</AttributeValue>
962 [36] </Attribute>
963 [37] </Subject>
964 [38] <Resource>
965 [39]   <ResourceContent>
966 [40]     <md:record
967 [41]       xmlns:md="//http:www.medico.com/schemas/record.xsd">
968 [42]       <md:patient>
969 [43]         <md:patientDoB>1992-03-21</md:patientDoB>
970 [44]       </md:patient>
971 [45]       <!-- other fields -->
972 [46]     </md:record>
973 [47]   </ResourceContent>
974 [48]   <Attribute AttributeId=
975 [49]     "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
976 [50]     DataType="http://www.w3.org/2001/XMLSchema#string">
977 [55]   <AttributeValue>
978 [56]     //medico.com/records/bart-simpson.xml#
979 [57]     xmlns(md="//http:www.medico.com/schemas/record.xsd)
980 [58]     xpointer(/md:record/md:patient/md:patientDoB)
981 [59]   </AttributeValue>
982 [60] </Attribute>
983 [61]   <Attribute AttributeId=
984 [62]     "urn:oasis:names:tc:xacml:1.0:resource:xpath"
985 [63]     DataType="http://www.w3.org/2001/XMLSchema#string">
986 [64]   <AttributeValue>
987 [65]     xmlns(md=http:www.medico.com/schemas/record.xsd)
988 [66]     xpointer(/md:record/md:patient/md:patientDoB)
989 [67]   </AttributeValue>
990 [68] </Attribute>
991 [69]   <Attribute AttributeId=
992 [70]     "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
993 [71]     DataType="http://www.w3.org/2001/XMLSchema#string">
994 [72]   <AttributeValue>
995 [73]     http://www.medico.com/schemas/record.xsd
996 [74]   </AttributeValue>
997 [75] </Attribute>
998 [76] </Resource>
999 [77] <Action>
1000 [78]   <Attribute AttributeId=
1001 [79]     "urn:oasis:names:tc:xacml:1.0:action:action-id"
1002 [80]     DataType="http://www.w3.org/2001/XMLSchema#string">
1003 [81]   <AttributeValue>read</AttributeValue>
1004 [82] </Attribute>
1005 [83] </Action>
1006 [84] </Request>

```

1007 [02]-[04] Standard namespace declarations.

1008 [05]-[37] **Subject** attributes are placed in the `Subject` section of the `Request`. Each **attribute**
 1009 consists of the **attribute** meta-data and the **attribute** value.

1010 [06]-[14] Each `Subject` section must have one and only one subject-category **attribute**. The
 1011 value of this **attribute** describes the role that the **subject** plays in making the **decision request**.
 1012 The value of “`access-subject`” denotes the identity for which the request was issued.

1013 [15]-[22] **Subject** `subject-id` **attribute**.

1014 [23]-[29] **Subject** `role` **attribute**.

1015 [30]-[36] **Subject** `physician-id` **attribute**.

1016 [38]-[69] **Resource** attributes are placed in the `Resource` section of the `Request`. Each **attribute**
 1017 consists of **attribute** meta-data and an **attribute** value.

1018 [39]-[47] **Resource** content. The XML document that is being requested is placed here.

1019 [48]-[60] **Resource** identifier.

1020 [56]-[58] The **Resource** is identified with an Xpointer expression that names the URI of the file that
 1021 is accessed, the target namespace of the document, and the XPath location path to the specific
 1022 element.

1023 [61]-[68] The XPath location path in the “`resource-id`” attribute is extracted and placed in the
 1024 `xpath` attribute.

1025 [69]-[75] **Resource** `target-namespace` **attribute**.

1026 [77]-[84] **Action** **attributes** are placed in the `Action` section of the `Request`.

1027 [78]-[82] **Action** identifier.

1028 4.2.3 Example plain-language rules

1029 The following plain-language rules are to be enforced:

1030 Rule 1: A person may read any record for which he or she is the designated patient.

1031 Rule 2: A person may read any record for which he or she is the designated parent or
 1032 guardian, and for which the patient is under 16 years of age.

1033 Rule 3: A physician may write to any medical element for which he or she is the designated
 1034 primary care physician, provided an email is sent to the patient.

1035 Rule 4: An administrator shall not be permitted to read or write to medical elements of a
 1036 patient record.

1037 These **rules** may be written by different **PAPs** operating independently, or by a single **PAP**.

1038 4.2.4 Example XACML rule instances

1039 4.2.4.1. Rule 1

1040 Rule 1 illustrates a simple **rule** with a single `<Condition>` element. The following XACML
 1041 `<Rule>` instance expresses Rule 1:

```
1042 [01] <?xml version="1.0" encoding="UTF-8"?>
1043 [02] <Rule
```

```

1044 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1045 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1046 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1047 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1048 [07]   RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"
1049 [08]   Effect="Permit">
1050 [09]   <Description>
1051 [10]     A person may read any medical record in the
1052 [11]     http://www.medico.com/schemas/record.xsd namespace
1053 [12]     for which he or she is a designated patient
1054 [13]   </Description>
1055 [14]   <Target>
1056 [15]     <Subjects>
1057 [16]       <AnySubject/>
1058 [17]     </Subjects>
1059 [18]     <Resources>
1060 [20]       <Resource>
1061 [21]         <!-- match document target namespace -->
1062 [22]         <ResourceMatch
1063 [23] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1064 [24]           <ResourceAttributeDesignator AttributeId=
1065 [25]             "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1066 [26]           DataType="http://www.w3.org/2001/XMLSchema#string"/>
1067 [27]             <AttributeValue
1068 [28]               DataType="http://www.w3.org/2001/XMLSchema#string">
1069 [29]                 http://www.medico.com/schemas/record.xsd
1070 [30]             </AttributeValue>
1071 [31]           </ResourceMatch>
1072 [32]         <!-- match requested xml element -->
1073 [33]         <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-
1074 [34] node-match">
1075 [35]           <ResourceAttributeDesignator AttributeId=
1076 [36]             "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1077 [37]           DataType="http://www.w3.org/2001/XMLSchema#string"/>
1078 [38]           <AttributeValue
1079 [39]             DataType="http://www.w3.org/2001/XMLSchema#string"/>/md:record</AttributeValue>
1080 [40]           </ResourceMatch>
1081 [41]         </Resource>
1082 [42]       </Resources>
1083 [43]     <Actions>
1084 [44]       <Action>
1085 [45]         <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1086 [46] equal">
1087 [47]           <ActionAttributeDesignator AttributeId=
1088 [48]             "urn:oasis:names:tc:xacml:1.0:action:action-id"
1089 [49]           DataType="http://www.w3.org/2001/XMLSchema#string"/>
1090 [50]           <AttributeValue
1091 [51]             DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1092 [52]           </ActionMatch>
1093 [53]         </Action>
1094 [54]       </Actions>
1095 [55]     </Target>
1096 [56]   <!-- compare policy number in the document with
1097 [57]     policy-number attribute -->
1098 [58]   <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1099 [59]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-
1100 [60] only">
1101 [61]       <!-- policy-number attribute -->
1102 [62]       <SubjectAttributeDesignator AttributeId=
1103 [63]         "urn:oasis:names:tc:xacml:1.0:examples:attribute:policy-number"
1104 [64]       DataType="http://www.w3.org/2001/XMLSchema#string"/>
1105 [65]     </Apply>

```



```

1106 [55] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-
1107 only">
1108 [56] <!-- policy number in the document -->
1109 [57] <AttributeSelector RequestContextPath=
1110 [58] " //md:record/md:patient/md:policyNumber"
1111 DataType="http://www.w3.org/2001/XMLSchema#string">
1112 [59] </AttributeSelector>
1113 [60] </Apply>
1114 [61] </Condition>
1115 [62] </Rule>

```

1116 [02]-[06]. XML namespace declarations.

1117 [07] **Rule** identifier.

1118 [08]. When a **rule** evaluates to 'True' it emits the value of the **Effect** attribute. This value is
1119 combined with the **Effect** values of other rules according to the **rule-combining algorithm**.

1120 [09]-[13] Free form description of the **rule**.

1121 [14]-[46]. A **rule target** defines a set of **decision requests** that are applicable to the **rule**. A
1122 decision request, such that the value of the
1123 "urn:oasis:names:tc:xacml:1.0:resource:target-namespace" **resource attribute** is
1124 equal to "<http://www.medico.com/schema/records.xsd>" and the value of the
1125 "urn:oasis:names:tc:xacml:1.0:resource:xpath" **resource attribute** matches the XPath
1126 expression /md:record and the value of the
1127 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** is equal to "read",
1128 matches the **target** of this **rule**.

1129 [15]-[17]. The **Subjects** element may contain either a **disjunctive sequence** of **Subject**
1130 elements or **AnySubject** element.

1131 [16] The **AnySubject** element is a special element that matches any **subject** in the request
1132 **context**.

1133 [18]-[36]. The **Resources** element may contain either a **disjunctive sequence** of **Resource**
1134 elements or **AnyResource** element.

1135 [20]-[35] The **Resource** element encloses the **conjunctive sequence** of **ResourceMatch**
1136 elements.

1137 [22]-[28] The **ResourceMatch** element compares its first and second child elements according to
1138 the matching function. A match is positive if any of the values selected by the first argument match
1139 the explicit value of the second argument. This match compares the target namespace of the
1140 requested document with the value of "<http://www.medico.com/schema.records.xsd>".

1141 [22] The **MatchId** attribute names the matching function.

1142 [23]-[24] The **ResourceAttributeDesignator** element selects the **resource attribute** values
1143 from the request **context**. The **attribute** name is specified by the **AttributeId**. The selection
1144 result is a **bag** of values.

1145 [25]-[27] Literal attribute value to match.

1146 [30]-[34] The **ResourceMatch**. This match compares the results of two XPath expressions. The
1147 first XPath expression is the location path to the requested xml element and the second XPath
1148 expression is /md:record. The "xpath-node-match" function evaluates to "True" if the requested
1149 XML element is below the /md:record element.

1150 [30] **MatchId** attribute names the matching function.

1151 [31]-[32] The ResourceAttributeDesignator selects the **bag** of values for the
1152 “urn:oasis:names:tc:xacml:1.0:xpath” **resource attribute**. Here, there is just one
1153 element in the **bag**, which is the location path for the requested XML element.

1154 [33] The literal XPath expression to match. The md prefix is resolved using a standard namespace
1155 declaration.

1156 [37]-[45] The Actions element may contain either a **disjunctive sequence** of Action elements
1157 or an AnyAction element.

1158 [38]-[44] The Action element contains a **conjunctive sequence** of ActionMatch elements.

1159 [39]-[43] The ActionMatch element compares its first and second child elements according to the
1160 matching function. Match is positive, if any of the values selected by the first argument match
1161 explicit value of the second argument. In this case, the value of the action-id action attribute in
1162 the request **context** is compared with the value “read”.

1163 [39] The MatchId attribute names the matching function.

1164 [40]-[41] The ActionAttributeDesignator selects **action attribute** values from the request
1165 **context**. The **attribute** name is specified by the AttributeId. The selection result is a **bag** of
1166 values. “urn:oasis:names:tc:xacml:1.0:action:action-id” is the predefined name for
1167 the action identifier.

1168 [42] The **Attribute** value to match. This is an **action** name.

1169 [49]-[61] The Condition element. A **condition** must evaluate to “True” for the **rule** to be
1170 applicable. This condition evaluates the truth of the statement: the policy-number **subject**
1171 **attribute** is equal to the policy number in the XML document.

1172 [49] The FunctionId attribute of the Condition element names the function to be used for
1173 comparison. In this case, comparison is done with function:string-equal; this function takes
1174 two arguments of the “http://www.w3.org/2001/XMLSchema#string” type.

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

1183 [52]-[53] The SubjectAttributeDesignator selects a **bag** of values for the policy-number
1184 **subject attribute** in the request **context**.

1185 [55] The second argument to the “function:string-equal” in the Condition. Functions can
1186 take other functions as arguments. The Apply element encodes function call with the
1187 FunctionId attribute naming the function. Since “function:string-equal” takes arguments
1188 of the “http://www.w3.org/2001/XMLSchema#string” type and the AttributeSelector
1189 selects a **bag** of “http://www.w3.org/2001/XMLSchema#string” values,
1190 “function:string-one-and-only” is used. This function guarantees that its argument
1191 evaluates to a **bag** containing one and only one
1192 “http://www.w3.org/2001/XMLSchema#string” element.

1193 [57] The AttributeSelector element selects a **bag** of values from the request **context**. The
1194 AttributeSelector is a free-form XPath pointing device into the request **context**. The

1195 RequestContextPath attribute specifies an XPath expression over the content of the requested
1196 XML document, selecting the policy number. Note that the namespace prefixes in the XPath
1197 expression are resolved with the standard XML namespace declarations.

1198 4.2.4.2. Rule 2

1199 Rule 2 illustrates the use of a mathematical function, i.e. the <Apply> element with functionId
1200 "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" to calculate date. It also
1201 illustrates the use of **predicate** expressions, with the functionId
1202 "urn:oasis:names:tc:xacml:1.0:function:and".

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


1310 [48] The Condition is using the “function:and” function. This is a boolean function that takes
1311 one or more boolean arguments (2 in this case) and performs the logical “AND” operation to
1312 compute the truth value of the expression.

1313 [51]-[65] The truth of the first part of the condition is evaluated: The requestor is the designated
1314 parent or guardian. The Apply element contains a function invocation. The function name is
1315 contained in the FunctionId attribute. The comparison is done with “function:string-
1316 equal” that takes 2 arguments of “http://www.w3.org/2001/XMLSchema#string” type.

1317 [52] Since “function:string-equal” takes arguments of the
1318 “http://www.w3.org/2001/XMLSchema#string” type, “function:string-one-and-
1319 only” is used to ensure that the **subject attribute**
1320 “urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id” in the request **context**
1321 contains one and only one value. “Function:string-equal” takes an argument expression
1322 that evaluates to a **bag** of “http://www.w3.org/2001/XMLSchema#string” values.

1323 [54] Value of the **subject attribute**
1324 “urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id” is
1325 selected from the request **context** with the SubjectAttributeDesignator element. This
1326 expression evaluates to a bag of “http://www.w3.org/2001/XMLSchema#string” values.

1327 [58] “function:string-one-and-only” is used to ensure that the **bag** of values selected by
1328 its argument contains one and only one value of type
1329 “http://www.w3.org/2001/XMLSchema#string”.

1330 [60] The value of the md:parentGuardianId element is selected from the **resource** content with
1331 the AttributeSelector element. AttributeSelector is a free-form XPath expression,
1332 pointing into the request **context**. The RequestContextPath XML attribute contains an XPath
1333 expression over the request **context**. Note that all namespace prefixes in the XPath expression
1334 are resolved with standard namespace declarations. The AttributeSelector evaluates to the
1335 **bag** of values of type “http://www.w3.org/2001/XMLSchema#string”.

1336 [66]-[83] The expression: “the patient is under 16 years of age” is evaluated. The patient is under
1337 16 years of age if the current date is less than the date computed by adding 16 to the patient’s date
1338 of birth.

1339 [66] “function:date-less-or-equal” is used to compute the difference of two dates.

1340 [67] “function:date-one-and-only” is used to ensure that the **bag** of values selected by its
1341 argument contains one and only one value of type
1342 “http://www.w3.org/2001/XMLSchema#date”.

1343 [68]-[69] Current date is evaluated by selecting the
1344 “urn:oasis:names:tc:xacml:1.0:environment:current-date” **environment attribute**.

1345 [71] “function:date-add-yearMonthDuration” is used to compute the date by adding 16 to
1346 the patient’s date of birth. The first argument is a
1347 “http://www.w3.org/2001/XMLSchema#date”, and the second argument is an
1348 “xf:yearMonthDuration”.

1349 [73] “function:date-one-and-only” is used to ensure that the **bag** of values selected by its
1350 argument contains one and only one value of type
1351 “http://www.w3.org/2001/XMLSchema#date”.

1352 [75]-[76] The <AttributeSelector> element selects the patient’s date of birth by taking the
1353 XPath expression over the document content.

1354 [79]-[81] Year Month Duration of 16 years.

1355

4.2.4.3. Rule 3

1356

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

1357

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

1358

<Policy> element.

1359

```
[01] <?xml version="1.0" encoding="UTF-8"?>
1360 [02] <Policy
1361 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1362 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1363 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1364 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1365 [07]   PolicyId="urn:oasis:names:tc:xacml:examples:policyid:3"
1366 [08]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
1367 [09]     rule-combining-algorithm:deny-overrides">
1368 [10]   <Description>
1369 [11]     Policy for any medical record in the
1370 [12]     http://www.medico.com/schemas/record.xsd namespace
1371 [13]   </Description>
1372 [14]   <Target>
1373 [15]     <Subjects>
1374 [16]       <AnySubject/>
1375 [17]     </Subjects>
1376 [18]     <Resources>
1377 [19]       <Resource>
1378 [20]         <!-- match document target namespace -->
1379 [21]         <ResourceMatch
1380 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1381 [22]           <ResourceAttributeDesignator AttributeId=
1382 [23]             "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1383 [24]           DataType="http://www.w3.org/2001/XMLSchema#string"/>
1384 [25]           <AttributeValue
1385 [26]             DataType="http://www.w3.org/2001/XMLSchema#string">
1386 [27]             http://www.medico.com/schemas/record.xsd
1387 [28]           </AttributeValue>
1388 [29]         </ResourceMatch>
1389 [30]       </Resource>
1390 [31]     </Resources>
1391 [32]     <Actions>
1392 [33]       <AnyAction/>
1393 [34]     </Actions>
1394 [35]   </Target>
1395 [36] <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:3"
1396 [37]   Effect="Permit">
1397 [38]   <Description>
1398 [39]     A physician may write any medical element in a record
1399 [40]     for which he or she is the designated primary care
1400 [41]     physician, provided an email is sent to the patient
1401 [42]   </Description>
1402 [43]   <Target>
1403 [44]     <Subjects>
1404 [45]       <Subject>
1405 [46]         <!-- match subject group attribute -->
1406 [47]         <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1407 [48]           equal">
1408 [49]           <SubjectAttributeDesignator AttributeId=
1409 [50]             "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
1410 [51]           DataType="http://www.w3.org/2001/XMLSchema#string"/>
1411 [52]           <AttributeValue
1412 [53]             DataType="http://www.w3.org/2001/XMLSchema#string">physician</AttributeValue>
1413 [54]           </SubjectMatch>
1414 [55]         </Subject>
```

```

1415 [51] </Subjects>
1416 [52] <Resources>
1417 [53] <Resource>
1418 [54] <!-- match requested xml element -->
1419 [55] <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-
1420 node-match">
1421 [56] <ResourceAttributeDesignator AttributeId=
1422 [57] "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1423 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1424 [58] <AttributeValue
1425 DataType="http://www.w3.org/2001/XMLSchema#string">
1426 [59] /md:record/md:medical
1427 [60] </AttributeValue>
1428 [61] </ResourceMatch>
1429 [62] </Resource>
1430 [63] </Resources>
1431 [64] <Actions>
1432 [65] <Action>
1433 [66] <!-- match action -->
1434 [67] <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1435 equal">
1436 [68] <ActionAttributeDesignator AttributeId=
1437 [069] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1438 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1439 [070] <AttributeValue
1440 DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
1441 [071] </ActionMatch>
1442 [072] </Action>
1443 [073] </Actions>
1444 [074] </Target>
1445 [075] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1446 equal">
1447 [076] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1448 and-only">
1449 [077] <!-- physician-id subject attribute -->
1450 [078] <SubjectAttributeDesignator AttributeId=
1451 [079] "urn:oasis:names:tc:xacml:1.0:example:
1452 [080] attribute:physician-id"
1453 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1454 [081] </Apply>
1455 [082] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1456 and-only">
1457 [083] <AttributeSelector RequestContextPath=
1458 [084] "//md:record/md:primaryCarePhysician/md:registrationID"
1459 [085] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1460 [086] </Apply>
1461 [087] </Condition>
1462 [089] </Rule>
1463 [090] <Obligations>
1464 [091] <!-- send e-mail message to the document owner -->
1465 [092] <Obligation ObligationId=
1466 [093] "urn:oasis:names:tc:xacml:example:obligation:email"
1467 [094] FulfillOn="Permit">
1468 [095] <AttributeAssignment AttributeId=
1469 [096] "urn:oasis:names:tc:xacml:1.0:example:attribute:mailto"
1470 [097] DataType="http://www.w3.org/2001/XMLSchema#string">
1471 [098] <AttributeSelector RequestContextPath=
1472 [099] "//md:/record/md:patient/md:patientContact/md:email"
1473 [100] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1474 [101] </AttributeAssignment>
1475 [102] <AttributeAssignment AttributeId=
1476 [103] "urn:oasis:names:tc:xacml:1.0:example:attribute:text"
1477 [104] DataType="http://www.w3.org/2001/XMLSchema#string">

```

```

1478 [105] <AttributeValue>
1479 [106]     Your medical record has been accessed by:
1480 [107] </AttributeValue>
1481 [108] </AttributeAssignment>
1482 [109] <AttributeAssignment AttributeId=
1483 [110]     "urn:oasis:names:tc:xacml:example:attribute:text"
1484 [111]     DataType="http://www.w3.org/2001/XMLSchema#string">
1485 [112]     <SubjectAttributeDesignator AttributeId=
1486 [113]     "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
1487 [114]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1488 [114] </AttributeAssignment>
1489 [115] </Obligation>
1490 [116] </Obligations>
1491 [117] </Policy>

```

1492 [01]-[09] The `Policy` element includes standard namespace declarations as well as policy specific
1493 parameters, such as `PolicyId` and `RuleCombiningAlgId`.

1494 [07] **Policy** identifier. This parameter is used for the inclusion of the `Policy` in the `PolicySet`
1495 element.

1496 [08]-[09] **Rule combining algorithm** identifier. This parameter is used to compute the combined
1497 outcome of **rule effects** for **rules** that are applicable to the **decision request**.

1498 [10-13] Free-form description of the **policy**.

1499 [14]-[33] **Policy target**. The **policy target** defines a set of applicable decision requests. The
1500 structure of the `Target` element in the `Policy` is identical to the structure of the `Target` element
1501 in the `Rule`. In this case, the **policy target** is a set of all XML documents conforming to the
1502 "<http://www.medico.com/schemas/record.xsd>" target namespace. For the detailed description of
1503 the `Target` element see Rule 1, section 4.2.4.1.

1504 [34]-[89] The only `Rule` element included in this `Policy`. Two parameters are specified in the **rule**
1505 header: `RuleId` and `Effect`. For the detailed description of the `Rule` structure see Rule 1,
1506 section 4.2.4.1.

1507 [41]-[74] A **rule target** narrows down a **policy target**. **Decision requests** with the value of
1508 "urn:oasis:names:tc:xacml:1.0:example:attribute:role" **subject attribute** equal to
1509 "physician" [42]-[51], and that access elements of the medical record that "xpath-node-match"
1510 the "/md:record/md:medical" XPath expression [52]-[63], and that have the value of the
1511 "urn:oasis:names:tc:xacml:1.0:action:action-id" **action attribute** equal to "read".

1512 [65]-[73] match the **target** of this **rule**. For a detailed description of the rule target see example 1,
1513 section 4.2.4.1.

1514 [75]-[87] The `Condition` element. For the **rule** to be applicable to the authorization request,
1515 **condition** must evaluate to True. This **rule condition** compares the value of the
1516 "urn:oasis:names:tc:xacml:1.0:examples:attribute:physician-id" **subject**
1517 **attribute** with the value of the `physician id` element in the medical record that is being
1518 accessed. For a detailed explanation of rule condition see Rule 1, section 4.2.4.1.

1519 [90]-[116] The `Obligations` element. **Obligations** are a set of operations that must be
1520 performed by the **PEP** in conjunction with an **authorization decision**. An **obligation** may be
1521 associated with a positive or negative **authorization decision**.

1522 [92]-[115] The `Obligation` element consists of the `ObligationId`, the authorization decision
1523 value for which it must fulfill, and a set of attribute assignments.

1524 [92]-[93] `ObligationId` identifies an **obligation**. **Obligation** names are not interpreted by the
1525 **PDP**.

1526 [94] `FulfillOn` attribute defines an **authorization decision** value for which this **obligation** must
1527 be fulfilled.

1528 [95]-[101] **Obligation** may have one or more parameters. The **obligation** parameter
1529 "urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto" is assigned the value
1530 from the content of the xml document.

1531 [95-96] `AttributeId` declares
1532 "urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto" **obligation** parameter.

1533 [97] The **obligation** parameter data type is defined.

1534 [98]-[100] The **obligation** parameter value is selected from the content of the XML document that is
1535 being accessed with the XPath expression over request **context**.

1536 [102]-[108] The **obligation** parameter
1537 "urn:oasis:names:tc:xacml:1.0:examples:attribute:text" of type
1538 "http://www.w3.org/2001/XMLSchema#string" is assigned the literal value "Your
1539 medical record has been accessed by:"

1540 [109]-[114] The **obligation** parameter
1541 "urn:oasis:names:tc:xacml:1.0:examples:attribute:text" of the
1542 "http://www.w3.org/2001/XMLSchema#string" data type is assigned the value of the
1543 "urn:oasis:names:tc:xacml:1.0:subject:subject-id" **subject attribute**.

4.2.4.4. Rule 4

1544 Rule 4 illustrates the use of the "Deny" Effect value, and a Rule with no Condition element.

```

1546 [01] <?xml version="1.0" encoding="UTF-8"?>
1547 [02] <Rule
1548 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1549 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1550 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1551 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1552 [07]   RuleId="urn:oasis:names:tc:xacml:example:ruleid:4"
1553 [08]   Effect="Deny">
1554 [09]   <Description>
1555 [10]     An Administrator shall not be permitted to read or write
1556 [11]     medical elements of a patient record in the
1557 [12]     http://www.medico.com/records.xsd namespace.
1558 [13]   </Description>
1559 [14]   <Target>
1560 [15]     <Subjects>
1561 [16]       <Subject>
1562 [17]         <!-- match role subject attribute -->
1563 [18]         <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1564 [19] equal">
1565 [19]           <SubjectAttributeDesignator AttributeId=
1566 [20]             "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
1567 [21]           DataType="http://www.w3.org/2001/XMLSchema#string"/>
1568 [21]           <AttributeValue
1569 [22]             DataType="http://www.w3.org/2001/XMLSchema#string">administrator</AttributeValue>
1570 [22]           </SubjectMatch>
1571 [23]         </Subject>
1572 [24]       </Subjects>
1573 [25]     </Target>
1574 [25]   </Resources>

```

```

1574 [26]     <Resource>
1575 [27]     <!-- match document target namespace -->
1576 [28]     <ResourceMatch
1577 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1578 [29]         <ResourceAttributeDesignator AttributeId=
1579 [30]         "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1580 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1581 [31]         <AttributeValue
1582 DataType="http://www.w3.org/2001/XMLSchema#string">
1583 [32]             http://www.medico.com/schemas/record.xsd
1584 [33]         </AttributeValue>
1585 [34]     </ResourceMatch>
1586 [35]     <!-- match requested xml element -->
1587 [36]     <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-
1588 node-match">
1589 [37]         <ResourceAttributeDesignator AttributeId=
1590 [38]         "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1591 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1592 [39]         <AttributeValue
1593 DataType="http://www.w3.org/2001/XMLSchema#string">
1594 [40]             /md:record/md:medical
1595 [41]         </AttributeValue>
1596 [42]     </ResourceMatch>
1597 [43] </Resource>
1598 [44] </Resources>
1599 [45] <Action>
1600 [46]     <Action>
1601 [47]     <!-- match 'read' action -->
1602 [48]     <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1603 equal">
1604 [49]         <ActionAttributeDesignator AttributeId=
1605 [50]         "urn:oasis:names:tc:xacml:1.0:action:action-id"
1606 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1607 [51]         <AttributeValue
1608 DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1609 [52]     </ActionMatch>
1610 [53] </Action>
1611 [54] <Action>
1612 [55]     <!-- match 'write' action -->
1613 [56]     <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
1614 equal">
1615 [57]         <ActionAttributeDesignator AttributeId=
1616 [58]         "urn:oasis:names:tc:xacml:1.0:action:action-id"
1617 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1618 [59]         <AttributeValue
1619 DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
1620 [60]     </ActionMatch>
1621 [61] </Action>
1622 [62] </Actions>
1623 [63] </Target>
1624 [64] </Rule>

```

1625 [01]-[08] The Rule element declaration. The most important parameter here is Effect. See Rule
1626 1, section 4.2.4.1 for a detailed explanation of the Rule structure.

1627 [08] **Rule** Effect. Every **rule** that evaluates to “True” emits **rule effect** as its value that will be
1628 combined later on with other **rule effects** according to the **rule combining algorithm**. This **rule**
1629 Effect is “Deny” meaning that according to this rule, access must be denied.

1630 [09]-[13] Free form description of the **rule**.

1631 [14]-[63] **Rule target.** The **Rule target** defines a set of **decision requests** that are applicable to
1632 the **rule**. This **rule** is matched by:

- 1633 • a **decision request** with **subject attribute**
1634 "urn:oasis:names:tc:xacml:1.0:examples:attribute:role" equal to
1635 "administrator";
- 1636 • the value of **resource attribute**
1637 "urn:oasis:names:tc:xacml:1.0:resource:target-namespace" is equal to
1638 "http://www.medico.com/schemas/record.xsd"
- 1639 • the value of the requested XML element matches the XPath expression
1640 "/md:record/md:medical";
- 1641 • the value of **action attribute** "urn:oasis:names:tc:xacml:1.0:action:action-id" is equal to
1642 "read"

1643 See Rule 1, section 4.2.4.1 for the detailed explanation of the Target element.

1644 This **rule** does not have a Condition element.

1645 4.2.4.5. Example PolicySet

1646 This section uses the examples of the previous sections to illustrate the process of combining
1647 **policies**. The policy governing read access to medical elements of a record is formed from each of
1648 the four **rules described in Section 4.2.3**. In plain language, the combined rule is:

- 1649 • Either the requestor is the patient; or
- 1650 • the requestor is the parent or guardian and the patient is under 16; or
- 1651 • the requestor is the primary care physician and a notification is sent to the patient; and
- 1652 • the requestor is not an administrator.

1653 The following XACML <PolicySet> illustrates the combined **policies**. **Policy 3** is included by
1654 reference and **policy 2** is explicitly included.

```
1655 [01] <?xml version="1.0" encoding="UTF-8"?>
1656 [02] <PolicySet
1657 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1658 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1659 [05]   PolicySetId=
1660 [06]     "urn:oasis:names:tc:xacml:1.0:examples:policysetid:1"
1661 [07]   PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
1662 [071] policy-combining-algorithm:deny-overrides"/>
1663 [08] <Description>
1664 [09]   Example policy set.
1665 [10] </Description>
1666 [11] <Target>
1667 [12]   <Subjects>
1668 [13]     <Subject>
1669 [14]       <!-- any subject -->
1670 [15]       <AnySubject/>
1671 [16]     </Subject>
1672 [17]   </Subjects>
1673 [18]   <Resources>
1674 [19]     <Resource>
1675 [20]       <!-- any resource in the target namespace -->
1676 [21]       <ResourceMatch
1677 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
```

```

1678 [22]         <ResourceAttributeDesignator AttributeId=
1679 [23]         "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1680 [23]         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1681 [24]         <AttributeValue
1682 [24]         DataType="http://www.w3.org/2001/XMLSchema#string">
1683 [25]             http://www.medico.com/records.xsd
1684 [26]         </AttributeValue>
1685 [27]     </ResourceMatch>
1686 [28] </Resource>
1687 [29] </Resources>
1688 [30] <Actions>
1689 [31]     <Action>
1690 [32]         <!-- any action -->
1691 [33]         <AnyAction/>
1692 [34]     </Action>
1693 [35] </Actions>
1694 [36] </Target>
1695 [37] <!-- include policy from the example 3 by reference -->
1696 [38] <PolicyIdReference>
1697 [39]     urn:oasis:names:tc:xacml:1.0:examples:policyid:3
1698 [40] </PolicyIdReference>
1699 [41]     <!-- policy 2 combines rules from the examples 1, 2,
1700 [42]     and 4 is included by value. -->
1701 [43] <Policy
1702 [44]     PolicyId="urn:oasis:names:tc:xacml:examples:policyid:2"
1703 [45]     RuleCombiningAlgId=
1704 [46] "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
1705 [47]     <Description>
1706 [48]         Policy for any medical record in the
1707 [49]         http://www.medico.com/schemas/record.xsd namespace
1708 [50]     </Description>
1709 [51]     <Target> ... </Target>
1710 [52]     <Rule
1711 [53]         RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"
1712 [54]         Effect="Permit"> ... </Rule>
1713 [55]     <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
1714 [56]         Effect="Permit"> ... </Rule>
1715 [57]     <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:4"
1716 [58]         Effect="Deny"> ... </Rule>
1717 [59]     <Obligations> ... </Obligations>
1718 [60] </Policy>
1719 [61] </PolicySet>

```

1720

1721 [02]-[07] PolicySet declaration. Standard XML namespace declarations are included as well as
1722 PolicySetId, and **policy combining algorithm** identifier.

1723 [05]-[06] PolicySetId is used for identifying this **policy set** and for possible inclusion of this
1724 **policy set** into another **policy set**.

1725 [07] **Policy combining algorithm** identifier. Policies in the **policy set** are combined according to
1726 the specified **policy combining algorithm** identifier when the **authorization decision** is
1727 computed.

1728 [08]-[10] Free form description of the **policy set**.

1729 [11]-[36] PolicySet Target element defines a set of **decision requests** that are applicable to
1730 this PolicySet.

1731 [38]-[40] PolicyIdReference includes **policy** by id.

1732 [43]-[60] **Policy 2** is explicitly included in this **policy set**.

1733
1734

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

1735

5.1. Element <PolicySet>

1736
1737
1738
1739
1740
1741

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

1742
1743

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

1744
1745

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

1746
1747
1748

The <Target> element defines the applicability of the <PolicySet> to *decision requests*. If there is a match between the <Target> element within <PolicySet> and the *request context*, then the <PolicySet> element MAY be used by the *PDP* in making its *authorization decision*.

1749
1750
1751

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

1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770

```
<xs:element name="PolicySet" type="xacml:PolicySetType"/>
<xs:complexType name="PolicySetType">
  <xs:sequence>
    <xs:element ref="xacml:Description" minOccurs="0"/>
    <xs:element ref="xacml:PolicySetDefaults" minOccurs="0"/>
    <xs:element ref="xacml:Target"/>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="xacml:PolicySet"/>
      <xs:element ref="xacml:Policy"/>
      <xs:element ref="xacml:PolicySetIdReference"/>
      <xs:element ref="xacml:PolicyIdReference"/>
    </xs:choice>
    <xs:element ref="xacml:Obligations" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="PolicySetId"
type="http://www.w3.org/2001/XMLSchema#anyURI" use="required"/>
  <xs:attribute name="PolicyCombiningAlgId"
type="http://www.w3.org/2001/XMLSchema#anyURI" use="required"/>
</xs:complexType>
```

1771

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

1772

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

1773

PolicySetId [Required]

1774
1775
1776
1777

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

1778 PolicyCombiningAlgId [Required]

1779 The identifier of the **policy-combining algorithm** by which the <PolicySet>
 1780 components MUST be combined. Standard **policy-combining algorithms** are listed in
 1781 Appendix C. Standard **policy-combining algorithm** identifiers are listed in Section B.10.

1782 <Description> [Optional]

1783 A free-form description of the <PolicySet>.

1784 <PolicySetDefaults> [Optional]

1785 A set of default values applicable to the <PolicySet>. The scope of the
 1786 <PolicyDefaults> element SHALL be the enclosing policy.

1787 <Target> [Required]

1788 The <Target> element defines the applicability of a <PolicySet> to **decision requests**.

1789 The <Target> element MAY be declared by the creator of the <PolicySet> or it MAY be
 1790 computed from the <Target> elements of the referenced <Policy> elements, either as an
 1791 intersection or as a union.

1792 <PolicySet> [Any Number]

1793 A **policy set** component that is included in this **policy set**.

1794 <Policy> [Any Number]

1795 A **policy** component that is included in this **policy set**.

1796 <PolicySetIdReference> [Any Number]

1797 A reference to a <PolicySet> component that MUST be included in this **policy set**. If
 1798 <PolicySetIdReference> is a URL, then it MAY be resolvable.

1799 <PolicyIdReference> [Any Number]

1800 A reference to a <Policy> component that MUST be included in this **policy set**. If the
 1801 <PolicyIdReference> is a URL, then it MAY be resolvable.

1802 <Obligations> [Optional]

1803 Contains the set of <Obligation> elements. See Section 7.11 for a description of how
 1804 the set of **obligations** to be returned by the **PDP** shall be determined.

1805 5.2. Element <Description>

1806 The <Description> element is used for a free-form description of the <PolicySet> element
 1807 and <Policy> element. The <Description> element is of **xs:string** simple type.

1808

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

1809 5.3. Element <PolicySetDefaults>

1810 The <PolicySetDefaults> element SHALL specify default values that apply to the
 1811 <PolicySet> element.

1812

```
<xs:element name="PolicySetDefaults" type="xacml:DefaultsType"/>
```

```

1813 <xs:complexType name="DefaultsType">
1814   <xs:sequence>
1815     <xs:choice>
1816       <xs:element ref="xacml:XPathVersion" minOccurs="0" />
1817     </xs:choice>
1818   </xs:sequence>
1819 </xs:complexType>

```

1820 <PolicySetDefaults> element is of **DefaultsType** complex type.

1821 <XPathVersion> [Optional]

1822 Default XPath version.

1823 **5.4. Element <XPathVersion>**

1824 The <XPathVersion> element SHALL specify the version of the XPath specification to be used by
1825 <AttributeSelector> elements.

```

1826 <xs:element name="XPathVersion" type="xs:anyURI" />

```

1827 The URI for the XPath 1.0 specification is "<http://www.w3.org/TR/1999/Rec-xpath-19991116>". The <XPathVersion> element is REQUIRED if the XACML policy contains
1828 <AttributeSelector> elements.
1829

1830 **5.5. Element <Target>**

1831 The <Target> element identifies the set of **decision requests** that the parent element is intended
1832 to evaluate. The <Target> element SHALL appear as a child of <PolicySet>, <Policy> and
1833 <Rule> elements. It contains definitions for **subjects**, **resources** and **actions**.

1834 The <Target> element SHALL contain a **conjunctive sequence** of <Subjects>, <Resources>
1835 and <Actions> elements. For the parent of the <Target> element to be applicable to the
1836 **decision request**, there MUST be at least one positive match between each section of the
1837 <Target> element and the corresponding section of the <xacml-context:Request> element.

```

1838 <xs:element name="Target" type="xacml:TargetType" />
1839 <xs:complexType name="TargetType">
1840   <xs:sequence>
1841     <xs:element ref="xacml:Subjects" />
1842     <xs:element ref="xacml:Resources" />
1843     <xs:element ref="xacml:Actions" />
1844   </xs:sequence>
1845 </xs:complexType>

```

1846 The <Target> element is of **TargetType** complex type.

1847 <Subjects> [Required]

1848 Matching specification for the **subject attributes** in the **context**.

1849 <Resources> [Required]

1850 Matching specification for the **resource attributes** in the **context**.

1851 <Actions> [Required]

1852 Matching specification for the **action attributes** in the **context**.

1853 5.6. Element <Subjects>

1854 The <Subjects> element SHALL contain a *disjunctive sequence* of <Subject> elements.

```
1855 <xs:element name="Subjects" type="xacml:SubjectsType" />
1856 <xs:complexType name="SubjectsType">
1857   <xs:choice>
1858     <xs:element ref="xacml:Subject" maxOccurs="unbounded" />
1859     <xs:element ref="xacml:AnySubject" />
1860   </xs:choice>
1861 </xs:complexType>
```

1862 The <Subjects> element is of **SubjectsType** complex type.

1863 <Subject> [One To Many, Required Choice]

1864 See section 5.7.

1865 <AnySubject> [Required Choice]

1866 See section 5.8.

1867 5.7. Element <Subject>

1868 The <Subject> element SHALL contain a *conjunctive sequence* of <SubjectMatch> elements.

```
1870 <xs:element name="Subject" type="xacml:SubjectType" />
1871 <xs:complexType name="SubjectType">
1872   <xs:sequence>
1873     <xs:element ref="xacml:SubjectMatch" maxOccurs="unbounded" />
1874   </xs:sequence>
1875 </xs:complexType>
```

1876 The <Subject> element is of **SubjectType** complex type.

1877 <Subject> element contains the following elements:

1878 <SubjectMatch> [One to Many]

1879 A *conjunctive sequence* of individual matches of the *subject attributes* in the *context*
1880 and the embedded *attribute* values.

1881 5.8. Element <AnySubject>

1882 The <AnySubject> element SHALL match any *subject attribute* in the *context*.

```
1883 <xs:element name="AnySubject" />
```

1884 5.9. Element <SubjectMatch>

1885 The <SubjectMatch> element SHALL identify a set of *subject*-related entities by matching
1886 *attribute* values in the <xacml-context:Subject> element of the *context* with the embedded
1887 *attribute* value.

```
1888 <xs:element name="SubjectMatch" type="xacml:SubjectMatchType" />
1889 <xs:complexType name="SubjectMatchType">
1890   <xs:sequence>
1891     <xs:choice>
```



```

1892     <xs:element ref="xacml:SubjectAttributeDesignator" />
1893     <xs:element ref="xacml:AttributeSelector" />
1894   </xs:choice>
1895   <xs:element ref="xacml:AttributeValue" />
1896 </xs:sequence>
1897 <xs:attribute name="MatchId" type="xs:anyURI" use="required" />
1898 </xs:complexType>

```

1899 The <SubjectMatch> element is of **SubjectMatchType** complex type.

1900 The <SubjectMatch> element contains the following attributes and elements:

1901 MatchId [Required]

1902 Specifies a matching function. The value of this attribute MUST be of type xs:anyURI with
1903 legal values documented in Appendix A.

1904 <SubjectAttributeDesignator> [Required choice]

1905 Identifies one or more **attribute** values in the <xacml-context:Subject> child of the
1906 <xacml-context:Request> element.

1907 <AttributeSelector> [Required choice]

1908 MAY be used to identify one or more **attribute** values in the <xacml-context:Subject>
1909 child of the <xacml-context:Request> element.

1910 <AttributeValue> [Required]

1911 Embedded **attribute** value.

1912 **5.10. Element <Resources>**

1913 The <Resources> element SHALL contain a **disjunctive sequence** of <Resource> elements.

```

1914 <xs:element name="Resources" type="xacml:ResourcesType" />
1915 <xs:complexType name="ResourcesType">
1916   <xs:choice>
1917     <xs:element ref="xacml:Resource" maxOccurs="unbounded" />
1918     <xs:element ref="xacml:AnyResource" />
1919   </xs:choice>
1920 </xs:complexType>

```

1921 The <Resources> element is of **ResourcesType** complex type.

1922 The <Resources> element consists of the following elements:

1923 <Resource> [One To Many, Required Choice]

1924 See section 5.11.

1925 <AnyResource> [Required Choice]

1926 See section 5.12.

1927 **5.11. Element <Resource>**

1928 The <Resource> element SHALL contain a **conjunctive sequence** of <ResourceMatch>
1929 elements.

```

1930 <xs:element name="Resource" type="xacml:ResourceType" />
1931 <xs:complexType name="ResourceType" >
1932   <xs:sequence>
1933     <xs:element ref="xacml:ResourceMatch" maxOccurs="unbounded" />
1934   </xs:sequence>
1935 </xs:complexType>

```

1936 The <Resource> element is of **ResourceType** complex type.

1937 The <Resource> element contains the following elements:

1938 <ResourceMatch> [One to Many]

1939 A **conjunctive sequence** of individual matches of the **resource attributes** in the **context**
1940 and the embedded **attribute** values.

1941 5.12. Element <AnyResource>

1942 The <AnyResource> element SHALL match any **resource attribute** in the **context**.

```

1943 <xs:element name="AnyResource" />

```

1944 5.13. Element <ResourceMatch>

1945 The <ResourceMatch> element SHALL identify a set of **resource**-related entities by matching
1946 **attribute** values in the <xacml-context:Resource> element of the **context** with the embedded
1947 **attribute** value.

```

1948 <xs:element name="ResourceMatch" type="xacml:ResourceMatchType" />
1949 <xs:complexType name="ResourceMatchType" >
1950   <xs:sequence>
1951     <xs:choice>
1952       <xs:element ref="xacml:ResourceAttributeDesignator" />
1953       <xs:element ref="xacml:AttributeSelector" />
1954     </xs:choice>
1955     <xs:element ref="xacml:AttributeValue" />
1956   </xs:sequence>
1957   <xs:attribute name="MatchId" type="xs:anyMatch" use="required" />
1958 </xs:complexType>

```

1959 The <ResourceMatch> element is of **ResourceMatchType** complex type.

1960 The <ResourceMatch> element contains the following attributes and elements:

1961 MatchId [Required]

1962 Specifies a matching function. Values of this attribute MUST be of type xs:anyURI, with
1963 legal values documented in Appendix A.

1964 <ResourceAttributeDesignator> [Required Choice]

1965 Identifies one or more **attribute** values in the <xacml-context:Resource> child of the
1966 <xacml-context:Request> element.

1967 <AttributeSelector> [Required Choice]

1968 MAY be used to identify one or more **attribute** values in the <xacml-
1969 context:Resource> child of the <xacml-context:Request> element.

1970 <AttributeValue> [Required]

1971 Embedded *attribute* value.

1972 **5.14. Element <Actions>**

1973 The <Actions> element SHALL contain a *disjunctive sequence* of <Action> elements.

```
1974 <xs:element name="Actions" type="xacml:ActionTypes"/>
1975 <xs:complexType name="ActionTypes">
1976   <xs:choice>
1977     <xs:element ref="xacml:Action" maxOccurs="unbounded"/>
1978     <xs:element ref="xacml:AnyAction"/>
1979   </xs:choice>
1980 </xs:complexType>
```

1981 The <Actions> element is of **ActionTypes** complex type.

1982 The <Actions> element contains the following elements:

1983 <Action> [One To Many, Required Choice]

1984 See section 5.15.

1985 <AnyAction> [Required Choice]

1986 See section 5.16.

1987 **5.15. Element <Action>**

1988 The <Action> element SHALL contain a *conjunctive sequence* of <ActionMatch> elements.

```
1989 <xs:element name="Action" type="xacml:ActionType"/>
1990 <xs:complexType name="ActionType">
1991   <xs:sequence>
1992     <xs:element ref="xacml:ActionMatch" maxOccurs="unbounded"/>
1993   </xs:sequence>
1994 </xs:complexType>
```

1995 The <Action> element is of **ActionType** complex type.

1996 The <Action> element contains the following elements:

1997 <ActionMatch> [One to Many]

1998 A *conjunctive sequence* of individual matches of the *action* attributes in the *context* and
1999 the embedded *attribute* values.

2000 **5.16. Element <AnyAction>**

2001 The <AnyAction> element SHALL match any *action attribute* in the *context*.

```
2002 <xs:element name="AnyAction"/>
```

2003

2004

5.17. Element <ActionMatch>

2005 The <ActionMatch> element SHALL identify a set of **action**-related entities by matching **attribute**
2006 values in the <xacml-context:Action> element of the **context** with the embedded **attribute**
2007 value.

```
2008 <xs:element name="ActionMatch" type="xacml:ActionMatchType"/>  
2009 <xs:complexType name="ActionMatchType">  
2010   <xs:sequence>  
2011     <xs:choice>  
2012       <xs:element ref="xacml:ActionAttributeDesignator"/>  
2013       <xs:element ref="xacml:AttributeSelector"/>  
2014     </xs:choice>  
2015     <xs:element ref="xacml:AttributeValue"/>  
2016   </xs:sequence>  
2017   <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>  
2018 </xs:complexType>
```

2019 The <ActionMatch> element is of **ActionMatchType** complex type.

2020 The <ActionMatch> element contains the following attributes and elements:

2021 MatchId [Required]

2022 Specifies a matching function. The value of this attribute MUST be of type xs:anyURI, with
2023 legal values documented in Appendix A.

2024 <ActionAttributeDesignator> [Required Choice]

2025 Identifies one or more **attribute** values in the <xacml-context:Action> child of the
2026 <xacml-context:Request> element.

2027 <AttributeSelector> [Required Choice]

2028 MAY be used to identify one or more **attribute** values in the <xacml-context:Action>
2029 child of the <xacml-context:Request> element.

2030 <AttributeValue> [Required]

2031 Embedded **attribute** value.

2032 5.18. Element <PolicySetIdReference>

2033 The <PolicySetIdReference> element SHALL be used to reference a <PolicySet> element
2034 by id. If <PolicySetIdReference> is a URL, then it MAY be resolvable to the <PolicySet>.
2035 The mechanism for resolving a **policy set** reference to the corresponding **policy set** is
2036 implementation dependent.

```
2037 <xs:element name="PolicySetIdReference" type="xs:anyURI"/>
```

2038 Element <PolicySetIdReference> is of **xs:anyURI** simple type.

2039 5.19. Element <PolicyIdReference>

2040 The <xacml:PolicyIdReference> element SHALL be used to reference a <Policy> element
2041 by id. If <PolicyIdReference> is a URL, then it MAY be resolvable to the <Policy>. The
2042 mechanism for resolving a **policy** reference to the corresponding **policy** is implementation
2043 dependent.

```
2044 <xs:element name="PolicyIdReference" type="xs:anyURI"/>
```

2045 Element <PolicyIdReference> is of **xs:anyURI** simple type.

2046 **5.20. Element <Policy>**

2047 The <Policy> element is the smallest entity that SHALL be presented to the **PDP** for evaluation.

2048 The main components of this element are the <Target>, <Rule> and <Obligations> elements
2049 and the RuleCombiningAlgId attribute.

2050 The <Target> element SHALL define <Policy> applicability to **decision requests**. A sequence
2051 of <Rule> elements SHALL specify authorizations that MUST be combined according to the
2052 RuleCombiningAlgId attribute. The <Obligations> element SHALL contain a set of
2053 **obligations** that MUST be discharged by the **PDP** in conjunction with the **authorization decision**.

```
2054 <xs:element name="Policy" type="xacml:PolicyType"/>  
2055 <xs:complexType name="PolicyType">  
2056   <xs:sequence>  
2057     <xs:element ref="xacml:Description" minOccurs="0"/>  
2058     <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>  
2059     <xs:element ref="xacml:Target"/>  
2060     <xs:element ref="xacml:Rule" minOccurs="0" maxOccurs="unbounded"/>  
2061     <xs:element ref="xacml:Obligations" minOccurs="0"/>  
2062   </xs:sequence>  
2063   <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>  
2064   <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>  
2065 </xs:complexType>
```

2066 The <Policy> element is of **PolicyType** complex type.

2067 The <Policy> element contains the following attributes and elements:

2068 **PolicyId** [Required]

2069 Policy identifier. The party assigning this identifier MUST minimize the potential of some
2070 other party reusing the same identifier. This MAY be achieved by following a predefined
2071 URN or URL scheme. It is OPTIONAL for the PolicyId URL to be resolvable to the
2072 corresponding <Policy> object.

2073 **RuleCombiningAlgId** [Required]

2074 The identifier of the rule-combining algorithm by which the <Policy> components MUST be
2075 combined. Standard rule-combining algorithms are listed in Appendix C. Standard rule-
2076 combining algorithm identifiers are listed in Section B.10.

2077 <Description> [Optional]

2078 A free-form description of the **policy**.

2079 <PolicyDefaults> [Optional]

2080 Defines a set of default values applicable to the **policy**. The scope of the
2081 <PolicyDefaults> element SHALL be the enclosing policy.

2082 <Target> [Required]

2083 The <Target> element SHALL define the applicability of a <Policy> to **decision requests**.

2084 The <Target> element MAY be declared by the creator of the <Policy> element, or it
2085 MAY be computed from the <Target> elements of the referenced <Rule> elements either
2086 as an intersection or as a union.

2087 <Rule> [Any Number]

2088 A sequence of authorizations that MUST be combined according to the
2089 RuleCombiningAlgId attribute. **Rules** whose <Target> elements match the **decision**
2090 **request** MUST be considered. **Rules** whose <Target> elements do not match the
2091 **decision request** MUST NOT be considered. Applicability of **rules** to the **decision**
2092 **request** is detailed in Appendix C.

2093 <Obligations> [Optional]

2094 A **conjunctive sequence** of **obligations** that MUST be discharged by the **PEP** in
2095 conjunction with the **authorization decision**. See Section 7.11 for a description of how the
2096 set of obligations to be returned by the **PDP** shall be determined.

2097 5.21. Element <Rule>

2098 The <Rule> element SHALL define individual **rules** in the **policy**. The main components of this
2099 element are the <Target> and <Condition> elements and the Effect attribute.

```
2100 <xs:element name="Rule" type="xacml:RuleType"/>  
2101 <xs:complexType name="RuleType">  
2102   <xs:sequence>  
2103     <xs:element ref="xacml:Description" minOccurs="0"/>  
2104     <xs:element ref="xacml:Target" minOccurs="0"/>  
2105     <xs:element ref="xacml:Condition" minOccurs="0"/>  
2106   </xs:sequence>  
2107   <xs:attribute name="RuleId" type="xs:anyURI" use="required"/>  
2108   <xs:attribute name="Effect" type="xacml:EffectType" use="required"/>  
2109 </xs:complexType>
```

2110 The <Rule> element is of **RuleType** complex type.

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

2112 RuleId [Required]

2113 A URN identifying this **rule**.

2114 Effect [Required]

2115 **Rule effect.** Values of this attribute are either "Permit" or "Deny".

2116 <Description> [optional]

2117 A free-form description of the **rule**.

2118 <Target> [optional]

2119 Identifies the set of **decision requests** that the <Rule> element is intended to evaluate. If
2120 this element is omitted, then the **target** for the <Rule> SHALL be defined by the enclosing
2121 <Policy> element. See Section 5.5 for details.

2122 <Condition> [optional]

2123 A *predicate* that MUST be satisfied for the *rule* to be assigned its `Effect` value. A
2124 *condition* is a boolean function over a combination of *subject*, *resource*, *action* and
2125 *environment attributes* or other functions.

2126 5.22. Simple type `EffectType`

2127 The `EffectType` simple type defines the values allowed for the `Effect` attribute of the `<Rule>`
2128 element and for the `FulfillOn` attribute of the `<Obligation>` element.

```
2129 <xs:simpleType name="EffectType">  
2130 <xs:restriction base="xs:string">  
2131 <xs:enumeration value="Permit" />  
2132 <xs:enumeration value="Deny" />  
2133 </xs:restriction>  
2134 </xs:simpleType>
```

2135 5.23. Element `<Condition>`

2136 The `<Condition>` element is a boolean function over *subject*, *resource*, *action* and
2137 *environment attributes* or functions of *attributes*. If the `<Condition>` element evaluates to
2138 "True", then the enclosing `<Rule>` element is assigned its `Effect` value.

```
2139 <xs:element name="Condition" type="xacml:ApplyType" />
```

2140 The `<Condition>` element is of `ApplyType` complex type.

2141 5.24. Element `<Apply>`

2142 The `<Apply>` element denotes application of a function to its arguments, thus encoding a function
2143 call. The `<Apply>` element can be applied to any combination of `<Apply>`,
2144 `<AttributeValue>`, `<SubjectAttributeDesignator>`,
2145 `<ResourceAttributeDesignator>`, `<ActionAttributeDesignator>`,
2146 `<EnvironmentAttributeDesignator>` and `<AttributeSelector>` arguments.

```
2147 <xs:element name="Apply" type="xacml:ApplyType" />  
2148 <xs:complexType name="ApplyType">  
2149 <xs:choice minOccurs="0" maxOccurs="unbounded">  
2150 <xs:element ref="xacml:Function" />  
2151 <xs:element ref="xacml:Apply" />  
2152 <xs:element ref="xacml:AttributeValue" />  
2153 <xs:element ref="xacml:SubjectAttributeDesignator" />  
2154 <xs:element ref="xacml:ResourceAttributeDesignator" />  
2155 <xs:element ref="xacml:ActionAttributeDesignator" />  
2156 <xs:element ref="xacml:EnvironmentAttributeDesignator" />  
2157 <xs:element ref="xacml:SubjectAttributeIsPresent" />  
2158 <xs:element ref="xacml:ResourceAttributeIsPresent" />  
2159 <xs:element ref="xacml:ActionAttributeIsPresent" />  
2160 <xs:element ref="xacml:EnvironmentAttributeIsPresent" />  
2161 <xs:element ref="xacml:AttributeSelector" />  
2162 </xs:choice>  
2163 <xs:attribute name="FunctionId" type="xs:anyURI" use="required" />  
2164 </xs:complexType>
```

2165 The `<Apply>` element is of `ApplyType` complex type.

2166 The `<Apply>` element contains the following attributes and elements:

2167 `FunctionId` [Required]

2168 The URN of a function. XACML-defined functions are described in Appendix A.

2169 <Function> [Optional]

2170 The name of a function that is applied to the elements of a **bag**. See section A14.11.

2171 <Apply> [Optional]

2172 A nested function-call argument.

2173 <AttributeValue> [Optional]

2174 A literal value argument.

2175 <ResourceAttributeDesignator> [Optional]

2176 A **resource attribute** argument.

2177 <ActionAttributeDesignator> [Optional]

2178 An **action attribute** argument.

2179 <EnvironmentAttributeDesignator> [Optional]

2180 An **environment attribute** argument.

2181 <SubjectAttributeIsPresent> [Optional]

2182 An argument that tests presence of the **subject attribute**

2183 <ResourceAttributeIsPresent> [Optional]

2184 An argument that tests presence of the **resource attribute**

2185 <ActionAttributeIsPresent> [Optional]

2186 An argument that tests presence of the **action attribute**

2187 <EnvironmentAttributeIsPresent> [Optional]

2188 An argument that tests presence of the **environment attribute**

2189 <AttributeSelector> [Optional]

2190 An **attribute** selector argument.

2191 **5.25. Element <Function>**

2192 The `Function` element SHALL be used to name a function that is applied by the higher-order **bag**
 2193 functions to every element of a **bag**. The higher-order **bag** functions are described in Section
 2194 A14.11.

```
2195 <xs:element name="Function" type="xacml:FunctionType" />
2196 <xs:complexType name="FunctionType">
2197   <xs:attribute name="FunctionId" type="xs:QName" use="required" />
2198 </xs:complexType>
```

2199 The `Function` element is of **FunctionType** complex type.

2200 The `Function` element contains the following attributes:

2201 FunctionId [Required]

2202 The identifier for the function that is applied to the elements of a **bag** by the higher-order
2203 **bag** functions.

2204 5.26. Complex type AttributeDesignatorType

2205 The **AttributeDesignatorType** complex type is the type for elements and extensions that refer to
2206 named **attributes**. A named **attribute** has specific criteria with which to match **attributes** within a
2207 specific part of the <xacml-context:Request> element. The **AttributeDesignatorType**
2208 complex type specifies the attributes used for the match criteria that are common to all named
2209 **attributes**. Elements and extensions of the **AttributeDesignatorType** complex type MAY
2210 determine the presence of named **attributes** or select **attribute values** associated with **attributes**
2211 that match named **attributes**. Elements and extensions of the **AttributeDesignatorType** SHALL
2212 NOT alter the match semantics of named **attributes**, but MAY narrow the search space.

2213

```
2214 <xs:complexType name="AttributeDesignatorType">  
2215   <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2216   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>  
2217   <xs:attribute name="Issuer" type="xs:anyURI" use="optional"/>  
2218   <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"/>  
2219 </xs:complexType>
```

2220

2221 A named **attribute** SHALL match an **attribute** if the values of their respective `AttributeId`,
2222 `DataType` and `Issuer` attributes match. The `AttributeId` attribute MUST match, by URI
2223 equality, that of the `AttributeId` attribute of the **attribute**. The `DataType` attribute MUST match,
2224 by URI equality, that of the `DataType` attribute of the same **attribute**. If the `Issuer` attribute is
2225 supplied, it MUST match, by URI equality, the `Issuer` attribute of the same **attribute**. If the
2226 `Issuer` attribute is not supplied, the matching of the **attribute** to the **named attribute** SHALL be
2227 governed by `AttributeId` and `DataType` attributes alone, regardless of the presence, absence,
2228 or actual value of the `Issuer` attribute.

2229 The <AttributeDesignatorType> contains the following attributes:

2230 `AttributeId` [Required]

2231 This attribute SHALL specify the `AttributeId` with which to match the **attribute**.

2232 `DataType` [Required]

2233 This attribute SHALL specify the data-type with which to match the **attribute**.

2234 `Issuer` [Optional]

2235 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **attribute**.

2236 `MustBePresent` [Optional]

2237 This attribute governs whether the element returns "Indeterminate" in the case of the
2238 absence of the named **attribute**. If the **named attribute** is absent and `MustBePresent` is
2239 set to "True", then this element SHALL result in "Indeterminate". If `MustBePresent` is not
2240 supplied, its default value SHALL be `false`.

2241 5.27. Element <ResourceAttributeDesignator>

2242

2243 The <ResourceAttributeDesignator> element retrieves a **bag** of values for a *named*
2244 *resource attribute*. A **resource attribute** is an **attribute** that SHALL only be located within the
2245 <Resource> element of the <xacml-context:Request> element. A *named resource attribute*
2246 is a *named attribute* that matches a **resource attribute**. A *named resource attribute* SHALL be
2247 considered *present* if there is at least one **resource attribute** that matches the criteria set out
2248 below. A **resource attribute** value is an **attribute** value that is contained within a **resource**
2249 **attribute**.

2250 The <ResourceAttributeDesignator> element SHALL return a **bag** of all the **resource**
2251 **attribute** values that are matched by the *named resource attribute*. The `MustBePresent` attribute
2252 governs whether this element returns an empty **bag** or “Indeterminate” in the case of the absence
2253 of the *named resource attribute*. If the *named resource attribute* is not present and the
2254 `MustBePresent` attribute is set to “False” (its default value) this element SHALL result in an empty
2255 **bag**. If the *named resource attribute* is not present and the `MustBePresent` attribute is set to
2256 “True”, this element SHALL result in “Indeterminate”. Regardless of the `MustBePresent` attribute,
2257 if it cannot be determined whether the *named resource attribute* is present or not in the **request**
2258 **context**, or the value of the *named resource attribute* is unavailable, then the expression SHALL
2259 evaluate to “Indeterminate”.

2260 A *named resource attribute* SHALL match a **resource attribute** as per the match semantics
2261 specified in the **AttributeDesignatorType** complex type [Section 5.26]

2262 The <ResourceAttributeDesignator> MAY appear in the <ResourceMatch> element and
2263 MAY be passed to the <Apply> element as an argument.

2264

```
2265 <xs:element name="ResourceAttributeDesignator"  
2266           type="xacml:AttributeDesignatorType"/>
```

2267

2268 The <ResourceAttributeDesignator> element is of the **AttributeDesignatorType** complex
2269 type.

2270 The <ResourceAttributeDesignator> element has the following attributes:

2271 `AttributeId` [Required]

2272 This attribute SHALL specify the `AttributeId` with which to match the **resource**
2273 **attribute**.

2274 `DataType` [Required]

2275 This attribute SHALL specify the `DataType` with which to match the **resource attribute**.

2276 `Issuer` [Optional]

2277 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **resource**
2278 **attribute**.

2279 `MustBePresent` [Optional]

2280 This attribute governs whether the <ResourceAttributeDesignator> element returns
2281 an empty **bag** or “Indeterminate” in the case of the absence of the *named resource*
2282 *attribute*. If the *named resource attribute* is absent and `MustBePresent` is set to “False”,
2283 then this element SHALL result in an empty **bag**. If the *named resource attribute* is absent
2284 and `MustBePresent` is set to “True”, then this element SHALL evaluate to

2285 "Indeterminate". If `MustBePresent` is not supplied, then its default value SHALL be
2286 "False".

2287 5.28. Element `<ActionAttributeDesignator>`

2288

2289 The `<ActionAttributeDesignator>` element retrieves a bag of values for a **named action**
2290 **attribute**. An **action attribute** is an **attribute** that SHALL only be located within the `<Action>`
2291 element of the `<xacml-context:Request>` element. A **named action attribute** has specific
2292 criteria (described below) with which to match an **action attribute**. A **named action attribute**
2293 SHALL be considered *present*, i.e. *not absent*, if there is at least one **action attribute** that matches
2294 the criteria. A **action attribute value** is an **attribute value** that is contained within a **action**
2295 **attribute**.

2296 The `<ActionAttributeDesignator>` element SHALL return a bag of all the **action attribute**
2297 **values** that are matched by the **named action attribute**. The `MustBePresent` attribute governs
2298 whether this element returns an empty bag or **indeterminate** in the case of the absence of the
2299 **named action attribute**. If the **named action attribute** is not present and the `MustBePresent`
2300 attribute is set to `false` (its default value) this element SHALL result in an empty bag. If the **named**
2301 **action attribute** is not present and the `MustBePresent` attribute is set to `true`, this element
2302 SHALL result in **indeterminate**. Regardless of the `MustBePresent` attribute, if it cannot be
2303 determined whether the **named action attribute** is present or not present in the request context, or
2304 the value of the **named action attribute** is unavailable, then the expression SHALL result in
2305 **indeterminate**.

2306 A **named action attribute** SHALL match a **action attribute** as per the match semantics specified
2307 in the **AttributeDesignatorType** complex type [Section 5.26].

2308 The `<ActionAttributeDesignator>` MAY appear in the `<ActionMatch>` element and MAY
2309 be passed to the `<Apply>` element as an argument.

2310

```
2311 <xs:element name="ActionAttributeDesignator"  
2312           type="xacml:AttributeDesignatorType"/>
```

2313

2314 The `<ActionAttributeDesignator>` element is of the **AttributeDesignatorType** complex
2315 type.

2316 The `<ActionAttributeDesignator>` element has the following attributes:

2317 `AttributeId` [Required]

2318 This attribute SHALL specify the `AttributeId` with which to match the **action attribute**.

2319 `DataType` [Required]

2320 This attribute SHALL specify the `DataType` with which to match the **action attribute**.

2321 `Issuer` [Optional]

2322 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **action**
2323 **attribute**.

2324 `MustBePresent` [Optional]

2325 This attribute governs the whether the <ActionAttributeDesignator> element returns
2326 an empty bag or *indeterminate* in the case of the absence of the **named action attribute**.
2327 If the **named action attribute** is absent and `MustBePresent` is set to `false`, this element
2328 SHALL result in an empty bag. If the **named action attribute** is absent and
2329 `MustBePresent` is set to `true`, this element SHALL result in *indeterminate*. If
2330 `MustBePresent` is not supplied, its default value SHALL be `false`.

2331 5.29. Element <EnvironmentAttributeDesignator>

2332

2333 The <EnvironmentAttributeDesignator> element retrieves a bag of values for a **named**
2334 **environment attribute**. A **environment attribute** is an **attribute** that SHALL only be located within
2335 the <Environment> element of the <xacml-context:Request> element. A **named**
2336 **environment attribute** has specific criteria (described below) with which to match a **environment**
2337 **attribute**. A **named environment attribute** SHALL be considered *present*, i.e. *not absent*, if there
2338 is at least one **environment attribute** that matches the criteria. A **environment attribute value** is
2339 an **attribute value** that is contained within a **environment attribute**.

2340 The <EnvironmentAttributeDesignator> element SHALL return a bag of all the
2341 **environment attribute values** that are matched by the **named environment attribute**. The
2342 `MustBePresent` attribute governs whether this element returns an empty bag or *indeterminate* in
2343 the case of the absence of the **named environment attribute**. If the **named environment**
2344 **attribute** is not present and the `MustBePresent` attribute is set to `false` (its default value) this
2345 element SHALL result in an empty bag. If the **named environment attribute** is not present and the
2346 `MustBePresent` attribute is set to `true`, this element SHALL result in *indeterminate*. Regardless
2347 of the `MustBePresent` attribute, if it cannot be determined whether the **named environment**
2348 **attribute** is present or not present in the request context, or the value of the **named environment**
2349 **attribute** is unavailable, then the expression SHALL result in *indeterminate*.

2350 A **named environment attribute** SHALL match a **environment attribute** as per the match
2351 semantics specified in the **AttributeDesignatorType** complex type [Section 5.26].

2352 The <EnvironmentAttributeDesignator> MAY be passed to the <Apply> element as an
2353 argument.

2354

```
2355 <xs:element name="EnvironmentAttributeDesignator"  
2356           type="xacml:AttributeDesignatorType"/>
```

2357

2358 The <EnvironmentAttributeDesignator> element is of the **AttributeDesignatorType**
2359 complex type.

2360 The <EnvironmentAttributeDesignator> element has the following attributes:

2361 `AttributeId` [Required]

2362 This attribute SHALL specify the `AttributeId` with which to match the **environment**
2363 **attribute**.

2364 `DataType` [Required]

2365 This attribute SHALL specify the `DataType` with which to match the **environment attribute**.

2366 `Issuer` [Optional]

2367 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **environment**
2368 **attribute**.

2369 `MustBePresent` [Optional]

2370 This attribute governs the whether the `<EnvironmentAttributeDesignator>` element returns
2371 an empty bag or **indeterminate** in the case of the absence of the **named environment attribute**. If
2372 the **named environment attribute** is absent and `MustBePresent` is set to `false`, this element
2373 SHALL result in an empty bag. If the **named environment attribute** is absent and
2374 `MustBePresent` is set to `true`, this element SHALL result in **indeterminate**. If `MustBePresent`
2375 is not supplied, its default value SHALL be `false`.

2376 5.30. Element `<ResourceAttributeIsPresent>`

2377

2378 The `<ResourceAttributeIsPresent>` element determines whether a **named resource**
2379 **attribute** is present. A **resource attribute** is an **attribute** that SHALL only be located within the
2380 `<Resource>` element of the `<xacml-context:Request>` element. A **named resource**
2381 **attribute** is a **named attribute** that matches a **resource attribute**. A **named resource attribute**
2382 SHALL be considered *present*, i.e. *not absent*, if there is at least one **resource attribute** that
2383 matches the criteria described below.

2384 The `<ResourceAttributeIsPresent>` element SHALL result in **true** if its **named resource**
2385 **attribute** is present. A result of **true** SHALL mean that a `<ResourceAttributeDesignator>`
2386 element for this **named resource attribute** SHALL return a bag consisting of at least one **attribute**
2387 **value**. The `MustBePresent` attribute governs whether this element returns **false** or **indeterminate**
2388 in the case of the absence of the **named resource attribute**. If the **named resource attribute** is
2389 not present and the `MustBePresent` attribute is set to `false` (its default value) this element
2390 SHALL result in **false**. If the **named resource attribute** is not present and the `MustBePresent`
2391 attribute is set to `true`, this element SHALL result in **indeterminate**. Regardless of the
2392 `MustBePresent` attribute, if it cannot be determined whether the **named resource attribute** is
2393 present or not present in the request context, or the value of the **named resource attribute** is
2394 unavailable, then the expression SHALL result in **indeterminate**.

2395 A **named resource attribute** SHALL be considered present if at least one **resource attribute**
2396 exists that matches the values of its corresponding `AttributeId`, `DataType`, and `Issuer`
2397 attributes as per the match semantics of the **AttributeDesignatorType** specification [Section
2398 **Error! Reference source not found.**].

2399 The `<ResourceAttributeIsPresent>` MAY be passed to the `<Apply>` element as an
2400 argument.

2401

```
2402 <xs:element name="ResourceAttributeIsPresent"  
2403           type="xacml:AttributeDesignatorType"/>
```

2404

2405 The `<ResourceAttributeIsPresent>` element is of the **AttributeDesignatorType** complex
2406 type.

2407 The `<ResourceAttributeIsPresent>` element has the following attributes:

2408 `AttributeId` [Required]

2409 This attribute SHALL specify the `AttributeId` with which to match the **resource**
2410 **attribute**.

2411 `DataType` [Required]
 2412 This attribute SHALL specify the `DataType` with which to match the *resource attribute*.
 2413 `Issuer` [Optional]
 2414 This attribute, if supplied, SHALL specify the `Issuer` with which to match the *resource*
 2415 *attribute*.
 2416 `MustBePresent` [Optional]
 2417 This attribute governs the whether the `<ResourceAttributeIsPresent>` element returns *false*
 2418 or *indeterminate* in the case of the absence of the *named resource attribute*. If the
 2419 *named resource attribute* is absent and `MustBePresent` is set to *false*, this element
 2420 SHALL result in *false*. If the *named resource attribute* is absent and `MustBePresent` is
 2421 set to *true*, this element SHALL result in *indeterminate*. If `MustBePresent` is not
 2422 supplied, its default value SHALL be *false*.

2423 5.31. Element `<ActionAttributeIsPresent>`

2424
 2425 The `<ActionAttributeIsPresent>` element determines whether a *named action attribute* is
 2426 present. An *action attribute* is an *attribute* that SHALL only be located within the `<Action>`
 2427 element of the `<xacml-context:Request>` element. A *named action attribute* is a *named*
 2428 *attribute* that matches an *action attribute*. A *named action attribute* SHALL be considered
 2429 *present*, i.e. *not absent*, if there is at least one *action attribute* that matches the criteria below
 2430
 2431 The `<ActionAttributeIsPresent>` element SHALL result in *true* if its *named action attribute*
 2432 is present. A result of *true* SHALL mean that a `<ActionAttributeDesignator>` element for
 2433 this *named action attribute* SHALL return a bag consisting of at least one *attribute value*. The
 2434 `MustBePresent` attribute governs whether this element returns *false* or *indeterminate* in the case
 2435 of the absence of the *named action attribute*. If the *named action attribute* is not present and the
 2436 `MustBePresent` attribute is set to *false* (its default value) this element SHALL result in *false*. If
 2437 the *named action attribute* is not present and the `MustBePresent` attribute is set to *true*, this
 2438 element SHALL result in *indeterminate*. Regardless of the `MustBePresent` attribute, if it cannot
 2439 be determined whether the *named action attribute* is present or not present in the request context,
 2440 or the value of the *named action attribute* is unavailable, then the expression SHALL result in
 2441 *indeterminate*.
 2442 A *named action attribute* SHALL be considered present if at least one *action attribute* exists that
 2443 matches the values of its corresponding `AttributeId`, `DataType`, and `Issuer` attributes as per
 2444 the match semantics of the `AttributeDesignatorType` specification [Section 5.26].
 2445 The `<ActionAttributeIsPresent>` MAY be passed to the `<Apply>` element as an argument.

```
2446 <xs:element name="ActionAttributeIsPresent"  
2447           type="xacml:AttributeDesignatorType" />
```

2448
 2449 The `<ActionAttributeIsPresent>` element is of the `AttributeDesignatorType` complex type.
 2450 The `<ActionAttributeIsPresent>` element has the following attributes:
 2451 `AttributeId` [Required]

2452 This attribute SHALL specify the `AttributeId` with which to match the **action attribute**.

2453 `DataType` [Required]

2454 This attribute SHALL specify the `DataType` with which to match the **action attribute**.

2455 `Issuer` [Optional]

2456 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **action**
2457 **attribute**.

2458 `MustBePresent` [Optional]

2459 This attribute governs the whether the `<ActionAttributeIsPresent>` element returns **false** or
2460 **indeterminate** in the case of the absence of the **named action attribute**. If the **named**
2461 **action attribute** is absent and `MustBePresent` is set to **false**, this element SHALL
2462 result in **false**. If the **named action attribute** is absent and `MustBePresent` is set to
2463 **true**, this element SHALL result in **indeterminate**. If `MustBePresent` is not supplied, its
2464 default value SHALL be **false**.

2465 5.32. Element `<EnvironmentAttributeIsPresent>`

2466

2467 The `<EnvironmentAttributeIsPresent>` element determines whether a **named environment**
2468 **attribute** is present. An **environment attribute** is an **attribute** that SHALL only be located within
2469 the `<Environment>` element of the `<xacml-context:Request>` element. A **named**
2470 **environment attribute** is a **named attribute** that matches an **environment attribute**. A **named**
2471 **environment attribute** SHALL be considered *present*, i.e. *not absent*, if there is at least one
2472 **environment attribute** that matches the criteria below.

2473 The `<EnvironmentAttributeIsPresent>` element SHALL result in **true** if its **named**
2474 **environment attribute** is present. A result of **true** SHALL mean that a
2475 `<EnvironmentAttributeDesignator>` element for this **named environment attribute** SHALL
2476 return a bag consisting of at least one **attribute value**. The `MustBePresent` attribute governs
2477 whether this element returns **false** or **indeterminate** in the case of the absence of the **named**
2478 **environment attribute**. If the **named environment attribute** is not present and the
2479 `MustBePresent` attribute is set to **false** (its default value) this element SHALL result in **false**. If
2480 the **named environment attribute** is not present and the `MustBePresent` attribute is set to **true**,
2481 this element SHALL result in **indeterminate**. Regardless of the `MustBePresent` attribute, if it
2482 cannot be determined whether the **named environment attribute** is present or not present in the
2483 request context, or the value of the **named environment attribute** is unavailable, then the
2484 expression SHALL result in **indeterminate**.

2485 A **named environment attribute** SHALL be considered present if at least one **environment**
2486 **attribute** exists that matches the values of its corresponding `AttributeId`, `DataType`, and
2487 `Issuer` attributes as per the match semantics of the **AttributeDesignatorType** specification
2488 [Section 5.26].

2489 The `<EnvironmentAttributeIsPresent>` MAY be passed to the `<Apply>` element as an
2490 argument.

2491

```
2492 <xs:element name="EnvironmentAttributeIsPresent"
2493           type="xacml:AttributeDesignatorType" />
```

2494

2495 The <EnvironmentAttributeIsPresent> element is of the **AttributeDesignatorType**
2496 complex type.

2497 The <EnvironmentAttributeIsPresent> element has the following attributes:

2498 **AttributeId** [Required]

2499 This attribute SHALL specify the **AttributeId** with which to match the **environment**
2500 **attribute**.

2501 **DataType** [Required]

2502 This attribute SHALL specify the **DataType** with which to match the **environment attribute**.

2503 **Issuer** [Optional]

2504 This attribute, if supplied, SHALL specify the **Issuer** with which to match the **environment**
2505 **attribute**.

2506 **MustBePresent** [Optional]

2507 This attribute governs the whether the <EnvironmentAttributeIsPresent> element returns
2508 **false** or **indeterminate** in the case of the absence of the **named environment attribute**. If
2509 the **named environment attribute** is absent and **MustBePresent** is set to **false**, this
2510 element SHALL result in **false**. If the **named environment attribute** is absent and
2511 **MustBePresent** is set to **true**, this element SHALL result in **indeterminate**. If
2512 **MustBePresent** is not supplied, its default value SHALL be **false**.

2513 **5.33. Complex type SubjectAttributeDesignatorType**

2514 The **SubjectAttributeDesignatorType** complex type that extends the **AttributeDesignatorType**
2515 complex type. It is the base type for elements and extensions that refer to **named categorized**
2516 **subject attributes**. A **named categorized subject attribute** is defined as follows:

2517 A **subject** is represented by a <Subject> element of the <Subjects> element in the <xacml-
2518 context:Request> element. A **categorized subject** is a **subject** that contains a particular
2519 **subject category attribute**. A **subject attribute** is an attribute located in a particular **subject**. A
2520 **named subject attribute** is a **named attribute** for a **subject**. A **subject category attribute** is the
2521 **subject attribute** that matches the **named subject attribute** with the **AttributeId** of
2522 “urn:oasis:tc:xacml:1.0:subject:subject-category” and the **DataType** of
2523 “http://www.w3.org/2001/XMLSchema-instance#string”. A **named categorized**
2524 **subject attribute** is a **named subject attribute** for a particular **categorized subject**.

2525 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType** with a
2526 **SubjectCategory** attribute. The **SubjectAttributeDesignatorType** extends the match semantics
2527 of the **AttributeDesignatorType** such that it narrows the attribute search space to the specific
2528 **categorized subject** such that the value of the **SubjectCategory** attribute matches by string
2529 equality the value of the subject’s **subject category attribute**.

2530 If there are multiple **subjects** with the same **subject category attribute**, then they SHALL be
2531 treated as if they were one **categorized subject**.

2532 Elements and extensions of the **SubjectAttributeDesignatorType** complex type determine the
2533 presence of select **attribute values** associated with **named categorized subject attributes**.
2534 Elements and extensions of the **SubjectAttributeDesignatorType** SHALL NOT alter the match
2535 semantics of **named categorized subject attributes**, but MAY narrow the search space.

2536


```

2537 <xs:complexType name="SubjectAttributeDesignatorType" >
2538   <xs:complexContent>
2539     <xs:extension base="xacml: AttributeDesignatorType">
2540       <xs:attribute name="SubjectCategory"
2541         type="xs:anyURI"
2542         use="optional"
2543         default=
2544           "urn:org:oasis:tc:xacml:1.0:subject-category:access-subject" />
2545     </xs:extension>
2546   </xs:complexContent>
2547 </xs:complexType>

```

2548

2549 The <SubjectAttributeDesignatorType> complex type contains the following attribute in
2550 addition to the attributes of the **AttributeDesignatorType** complex type:

2551 SubjectCategory [Optional]

2552 This attribute SHALL specify the **categorized subject** from which to match **named subject**
2553 **attributes**. If SubjectCategory is not supplied, its default value SHALL
2554 urn:org:oasis:tc:xacml:1.0:subject-category:access-subject.

2555 **5.34. Element <SubjectAttributesPresent>**

2556 The <SubjectAttributesPresent> element determines whether a **named categorized subject**
2557 **attribute** is present or not. Its match semantics are that of the **SubjectAttributeDesignatorType**.

2558 The <SubjectAttributeIsPresent> element SHALL result in **true** if its **named categorized**
2559 **subject attribute** is present. A result of **true** SHALL mean that a
2560 <SubjectAttributeDesignator> element for the same **named categorized subject attribute**
2561 SHALL return a bag consisting of at least one **attribute value**. The **MustBePresent** attribute
2562 governs whether this element returns **false** or **indeterminate** in the case of the absence of the
2563 **named categorized subject attribute**. If the **named categorized subject attribute** is not present
2564 and the **MustBePresent** attribute is set to **false** (its default value) this element SHALL result in
2565 **false**. If the **named categorized subject attribute** is not present and the **MustBePresent**
2566 attribute is set to **true**, this element SHALL result in **indeterminate**. Regardless of the
2567 **MustBePresent** attribute, if it cannot be determined whether the **named categorized subject**
2568 **attribute** is present or not present in the request context, or the value of the **named categorized**
2569 **subject attribute** is unavailable, then the expression SHALL result in **indeterminate**.

2570 A **named categorized subject attribute** SHALL be considered present if at least one **subject**
2571 **attribute** exists that matches the values of its corresponding **AttributeId**, **DataType**, and
2572 **Issuer** attributes from the **categorized subject** as per the match semantics of the
2573 **SubjectAttributeDesignatorType** specification [Section 5.33]

2574 The <SubjectAttributeIsPresent> MAY be passed to the <Apply> element as an argument.

2575

```

2576 <xs:element name="SubjectAttributeIsPresent"
2577   type="xacml: AttributeDesignatorType" />

```

2578

2579 The <CategorizedAttributeIsPresent> element has the following attributes:

2580 AttributeId [Required]

2581 This attribute SHALL specify the `AttributeId` with which to match the **subject attribute**
2582 of the **categorized subject**.

2583 `DataType` [Required]

2584 This attribute SHALL specify the `DataType` with which to match the **subject attribute** of the
2585 **categorized subject**.

2586 `Issuer` [Optional]

2587 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **subject**
2588 **attribute** of the **categorized subject**.

2589 `MustBePresent` [Optional]

2590 This attribute governs the whether the `<SubjectAttributeIsPresent>` element returns
2591 **false** or **indeterminate** in the case of the absence of the **named categorized subject**
2592 **attribute**. If the **named categorized subject attribute** is absent and `MustBePresent` is
2593 set to **false**, this element SHALL result in **false**. If the **named categorized subject**
2594 **attribute** is absent and `MustBePresent` is set to **true**, this element SHALL result in
2595 **indeterminate**. If `MustBePresent` is not supplied, its default value SHALL be **false**.

2596 **5.35. Element <AttributeSelector>**

2597 The `AttributeSelector`'s `RequestContextPath` XML attribute SHALL contain a legal XPATH
2598 expression over the `<xacml-context:Request>` element. The `AttributeSelector` element
2599 evaluates to a bag of values of a single primitive type that is specified by the selector's `DataType`
2600 attribute. In the case where the XPath expression matches attributes in the request context by
2601 `AttributeId`, it must also match the attribute's `DataType` with the selector's `DataType`. In the case
2602 of using XPath 1.0, the value of the XPath expression is either a node-set, string value, numeric
2603 value, or boolean value. If the XPath 1.0 expression evaluates to a node-set, each node may
2604 consist of a string, numeric, boolean value, or a child node (i.e. structured node). In this case, each
2605 node is logically converted to string data by applying the "string" function defined in the XPath
2606 1.0 specification, resulting in a sequence of string data. In the single string, numeric, or boolean
2607 value case, the value is converted to string data by applying the "string" function defined in the
2608 XPath 1.0 specification, resulting in a sequence of one string data element. In XPath 2.0, the result
2609 of the XPath expression is a sequence of items (where an item is an atomic value or a node) or the
2610 error value. When the error value is returned, the **PD**P SHOULD return "Indeterminate".
2611 Otherwise, each node is logically converted to a string using the `xf:string` accessor function,
2612 resulting in a sequence of string data. The resulting sequence of string data is converted to a bag of
2613 primitive values that is implied by the type system.

2614 Support for the `<AttributeSelector>` element is OPTIONAL.

```
2615 <xs:element name="AttributeSelector" type="xacml:AttributeSelectorType"/>
2616 <xs:complexType name="AttributeSelectorType">
2617   <xs:attribute name="RequestContextPath" type="xs:anyURI" use="required"/>
2618   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
2619   <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"
2620   default="false"
2621 </xs:complexType>
```

2622 The `<AttributeSelector>` element is of **AttributeSelectorType** complex type.

2623 The `<AttributeSelector>` element has the following attributes:

2624 `RequestContextPath` [Required]

2625 An XPath expression into the request **context**. There SHALL be no restriction on the XPath
2626 syntax.

2627 `DataType` [Required]

2628 The data type of the **attribute**.

2629 `MustBePresent` [Optional]

2630 Whether or not designated **attribute** must be present in the **context**.

2631 **5.36. Element <AttributeValue>**

2632 The <AttributeValue> element SHALL contain a literal **attribute** value.

```
2633 <xs:element name="AttributeValue" type="xacml:AttributeValueType"/>
2634 <xs:complexType name="AttributeValueType" mixed="true">
2635 <xs:sequence>
2636 <xs:any namespace="##any" processContents="lax" minOccurs="0"
2637 maxOccurs="unbounded"/>
2638 </xs:sequence>
2639 <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
2640 <xs:anyAttribute namespace="##any" processContents="lax"/>
2641 </xs:complexType>
```

2642 The <AttributeValue> element is of **AttributeValueType** complex type.

2643 The <AttributeValue> element has following attributes:

2644 `DataType` [Required]

2645 The data type of the **attribute** value.

2646 **5.37. Element <Obligations>**

2647 The <Obligations> element SHALL contain a set of <Obligation> elements.

```
2648 <xs:element name="Obligations" type="xacml:ObligationsType"/>
2649 <xs:complexType name="ObligationsType">
2650 <xs:sequence>
2651 <xs:element ref="xacml:Obligation" maxOccurs="unbounded"/>
2652 </xs:sequence>
2653 </xs:complexType>
```

2654 The <Obligations> element is of **ObligationsType** complexType.

2655 <Obligation> [One to Many]

2656 A sequence of **obligations**

2657 **5.38. Element <Obligation>**

2658 The <Obligation> element SHALL contain an identifier for the **obligation** and a set of attributes
2659 that form arguments of the action defined by the **obligation**. The `FulfillOn` attribute SHALL
2660 indicate the **effect** for which this **obligation** applies.

```
2661 <xs:element name="Obligation" type="xacml:ObligationType"/>
2662 <xs:complexType name="ObligationType">
2663 <xs:sequence>
2664 <xs:element ref="xacml:AttributeAssignment" maxOccurs="unbounded"/>
```

```

2665     </xs:sequence>
2666     <xs:attribute name="ObligationId" type="xs:anyURI" use="required" />
2667     <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required" />
2668 </xs:complexType>

```

2669 The <Obligation> element is of **ObligationType** complexType. See Section 7.11 for a
 2670 description of how the set of obligations to be returned by the PDP is determined.

2671 The ObligationId [required]

2672 **Obligation** identifier. The value of the **obligation** identifier SHALL be interpreted by the
 2673 **PEP**.

2674 FulfillOn [required]

2675 The **effect** for which this **obligation** applies.

2676 <AttributeAssignment> [required]

2677 **Obligation** arguments assignment. The values of the **obligation** arguments SHALL be
 2678 interpreted by the **PEP**.

2679 **5.39. Element <AttributeAssignment>**

2680 The <AttributeAssignment> element SHALL contain an AttributeId and the corresponding
 2681 **attribute** value. The AttributeId is part of **attribute** meta-data, and is used when the **attribute**
 2682 cannot be referenced by its location in the <xacml-context:Request>. This situation may arise
 2683 in an <Obligation> element if the **obligation** includes parameters.

```

2684     <xs:element name="AttributeAssignment" type="xacml:AttributeAssignmentType" />
2685     <xs:complexType name="AttributeAssignmentType">
2686       <xs:complexContent>
2687         <xs:extension base="xacml:AttributeValueType">
2688           <xs:attribute name="AttributeId" type="xs:anyURI" use="required" />
2689         </xs:extension>
2690       </xs:complexContent>
2691     </xs:complexType>

```

2692 The <AttributeAssignment> element is of **AttributeAssignmentType** complex type.

2693 AttributeId [Required]

2694 The **attribute** Identifier

2695 DataType [Required]

2696 The data type for the assigned value.

2697

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

2698

2699

6.1. Element <Request>

2700

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

2701

2702

2703

The <Request> element consists of sections denoted by the <Subject>, <Resource>, <Action>, and <Environment> elements. There may be multiple <Subject> elements. Each section contains a sequence of XACML context <Attribute> elements associated with the *subject*, *resource*, *action*, and *environment* respectively.

2704

2705

2706

2707

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

2708

2709

2710

2711

2712

2713

2714

2715

2716

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

2717

The <Request> element contains the following elements:

2718

<Subject> [One to Many]

2719

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 making the *access* request. One *subject* might be a human user that initiated the application from which the request is being issued. Another *subject* might be the application's executable code that issued this request. Another *subject* might be the machine on which the application is executing. Another *subject* might be the target entity that is to be the recipient of the resource. Attributes of each of these entities MUST be enclosed in a separate <Subject> element.

2720

2721

2722

2723

2724

2725

2726

2727

<Resource> [Required]

2728

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

2729

2730

include a <ResourceContent> element.

2731

<Action> [Required]

2732

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

2733

2734

<Environment> [Optional]

2735

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

2736

2737

6.2. Element <Subject>

2738 The <Subject> element specifies a **subject** of a **decision request context** by listing a sequence
2739 of <Attribute> elements associated with the **subject**.

```
2740 <xs:element name="Subject" type="xacml-context:SubjectType" />
2741 <xs:complexType name="SubjectType">
2742   <xs:sequence>
2743     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2744     minOccurs="unbounded" />
2745   </xs:sequence>
2746 </xs:complexType>
```

2747 The <Subject> element is of **SubjectType** complex type.

2748 <Attribute> [Any Number]

2749 A sequence of **attributes** that apply to the **subject**.

2750 Every <Subject> element MUST contain one and only one <Attribute> with AttributeId
2751 "urn:oasis:names:tc:xacml:1.0:subject:subject-category". This **attribute**
2752 indicates a role that the parent <Subject> entity plays in making the **access** request. If this
2753 **attribute** is not present in a given <Subject> element, that <Subject> implicitly contains this
2754 **attribute** with the value of "urn:oasis:names:tc:xacml:1.0:subject:subject-
2755 category:access-subject", indicating that the **subject** is the entity ultimately associated
2756 with initiating the **access** request. Typically, a <Subject> element will also contain an
2757 <Attribute> with AttributeId "urn:oasis:names:tc:xacml:1.0:subject:subject-
2758 id", containing the identity of the **subject** entity.

2759 No more than one <Subject> element may contain an <Attribute> with the given value for
2760 AttributeId "urn:oasis:names:tc:xacml:1.0:subject:subject-category".

2761 A <Subject> element MAY contain additional <Attribute> elements.

6.3. Element <Resource>

2763 The <Resource> element specifies information about the **resource** for which access is being
2764 requested by listing a sequence of <Attribute> elements associated with the **resource**. It MAY
2765 include the **resource** content.

```
2766 <xs:element name="Resource" type="xacml-context:ResourceType" />
2767 <xs:complexType name="ResourceType">
2768   <xs:sequence>
2769     <xs:element ref="xacml-context:ResourceContent" minOccurs="0" />
2770     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2771     minOccurs="unbounded" />
2772   </xs:sequence>
2773 </xs:complexType>
```

2774 The <Resource> element is of **ResourceType** complex type.

2775 The <Resource> element contains the following elements:

2776 <ResourceContent> [Optional]

2777 The **resource** content.

2778 <Attribute> [Any Number]

2779 A sequence of **resource attributes**. The <Resource> element MUST contain one and
2780 only one <Attribute> with AttributeId
2781 "urn:oasis:names:tc:xacml:1.0:resource:resource-id". This **attribute**
2782 specifies the identity of the **resource** for which **access** is requested. The <Resource>
2783 element MAY contain additional <Attribute> elements.

2784 6.4. Element <ResourceContent>

2785 The <ResourceContent> element is a notional placeholder for the **resource** content. If an
2786 XACML **policy** references the contents of the **resource**, then the <ResourceContent> element
2787 is used as the reference point.

```
2788 <xs:complexType name="ResourceContentType" mixed="true">  
2789 <xs:sequence>  
2790 <xs:any namespace="##any" processContents="lax" minOccurs="0"  
2791 maxOccurs="unbounded" />  
2792 </xs:sequence>  
2793 <xs:anyAttribute namespace="##any" processContents="lax" />  
2794 </xs:complexType>
```

2795 The <ResourceContent> element is of **ResourceContentType** complex type.

2796 The <ResourceContent> element allows arbitrary elements and attributes.

2797 6.5. Element <Action>

2798 The <Action> element specifies the requested **action** to be performed on the **resource** by listing
2799 a set of <Attribute> elements associated with the **action**.

```
2800 <xs:element name="Action" type="xacml-context:ActionType" />  
2801 <xs:complexType name="ActionType">  
2802 <xs:sequence>  
2803 <xs:element ref="xacml-context:Attribute" minOccurs="0"  
2804 maxOccurs="unbounded" />  
2805 </xs:sequence>  
2806 </xs:complexType>
```

2807 The <Action> element is of **ActionType** complex type.

2808 The <Attribute> [Any Number]

2809 List of **attributes** of the **action** to be performed on the **resource**.

2810 6.6. Element <Environment>

2811 The <Environment> element contains a set of **attributes** of the **environment**. These **attributes**
2812 MAY form part of the **policy** evaluation.

```
2813 <xs:element name="Environment" type="xacml-context:EnvironmentType" />  
2814 <xs:complexType name="EnvironmentType">  
2815 <xs:sequence>  
2816 <xs:element ref="xacml-context:Attribute" minOccurs="0"  
2817 maxOccurs="unbounded" />  
2818 </xs:sequence>  
2819 </xs:complexType>
```

2820 The <Environment> element is of **EnvironmentType** complex type.

2821 The <Environment> element contains the following elements:

2822 <Attribute> [Any Number]

2823 A list of **environment attributes**. Environment attributes are attributes that are not
2824 associated with the **resource**, the **action**, or with any of the **subjects** of the **access**
2825 request.

2826 6.7. Element <Attribute>

2827 The <Attribute> element is the central abstraction of the request **context**. It contains an
2828 **attribute** value and **attribute** meta-data. The **attribute** meta-data comprises the **attribute**
2829 identifier, the **attribute** issuer and the **attribute** issue instant. **Attribute** designators and **attribute**
2830 selectors in the **policy** refer to **attributes** by this meta-data.

```
2831 <xs:element name="Attribute" type="xacml-context:AttributeType"/>  
2832 <xs:complexType name="AttributeType">  
2833   <xs:sequence>  
2834     <xs:element ref="xacml-context:AttributeValue" minOccurs="0"/>  
2835   </xs:sequence>  
2836   <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2837   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>  
2838   <xs:attribute name="Issuer" type="xs:string" use="optional"/>  
2839   <xs:attribute name="IssueInstant" type="xs:dateTime" use="optional"/>  
2840 </xs:complexType>
```

2841 The <Attribute> element is of **AttributeType** complex type.

2842 The <Attribute> element contains the following attributes and elements:

2843 AttributeId [Required]

2844 **Attribute** identifier. A number of identifiers are reserved by XACML to denote commonly
2845 used **attributes**.

2846 DataType [Required]

2847 **Attribute** data type.

2848 Issuer [Optional]

2849 **Attribute** issuer. This attribute value MAY be an x500Name that binds to a public key, or it
2850 may be some other identifier exchanged out-of-band by issuing and relying parties.

2851 IssueInstant [Optional]

2852 The date and time at which the **attribute** was issued.

2853 <AttributeValue> [Optional]

2854 At most one **attribute** value.

2855 6.8. Element <AttributeValue>

2856 The <AttributeValue> element contains the value of an **attribute**.

```
2857 <xs:element name="AttributeValue" type="xacml-context:AttributeValueType"/>  
2858 <xs:complexType name="AttributeValueType" mixed="true">  
2859   <xs:sequence>
```



```

2860     <xs:any namespace="##any" processContents="lax" minOccurs="0"
2861 maxOccurs="unbounded" />
2862   </xs:sequence>
2863   <xs:anyAttribute namespace="##any" processContents="lax" />
2864 </xs:complexType>

```

2865 The <AttributeValue> element is of **AttributeValueType** type.

2866 The data type of the <AttributeValue> MAY be specified by using the DataType attribute of the
 2867 parent <Attribute> element.

2868 **6.9. Element <Response>**

2869 The <Response> element encapsulates the **authorization decision** returned by the **PDP**. It
 2870 includes a sequence of one or more results with one <Result> element per requested **resource**.
 2871 Multiple results MAY be returned when the value of the "urn:oasis:xacml:1.0:resource:scope"
 2872 resource **attribute** in the request **context** is "Descendants". Support for multiple results is
 2873 OPTIONAL.

```

2874 <xs:element name="Response" type="xacml-context:ResponseType" />
2875 <xs:complexType name="ResponseType">
2876   <xs:sequence>
2877     <xs:element ref="xacml-context:Result" maxOccurs="unbounded" />
2878   </xs:sequence>
2879 </xs:complexType>

```

2880 The <Response> element is of **ResponseType** complex type.

2881 The <Response> element contains the following elements:

2882 <Result> [One to Many]

2883 An authorization decision result.

2884 **6.10. Element <Result>**

2885 The <Result> element represents an **authorization decision** result for the **resource** specified by
 2886 the ResourceId **attribute**. It MAY include a set of **obligations** that MUST be fulfilled by the **PEP**.
 2887 If the **PEP** does not understand an **obligation**, then it MUST act as if the **PDP** had denied **access**
 2888 to the requested **resource**.

```

2889 <xs:element name="Result" type="xacml-context:ResultType" />
2890 <xs:complexType name="ResultType">
2891   <xs:sequence>
2892     <xs:element ref="xacml-context:Decision" />
2893     <xs:element ref="xacml-context:Status" minOccurs="0" />
2894     <xs:element ref="xacml:Obligations" minOccurs="0" />
2895   </xs:sequence>
2896   <xs:attribute name="ResourceId" type="xs:anyURI" use="optional" />
2897 </xs:complexType>

```

2898 The <Result> element is of **ResultType** complex type.

2899 The <Result> element contains the following attributes and elements:

2900 ResourceId [Optional]

2901 The identifier of the requested **resource**. If this attribute is omitted, then the **resource**
2902 identity is specified by the "urn:oasis:names:tc:xacml:1.0:resource:resource-
2903 id" **resource attribute** in the <Request> element.

2904 <Decision> [Required]

2905 The **authorization decision**: "Permit", "Deny", "Indeterminate", or "Not-applicable".

2906 <Status> [Optional]

2907 Indicates whether errors occurred during evaluation of the request, and optionally,
2908 information about those errors.

2909 <xacml:Obligations> [Optional]

2910 A list of **obligations** that MUST be discharged by the **PEP**. If the **PEP** does not
2911 understand an **obligation**, then it MUST act as if the **PDP** had denied **access** to the
2912 requested **resource**. See Section 7.11 for a description of how the set of obligations to be
2913 returned by the PDP is determined.

2914 6.11. Element <Decision>

2915 The <Decision> element contains the result of **policy** evaluation.

```
2916 <xs:element name="Decision" type="xacml-context:DecisionType" />  
2917 <xs:simpleType name="DecisionType">  
2918 <xs:restriction base="xs:string">  
2919 <xs:enumeration value="Permit" />  
2920 <xs:enumeration value="Deny" />  
2921 <xs:enumeration value="Indeterminate" />  
2922 <xs:enumeration value="Not-applicable" />  
2923 </xs:restriction>  
2924 </xs:simpleType>
```

2925 The <Decision> element is of **DecisionType** simple type.

2926 The values of the <Decision> element have the following meanings:

2927 "Permit": the requested resource access is permitted.

2928 "Deny": the requested resource access is denied.

2929 "Indeterminate": the **PDP** is unable to evaluate the requested **resource access**. Reasons
2930 for such inability include: missing **attributes**, network errors while retrieving policies,
2931 division by zero during policy evaluation, syntax errors in the request or in the policy.

2932 "Not-applicable": the **PDP** does not have any policy that applies to this request.

2933 6.12. Element <Status>

2934 The <Status> element represents the status of the **authorization decision** result.

```
2935 <xs:element name="Status" type="xacml-context:StatusType" />  
2936 <xs:complexType name="StatusType">  
2937 <xs:sequence>  
2938 <xs:element ref="xacml-context:StatusCode" />  
2939 <xs:element ref="xacml-context:StatusMessage" minOccurs="0" />  
2940 <xs:element ref="xacml-context:StatusDetail" minOccurs="0" />  
2941 </xs:sequence>  
2942 </xs:complexType>
```

2943 The <Status> element is of **StatusType** complex type.

2944 The <Status> element contains the following elements:

2945 <StatusCode> [Required]

2946 Status code.

2947 <StatusMessage> [Optional]

2948 A status message describing the status code.

2949 <StatusDetail> [Optional]

2950 Additional status information.

2951 **6.13. Element <StatusCode>**

2952 The <StatusCode> element contains a major status code value and an optional sequence of
2953 minor status codes.

```
2954 <xs:element name="StatusCode" type="xacml-context:StatusCodeType" />
2955 <xs:complexType name="StatusCodeType">
2956   <xs:sequence>
2957     <xs:element ref="xacml-context:StatusCode" minOccurs="0" />
2958   </xs:sequence>
2959   <xs:attribute name="Value" type="xs:QName" use="required" />
2960 </xs:complexType>
```

2961 The <StatusCode> element is of **StatusCodeType** complex type.

2962 The <StatusCode> element contains the following attributes and elements:

2963 Value [Required]

2964 See Section B.7 for a list of values.

2965 <StatusCode> [Any Number]

2966 Minor status code. This status code qualifies its parent status code.

2967 **6.14. Element <StatusMessage>**

2968 The <StatusMessage> element is a free-form description of the status code.

```
2969 <xs:element name="StatusMessage" type="xs:string" />
```

2970 The <StatusMessage> element is of **xs:string** type.

2971 **6.15. Element <StatusDetail>**

2972 The <StatusDetail> element qualifies the <Status> element with additional information.

```
2973 <xs:element name="StatusDetail" type="xacml-context:StatusDetailType" />
2974 <xs:complexType name="StatusDetailType">
2975   <xs:sequence>
2976     <xs:any namespace="##any" processContents="lax" minOccurs="0"
2977 maxOccurs="unbounded" />
2978   </xs:sequence>
2979 </xs:complexType>
```

2980 The <StatusDetail> element is of **StatusDetailType** complex type.

2981 The <StatusDetail> element allows arbitrary xml content.

2982 Inclusion of a <StatusDetail> element is optional. However, if a **PDP** returns one of the
2983 following XACML-defined <StatusCode> values and includes a <StatusDetail> element, then
2984 the following rules apply.

2985 urn:oasis:names:tc:xacml:1.0:status:ok

2986 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “ok” status value.

2987 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

2988 A **PDP** MAY choose not to return any <StatusDetail> information or MAY choose to return a
2989 <StatusDetail> element containing one or more <xacml-context:Attribute> elements. If
2990 the **PDP** includes <AttributeValue> elements in the <Attribute> element, then this indicates
2991 the acceptable values for that **attribute**. If no <AttributeValue> elements are included, then
2992 this indicates the names of **attributes** that the **PDP** failed to resolve during its evaluation. The list
2993 of **attributes** may be partial or complete. There is no guarantee by the **PDP** that supplying the
2994 missing values or **attributes** will be sufficient to satisfy the **policy**.

2995 urn:oasis:names:tc:xacml:1.0:status:syntax-error

2996 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “syntax-error” status
2997 value. A syntax error may represent either a problem with the **policy** being used or with the
2998 request **context**. The **PDP** MAY return a <StatusMessage> describing the problem.

2999 urn:oasis:names:tc:xacml:1.0:status:processing-error

3000 A **PDP** MUST NOT return <StatusDetail> element in conjunction with the “processing-error”
3001 status value. This status code indicates an internal problem in the **PDP**. For security reasons, the
3002 **PDP** MAY choose to return no further information to the **PEP**. In the case of a divide-by-zero error
3003 or other computational error, the **PDP** MAY return a <StatusMessage> describing the nature of
3004 the error.

3005 7. Functional requirements (normative)

3006 This section specifies certain functional requirements that are not directly associated with the
3007 production or consumption of a particular XACML element.

3008 7.1. Policy enforcement point

3009 This section describes the requirements for the **PEP**.

3010 An application functions in the role of the **PEP** if it guards access to a set of **resources** and asks
3011 the **PDP** for an **authorization decision**. The **PEP** MUST abide by the **authorization decision** in
3012 the following way:

3013 A **PEP** SHALL allow access to the **resource** only if a valid XACML response of "Permit" is returned
3014 by the **PDP**. The **PEP** SHALL deny access to the **resource** in all other cases. An XACML
3015 response of "Permit" SHALL be considered valid only if the **PEP** understands all of the **obligations**
3016 contained in the response.

3017

7.2. Base policy

3018 A **PDP** SHALL represent one **policy** or **policy set**, called its **base policy**. This base **policy** MAY be
 3019 a <Policy> element containing a <Target> element that matches every possible **decision**
 3020 **request**, or (for instance) it MAY be a <Policy> element containing a <Target> element that
 3021 matches only a specific **subject**. In such cases, the base policy SHALL form the root-node of a
 3022 tree of policies connected by <PolicyIdReference> and <PolicySetIdReference>
 3023 elements to all the **rules** that may be applicable to any **decision request** that the **PDP** is capable
 3024 of evaluating.

3025 In the case of a **PDP** that retrieves **policies** according to the **decision request** that it is processing,
 3026 the base policy SHALL contain a <Policy> element containing a <Target> element that matches
 3027 every possible **decision request** and a PolicyCombiningAlgId attribute with the value "Only-
 3028 one-applicable". In other words, the **PDP** SHALL return an error if it retrieves policies that do not
 3029 form a single tree.

3030

7.3. Target evaluation

3031 The **target** value SHALL be "Match" if the **subjects**, **resource** and **action** specified in the request
 3032 **context** are all present in (i.e., within the scope of) the **target**. Its value SHALL be "No-match" if
 3033 one or more of the **subjects**, **resource** or **action** specified in the request **context** is not present in
 3034 the **target**. Its value SHALL be "Indeterminate" if any **attribute** value referenced in the **target**
 3035 cannot be obtained.

3036

7.4. Condition evaluation

3037 The **condition** value SHALL be "True" if the <Condition> element is absent, or if it evaluates to
 3038 "True" for the **attribute** values supplied in the request **context**. Its value is "False" if the
 3039 <Condition> element evaluates to "False" for the **attribute** values supplied in the request
 3040 **context**. If any **attribute** value referenced in the **condition** cannot be obtained, then the **condition**
 3041 SHALL evaluate to "Indeterminate".

3042

7.5. Rule evaluation

3043 A **rule** has a value that can be calculated by evaluating its contents. **Rule** evaluation involves
 3044 separate evaluation of the **rule's target** and **condition**. The **rule** truth table is shown in Table 1.

| Target | Condition | Rule Value |
|-----------------|-----------------|------------------|
| "Match" | "True" | Effect |
| "Match" | "False" | "Not-applicable" |
| "Match" | "Indeterminate" | "Indeterminate" |
| Not "Match" | Don't care | "Not-applicable" |
| "Indeterminate" | Don't care | "Indeterminate" |

3045

Table 1 - Rule truth table

3046 If the **target** value is "No-match" or "Indeterminate" then the **rule** value SHALL be "Not-applicable"
 3047 or "Indeterminate", respectively, regardless of the value of the **condition**. For these cases,
 3048 therefore, the **condition** need not be evaluated in order to determine the **rule** value.

3049 If the **target** value is “Match” and the **condition** value is “True”, then the **effect** specified in the **rule**
 3050 SHALL determine the **rule** value.

3051 **7.6. Policy evaluation**

3052 A **policy** has a value that can be calculated by evaluating its contents. **Policy** evaluation involves
 3053 separate evaluation of the **policy's target** and **rules**. The **policy** truth table is shown in Table 2.

| Target | Rule values | Policy Value |
|-----------------|--|--|
| “Match” | At least one rule value is its Effect | Specified by the rule-combining algorithm |
| “Match” | All rule values are “Not-applicable” | “Not-applicable” |
| “Match” | At least one rule value is “Indeterminate” | Specified by the rule-combining algorithm |
| Not “Match” | Don’t-care | “Not-applicable” |
| “Indeterminate” | Don’t-care | “Indeterminate” |

Table 2 - Rule truth table

3054

3055 A Rules value of "At-least-one-applicable" SHALL be used if the <Rule> element is absent, or if
 3056 one or more of the **rules** contained in the **policy** is applicable to the **decision request** (i.e., returns
 3057 a value of “Effect”; see Section 7.5). A value of “None-applicable” SHALL be used if no **rule**
 3058 contained in the **policy** is applicable to the request and if no **rule** contained in the **policy** returns a
 3059 value of “Indeterminate”. If no **rule** contained in the **policy** is applicable to the request but one or
 3060 more **rule** returns a value of “Indeterminate”, then **rules** SHALL evaluate to "Indeterminate".

3061 If the **target** value is "No-match" or “Indeterminate” then the **policy** value SHALL be “Not-
 3062 applicable” or “Indeterminate”, respectively, regardless of the value of the **rules**. For these cases,
 3063 therefore, the **rules** need not be evaluated in order to determine the **policy** value.

3064 If the **target** value is “Match” and the **rules** value is “At-least-one-applicable” or “Indeterminate”,
 3065 then the **rule-combining algorithm** specified in the **policy** SHALL determine the **policy** value.

3066 **7.7. Policy Set evaluation**

3067 A **policy set** has a value that can be calculated by evaluating its contents. **Policy set** evaluation
 3068 involves separate evaluation of the **policy set's target** and **policies**. The **policy set** truth table is
 3069 shown in Table 3.

| Target | Policy values | Policy Set Value |
|--------|---|--|
| Match | At least one policy value is its Effect | Specified by the policy-combining algorithm |
| Match | All policy values are “Not-applicable” | “Not-applicable” |
| Match | At least one policy value is | Specified by the policy-combining algorithm |

| | | |
|---------------|-----------------|------------------|
| | "Indeterminate" | |
| Not match | Don't-care | "Not-applicable" |
| Indeterminate | Don't-care | "Indeterminate" |

Table 3 - Rule truth table

3070

3071 A **policies** value of "At-least-one-applicable" SHALL be used if there are no contained or
 3072 referenced **policies** or **policy sets**, or if one or more of the **policies** or **policy sets** contained in or
 3073 referenced by the **policy set** is applicable to the **decision request** (i.e., returns a value determined
 3074 by its **rule-combining algorithm**; see Section 7.6). A value of "None-applicable" SHALL be used if
 3075 no **policy** or **policy set** contained in or referenced by the **policy set** is applicable to the request
 3076 and if no **policy** or **policy set** contained in or referenced by the **policy set** returns a value of
 3077 "Indeterminate". If no **policy** or **policy set** contained in or referenced by the **policy set** is
 3078 applicable to the request but one or more **policy** or **policy set** returns a value of "Indeterminate",
 3079 then **policies** SHALL evaluate to "Indeterminate".

3080 If the **target** value is "No-match" or "Indeterminate" then the **policy set** value SHALL be "Not-
 3081 applicable" or "Indeterminate", respectively, regardless of the value of the **policies**. For these
 3082 cases, therefore, the **policies** need not be evaluated in order to determine the **policy set** value.

3083 If the **target** value is "Match" and the **policies** value is "At-least-one-applicable" or "Indeterminate",
 3084 then the **policy-combining algorithm** specified in the **policy set** SHALL determine the **policy set**
 3085 value.

3086 7.8. Hierarchical resources

3087 It is often the case that a **resource** is organized as a hierarchy (e.g. file system, XML document).
 3088 Some access requesters may request **access** to an entire subtree of a **resource** specified by a
 3089 node. XACML allows the **PEP** (or **context handler**) to specify whether the **decision request** is
 3090 just for a single **resource** or for a subtree below the specified **resource**. The latter is equivalent to
 3091 repeating a single request for each node in the entire subtree. When a request **context** contains a
 3092 resource attribute of type

3093 "urn:oasis:names:tc:xacml:1.0:resource:scope"

3094 with a value of "Immediate", or if it does not contain that **attribute**, then the **decision request**
 3095 SHALL be interpreted to apply to just the single **resource** specified by the **ResourceId attribute**.

3096 When the

3097 "urn:oasis:names:tc:xacml:1.0:resource:scope"

3098 **attribute** has the value "Children", the **decision request** SHALL be interpreted to apply to the
 3099 specified **resource** and its immediate children **resources**.

3100 When the

3101 "urn:oasis:names:tc:xacml:1.0:resource:scope"

3102 **attribute** has the value "Descendants", the **decision request** SHALL be interpreted to apply to
 3103 both the specified **resource** and all its descendant **resources**.

3104 In the case of "Children" and "Descendants", the **authorization decision** MAY include multiple
 3105 results for the multiple sub-nodes in the **resource** sub-tree.

3106 An XACML **authorization response** MAY contain multiple <Result> elements. In this case, the
3107 <Status> element SHOULD be included only in the first <Result> element (the remaining
3108 <Result> elements SHOULD NOT include the <Status> element).

3109 Note that the method by which the **PDP** discovers whether the **resource** is hierarchically organized
3110 or not is outside the scope of XACML.

3111 7.9. Attributes

3112 **Attributes** are specified in the request **context** and are referred to in the **policy** by **subject**,
3113 **resource**, **action** and **environment attribute** designators and **attribute** selectors. A *named*
3114 *attribute* is the term used for the criteria that the specific **subject**, **resource**, **action** and
3115 **environment attribute** designators and selectors use to refer to **attributes** in the **subject**,
3116 **resource**, **action** and **environment** elements of the request **context**, respectively.

3117 7.9.1. Attribute Matching

3118 A *named attribute* has specific criteria with which to match **attributes** within the **context**. An
3119 **attribute** specifies `AttributeId`, `DataType` and `Issuer` attributes, and each *named attribute*
3120 also specifies `AttributeId`, `DataType` and `Issuer` attributes. A *named attribute* SHALL match
3121 an **attribute** if the values of their respective `AttributeId`, `DataType` and `Issuer` attributes
3122 match within their particular element, e.g. **subject**, **resource**, **action** or **environment**, of the
3123 **context**. The `AttributeId` attribute MUST match, by URI equality, that of the `AttributeId`
3124 attribute of the **attribute**. The `DataType` attribute MUST match, by URI equality, that of the
3125 `DataType` attribute of the same **attribute**. If the `Issuer` attribute is supplied, it MUST match, by
3126 URI equality, the `Issuer` attribute of the same **attribute**. If the `Issuer` attribute is not supplied in
3127 the *named attribute*, then the matching of the **attribute** to the *named attribute* SHALL be governed
3128 by `AttributeId` and `DataType` attributes alone, regardless of the presence, absence, or actual
3129 value of the `Issuer` attribute. In the case of the **attribute** selector, the matching of the **attribute** to
3130 the *named attribute* SHALL be governed by the XPath expression, `DataType` and `Issuer`
3131 attributes.

3132 7.9.2. Attribute Retrieval

3133 The **PDP** SHALL request the values of **attributes** in the request **context** from the **context handler**.
3134 The **PDP** SHALL reference the **attributes** as if they were in a physical request **context** document,
3135 but the **context handler** is responsible for obtaining and supplying the requested values. The
3136 **context handler** SHALL return the values of **attributes** that match the **attribute** designator or
3137 **attribute** selector and form them into a **bag** of values with the specified `DataType` attribute. If no
3138 **attributes** from the request **context** match, then the **attribute** SHALL be considered missing. If
3139 the **attribute** is missing, the `MustBePresent` attribute governs whether the **attribute**
3140 designator or **attribute** selector returns an empty **bag** or an **indeterminate** result. If
3141 `MustBePresent` is "False" (default value), then a missing attribute results in an empty **bag**. If
3142 `MustBePresent` is "True", then a missing **attribute** results in "Indeterminate". This
3143 "Indeterminate" result SHALL be handled in accordance with the specification of the encompassing
3144 expressions, rules, policies, and policy sets. If the result is "Indeterminate", then the
3145 `AttributeId`, `DataType` and `Issuer` of the **attribute** MAY be listed in the **authorization**
3146 **decision** as described in Section 7.10. However, a **PDP** MAY choose not to return such
3147 information for security reasons.

3148

7.9.3. Environment Attributes

3149 *Environment attributes* are listed in Section B.8. If a value for one of these *attributes* is supplied
3150 in the *decision request*, then the *context handler* SHALL use that value. Otherwise, the *context*
3151 *handler* SHALL supply a value. For the date and time *attributes*, the supplied value SHALL have
3152 the semantics of "date and time that apply to the *decision request*".

3153

7.9.4. Subject Attributes

3154 The "subject-category" *attribute* is a *named attribute* with the criteria of an AttributeId of
3155 "urn:oasis:names:tc:xacml:1.0:subject:subject-category" and DataType attribute
3156 of "http://www.w3.org/2001/XMLSchema#string". For each <Subject> element in the
3157 *decision request*, if a value for the "subject-category" *attribute* is supplied, then the *context*
3158 *handler* SHALL use that value. Otherwise, the *context handler* SHALL supply the default value
3159 "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject". If there is
3160 more than one "subject-category" *attribute* supplied in the *decision request* for any given
3161 <Subject> element, then the *decision request* is invalid.

3162

7.10. Authorization decision

3163 Given a valid XACML *policy* or *policy set*, a compliant XACML *PDP* MUST evaluate the *policy* as
3164 specified in Sections 5, 6 and 4.2. The *PDP* MUST return a response *context*, with one
3165 <Decision> element of value "Permit", "Deny", "Indeterminate" or "Not-applicable".

3166 If the *PDP* cannot make a decision, then an "Indeterminate" <Decision> element contents SHALL
3167 be returned. The *PDP* MAY return a <Decision> element contents of "Indeterminate" with a
3168 status code of:

3169 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3170 signifying that more information is needed. In this case, the <Status> element MAY list the
3171 names and data-types of any *attributes* of the *subjects* and the *resource* that are needed by the
3172 *PDP* to refine its decision. A *PEP* MAY resubmit a refined request *context* in response to a
3173 <Decision> element contents of "Indeterminate" with a status code of

3174 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3175 by adding *attribute* values for the *attribute* names that were listed in the previous response. When
3176 the *PDP* returns a <Decision> element contents of "Indeterminate", with a status code of

3177 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3178 it MUST NOT list the names and data-types of any *attribute* of the *subject* or the *resource* for
3179 which values were supplied in the original request. Note, this requirement forces the *PDP* to
3180 eventually return an *authorization decision* of "Permit", "Deny" or "Indeterminate" with some other
3181 status code, in response to successively-refined requests.

3182

7.11. Obligations

3183 A *policy* or *policy set* may contain one or more *obligations*. When such a *policy* or *policy set* is
3184 evaluated, an *obligation* SHALL be passed up to the next level of evaluation (the enclosing or
3185 referencing *policy set* or *authorization decision*) only if the *effect* of the *policy* or *policy set*
3186 being evaluated matches the value of the `xacml:FulfillOn` attribute of the *obligation*.
3187

3188 As a consequence of this procedure, no **obligations** SHALL be returned to the **PEP** if the **policies**
3189 or **policy sets** from which they are drawn are not evaluated, or if their evaluated result is
3190 "Indeterminate" or "Not-applicable", or if the **decision** resulting from evaluating the **policy** or **policy**
3191 **set** does not match the **decision** resulting from evaluating an enclosing **policy set**.
3192
3193 If the **PDP's** evaluation is viewed as a tree of **policy sets** and **policies**, each of which returns
3194 "Permit" or "Deny", then the set of **obligations** returned by the **PDP** to the **PEP** will include only the
3195 **obligations** associated with those paths where the **effect** at each level of evaluation is the same as
3196 the **effect** being returned by the **PDP**.
3197 A **PEP** that receives a valid XACML response of "Permit" with **obligations** SHALL be responsible
3198 for fulfilling *all* of those **obligations**. A **PEP** that receives an XACML response of "Deny" with
3199 **obligations** SHALL be responsible for fulfilling all of the **obligations** that it *understands*.

3200 8. XACML extensibility points (non-normative)

3201 This section describes the points within the XACML model and schema where extensions can be
3202 added

3203 8.1. Extensible XML attribute types

3204 The following XML attributes have values that are URIs or QNames. These may be extended by
3205 the creation of new URIs or QNames associated with new semantics for these attributes.

3206 AttributeId,

3207 AttributeValue,

3208 DataType,

3209 FunctionId,

3210 MatchId,

3211 ObligationId,

3212 PolicyCombiningAlgId,

3213 RuleCombiningAlgId,

3214 StatusCode.

3215 See Section 5 for definitions of these attribute types.

3216 8.2. Extensible XACML attribute types

3217 The following XACML standard `AttributeIds` associated with the XACML standard element:
3218 `<Attribute>` have values that are URIs or QNames. These may be extended by the creation of
3219 new URIs or QNames associated with new semantics for these attributes.

3220 urn:oasis:names:tc:xacml:1.0:subject:subject-category.

3221

8.3. Structured attributes

3222

An XACML <AttributeValue> element MAY contain an instance of a structured XML data-type.

3223

Section A.3 describes a number of standard techniques to identify data items within such a

3224

structured attribute. Listed here are some additional techniques that require XACML extensions.

3225

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

3226

3227

3228

3229

3230

3231

3232

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

3233

3234

3235

9. Security and privacy considerations (non-normative)

3236

3237

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

3238

3239

3240

3241

9.1. Threat model

3242

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

3243

3244

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

3245

3246

3247

3248

3249

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

3250

3251

3252

3253

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

3254

3255

3256

3257

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

3258

3259

9.1.1. Unauthorized disclosure

3260 XACML does not specify any inherent mechanisms for confidentiality of the messages exchanged
3261 between actors. Therefore, an adversary could observe the messages in transit. Under certain
3262 security policies, disclosure of this information is a violation. Disclosure of **attributes** or the types
3263 of **decision requests** that a **subject** submits may be a breach of privacy policy. In the commercial
3264 sector, the consequences of unauthorized disclosure of personal data may range from
3265 embarrassment to the custodian to imprisonment and large fines in the case of medical or financial
3266 data.

3267 Unauthorized disclosure is addressed by confidentiality mechanisms.

3268

9.1.2. Message replay

3269 A message replay attack is one in which the adversary records and replays legitimate messages
3270 between XACML actors. This attack may lead to denial of service, the use of out-of-date
3271 information or impersonation.

3272 Prevention of replay attacks requires the use of message freshness mechanisms.

3273 Note that encryption of the message does not mitigate a replay attack since the message is just
3274 replayed and does not have to be understood by the adversary.

3275

9.1.3. Message insertion

3276 A message insertion attack is one in which the adversary inserts messages in the sequence of
3277 messages between XACML actors.

3278 The solution to a message insertion attack is to use mutual authentication and a message
3279 sequence integrity mechanism between the actors. It should be noted that just using SSL mutual
3280 authentication is not sufficient. This only proves that the other party is the one identified by the
3281 subject of the X.509 certificate. In order to be effective, it is necessary to confirm that the certificate
3282 subject is authorized to send the message.

3283

9.1.4. Message deletion

3284 A message deletion attack is one in which the adversary deletes messages in the sequence of
3285 messages between XACML actors. Message deletion may lead to denial of service. However, a
3286 properly designed XACML system should not render an incorrect authorization decision as a result
3287 of a message deletion attack.

3288 The solution to a message deletion attack is to use a message integrity mechanism between the
3289 actors.

3290

9.1.5. Message modification

3291 If an adversary can intercept a message and change its contents, then they may be able to alter an
3292 **authorization decision**. Message integrity mechanisms can prevent a successful message
3293 modification attack.

3294

9.1.6. Not-applicable results

3295 A result of "Not-applicable" means that the **PDP** did not have a policy whose target matched the
3296 information in the **decision request**. In general, we highly recommend using a "default-deny"

3297 policy, so that when a **PDP** would have returned "Not-applicable", a result of "Deny" is returned
3298 instead.

3299 In some security models, however, such as is common in many Web Servers, a result of "Not-
3300 applicable" is treated as equivalent to "Permit". There are particular security considerations that
3301 must be taken into account for this to be safe. These are explained in the following paragraphs.

3302 If "Not-applicable" is to be treated as "Permit", it is vital that the matching algorithms used by the
3303 policy to match elements in the decision request are closely aligned with the data syntax used by
3304 the applications that will be submitting the decision request. A failure to match will be treated as
3305 "Permit", so an unintended failure to match may allow unintended access.

3306 A common example of this is a Web Server. Commercial http responders allow a variety of
3307 syntaxes to be treated equivalently. The "%" can be used to represent characters by hex value.
3308 The URL path "../" provides multiple ways of specifying the same value. Multiple character sets
3309 may be permitted and, in some cases, the same printed character can be represented by different
3310 binary values. Unless the matching algorithm used by the policy is sophisticated enough to catch
3311 these variations, unintended access may be permitted.

3312 It is safe to treat "Not-applicable" as "Permit" only in a closed environment where all applications
3313 that formulate a decision request can be guaranteed to use the exact syntax expected by the
3314 policies used by the **PDP**. In a more open environment, where decision requests may be received
3315 from applications that may use any legal syntax, it is strongly recommended that "Not-applicable"
3316 NOT be treated as "Permit" unless matching rules have been very carefully designed to match all
3317 possible applicable inputs, regardless of syntax or type variations.

3318 **9.1.7. Negative rules**

3319 A negative **rule** is one that is based on a **predicate** not being "True". If not used with care,
3320 negative **rules** can lead to policy violation, therefore some authorities recommend that they not be
3321 used. However, negative **rules** can be extremely efficient in certain cases, so XACML has chosen
3322 to include them. Nevertheless, it is recommended that they be used with care and avoided if
3323 possible.

3324 A common use for negative **rules** is to deny **access** to an individual or subgroup when their
3325 membership in a larger group would otherwise permit them access. For example, we might want to
3326 write a **rule** that allows all Vice Presidents to see the unpublished financial data, except for Joe,
3327 who is only a Ceremonial Vice President and can be indiscreet in his communications. If we have
3328 complete control of the administration of **subject attributes**, a superior approach would be to
3329 define "Vice President" and "Ceremonial Vice President" as distinct groups and then define **rules**
3330 accordingly. However, in some environments this approach may not be feasible. (It is worth noting
3331 in passing that, generally speaking, referring to individuals in **rules** does not scale well. Generally,
3332 shared **attributes** are preferred.)

3333 If not used with care, negative **rules** can lead to policy violation in two common cases. They are:
3334 when **attributes** are suppressed and when the base group changes. An example of suppressed
3335 **attributes** would be if we have a policy that **access** should be permitted, *unless* the **subject** is a
3336 credit risk. If it is possible that the **attribute** of being a credit risk may be unknown to the **PDP** for
3337 some reason, then unauthorized **access** may be permitted. In some environments, the **subject**
3338 may be able to suppress the publication of **attributes** by the application of privacy controls, or the
3339 server or repository that contains the information may be unavailable for accidental or intentional
3340 reasons.

3341 An example of a changing base group would be if there is a policy that everyone in the engineering
3342 department may change software source code, except for secretaries. Suppose now that the
3343 department was to merge with another engineering department and the intent is to maintain the
3344 same policy. However, the new department also includes individuals identified as administrative

3345 assistants, who ought to be treated in the same way as secretaries. Unless the policy is altered,
3346 they will unintentionally be permitted to change software source code. Problems of this type are
3347 easy to avoid when one individual administers all **policies**, but when administration is distributed,
3348 as XACML allows, this type of situation must be explicitly guarded against.

3349 **9.2. Safeguards**

3350 **9.2.1. Authentication**

3351 Authentication provides the means for one party in a transaction to determine the identity of the
3352 other party in the transaction. Authentication may be in one direction, or it may be bilateral.

3353 Given the sensitive nature of **access control** systems, it is important for a **PEP** to authenticate the
3354 identity of the **PDP** to which it sends **decision requests**. Otherwise, there is a risk that an
3355 adversary could provide false or invalid **authorization decisions**, leading to a policy violation.

3356 It is equally important for a **PDP** to authenticate the identity of the **PEP** and assess the level of trust
3357 to determine what, if any, sensitive data should be passed. One should keep in mind that even
3358 simple "Permit" or "Deny" responses could be exploited if an adversary were allowed to make
3359 unlimited requests to a **PDP**.

3360 Many different techniques may be used to provide authentication, such as co-located code, a
3361 private network, a VPN or digital signatures. Authentication may also be performed as part of the
3362 communication protocol used to exchange the **contexts**. In this case, authentication may be
3363 performed at the message level or at the session level.

3364 **9.2.2. Policy administration**

3365 If the contents of **policies** are exposed outside of the **access control** system, potential **subjects**
3366 may use this information to determine how to gain unauthorized **access**.

3367 To prevent this threat, the repository used for the storage of **policies** may itself require **access**
3368 **control**. In addition, the <Status> element should be used to return values of missing **attributes**
3369 only when exposure of the identities of those **attributes** will not compromise security.

3370 **9.2.3. Confidentiality**

3371 Confidentiality mechanisms ensure that the contents of a message can be read only by the desired
3372 recipients and not by anyone else who encounters the message while it is in transit. There are two
3373 areas in which confidentiality should be considered: one is confidentiality during transmission; the
3374 other is confidentiality within a <Policy> element.

3375 **9.2.3.1. Communication confidentiality**

3376 In some environments it is deemed good practice to treat all data within an **access control** system
3377 as confidential. In other environments, **policies** may be made freely available for distribution,
3378 inspection and audit. The idea behind keeping **policy** information secret is to make it more difficult
3379 for an adversary to know what steps might be sufficient to obtain unauthorized **access**. Regardless
3380 of the approach chosen, the security of the **access control** system should not depend on the
3381 secrecy of the **policy**.

3382 Any security concerns or requirements related to transmitting or exchanging XACML <policy>
3383 elements are outside the scope of the XACML standard. While it is often important to ensure that
3384 the integrity and confidentiality of <policy> elements is maintained when they are exchanged

3385 between two parties, it is left to the implementers to determine the appropriate mechanisms for their
3386 environment.

3387 Communications confidentiality can be provided by a confidentiality mechanism, such as SSL.
3388 Using a point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points
3389 is compromised.

3390 **9.2.3.2. Statement level confidentiality**

3391 In some cases, an implementation may want to encrypt only parts of an XACML <Policy>
3392 element.

3393 The XML Encryption Syntax and Processing Candidate Recommendation from W3C can be used
3394 to encrypt all or parts of an XML document. This specification is recommended for use with
3395 XACML.

3396 It should go without saying that if a repository is used to facilitate the communication of cleartext
3397 (i.e., unencrypted) **policy** between the **PAP** and **PDP**, then a secure repository should be used to
3398 store this sensitive data.

3399 **9.2.4. Policy integrity**

3400 The XACML **policy**, used by the **PDP** to evaluate the request **context**, is the heart of the system.
3401 Therefore, maintaining its integrity is essential. There are two aspects to maintaining the integrity of
3402 the **policy**. One is to ensure that <Policy> elements have not been altered since they were
3403 originally created by the **PAP**. The other is to ensure that <Policy> elements have not been
3404 inserted or deleted from the set of **policies**.

3405 In many cases, both aspects can be achieved by ensuring the integrity of the actors and
3406 implementing session-level mechanisms to secure the communication between actors. The
3407 selection of the appropriate mechanisms is left to the implementers. However, when **policy** is
3408 distributed between organizations to be acted on at a later time, or when the **policy** travels with the
3409 protected resource, it would be useful to sign the **policy**. In these cases, the XML Signature
3410 Syntax and Processing standard from W3C is recommended to be used with XACML.

3411 Digital signatures should only be used to ensure the integrity of the statements. Digital signatures
3412 should not be used as a method of selecting or evaluating **policy**. That is, the **PDP** should not
3413 request a **policy** based on who signed it or whether or not it has been signed (as such a basis for
3414 selection would, itself, be a matter of policy). However, the **PDP** must verify that the key used to
3415 sign the **policy** is one controlled by the purported issuer of the **policy**. The means to do this are
3416 dependent on the specific signature technology chosen and are outside the scope of this document.

3417 **9.2.5. Policy identifiers**

3418 Since **policies** can be referenced by their identifiers, it is the responsibility of the **PAP** to ensure
3419 that these are unique. Confusion between identifiers could lead to misidentification of the
3420 **applicable policy**. This specification is silent on whether a **PAP** must generate a new identifier
3421 when a **policy** is modified or may use the same identifier in the modified **policy**. This is a matter of
3422 administrative practice. However, care must be taken in either case. If the identifier is reused,
3423 there is a danger that other **policies** or **policy sets** that reference it may be adversely affected.
3424 Conversely, if a new identifier is used, these other **policies** may continue to use the prior **policy**,
3425 unless it is deleted. In either case the results may not be what the **policy** administrator intends.

3426

9.2.6. Trust model

3427 Discussions of authentication, integrity and confidentiality mechanisms necessarily assume an
3428 underlying trust model: how can one actor come to believe that a given key is uniquely associated
3429 with a specific, identified actor so that the key can be used to encrypt data for that actor or verify
3430 signatures (or other integrity structures) from that actor? Many different types of trust model exist,
3431 including strict hierarchies, distributed authorities, the Web, the bridge and so on.

3432 It is worth considering the relationships between the various actors of the **access control** system in
3433 terms of the interdependencies that do and do not exist.

- 3434 • None of the entities of the authorization system are dependent on the **PEP**. They may
3435 collect data from it, for example authentication, but are responsible for verifying it.
- 3436 • The correct operation of the system depends on the ability of the **PEP** to actually enforce
3437 **policy** decisions.
- 3438 • The **PEP** depends on the **PDP** to correctly evaluate **policies**. This in turn implies that the
3439 **PDP** is supplied with the correct inputs. Other than that, the **PDP** does not depend on the
3440 **PEP**.
- 3441 • The **PDP** depends on the **PAP** to supply appropriate policies. The **PAP** is not dependent
3442 on other components.

3443

9.2.7. Privacy

3444 It is important to be aware that any transactions that occur with respect to **access control** may
3445 reveal private information about the actors. For example, if an XACML **policy** states that certain
3446 data may only be read by **subjects** with "Gold Card Member" status, then any transaction in which
3447 a **subject** is permitted **access** to that data leaks information to an adversary about the **subject's**
3448 status. Privacy considerations may therefore lead to encryption and/or to **access control policies**
3449 surrounding the enforcement of XACML **policy** instances themselves: confidentiality-protected
3450 channels for the request/response protocol messages, protection of **subject attributes** in storage
3451 and in transit, and so on.

3452 Selection and use of privacy mechanisms appropriate to a given environment are outside the scope
3453 of XACML. The decision regarding whether, how and when to deploy such mechanisms is left to
3454 the implementers associated with the environment.

10. Conformance (normative)

3455

10.1. Introduction

3456

3457 The XACML specification addresses two aspects of conformance:

3458 1. The OASIS procedure for ratification of a committee specification as an OASIS standard requires
3459 that three independent implementers attest that they are "successfully using" the committee
3460 specification, and

3461 2. The XACML specification defines a number of functions, etc. that have somewhat specialist
3462 application, therefore they are not required to be implemented in an implementation that claims to
3463 conform with the OASIS standard.

3464

10.2. Attestation

3465 An implementer MAY attest to be "successfully using" the XACML committee specification provided
 3466 the implementation successfully executes a set of test-cases. The test cases are hosted by Sun
 3467 Microsystems and can be located from the XACML web page. The site hosting the test cases
 3468 contains a full description of the test cases and how to execute them.

3469

10.3. Conformance tables

3470 This section lists those portions of the specification that MUST be included in an implementation of
 3471 a *PDP* that claims to conform with XACML v1.0.

3472 Note: "M" means mandatory-to-implement. "O" means optional.

3473

10.3.1. Schema elements

3474 The implementation MUST support those schema elements that are marked "M".

| Namespace | Element | M/O |
|---------------|-------------------------------------|-----|
| xacml:Policy | Action | M |
| xacml:Policy | ActionAttributeDesignator | M |
| xacml:Policy | ActionMatch | M |
| xacml:Policy | Actions | M |
| xacml:Policy | AnyAction | M |
| xacml:Policy | AnyResource | M |
| xacml:Policy | AnySubject | M |
| xacml:Policy | Apply | M |
| xacml:Policy | AttributeAssignment | O |
| xacml:Policy | AttributeSelector | O |
| xacml:Policy | AttributeValue | M |
| xacml:Policy | Condition | M |
| xacml:Policy | Description | M |
| xacml:Policy | EnvironmentAttributeDesignator | M |
| Xacml:Policy | Function | M |
| xacml:Policy | Obligation | O |
| xacml:Policy | Obligations | O |
| xacml:Policy | Policy | M |
| xacml:Policy | PolicyDefaults | O |
| xacml:Policy | PolicyIdReference | M |
| xacml:Policy | PolicySet | M |
| xacml:Policy | PolicySetDefaults | O |
| xacml:Policy | PolicySetIdReference | M |
| xacml:Policy | Resource | M |
| xacml:Policy | ResourceAttributeDesignator | M |
| xacml:Policy | ResourceMatch | M |
| xacml:Policy | Resources | M |
| xacml:Policy | Rule | M |
| xacml:Policy | Subject | M |
| xacml:Policy | SubjectAttributeDesignator | M |
| xacml:Policy | QualifiedSubjectAttributeDesignator | M |
| xacml:Policy | SubjectMatch | M |
| xacml:Policy | Subjects | M |
| xacml:Policy | Target | M |
| xacml:Policy | XPathVersion | O |
| xacml:Context | Action | M |

| | | |
|---------------|-----------------|---|
| xacml:Context | Attribute | M |
| xacml:Context | AttributeValue | M |
| xacml:Context | Decision | M |
| xacml:Context | Environment | M |
| xacml:Context | Obligations | O |
| xacml:Context | Request | M |
| xacml:Context | Resource | M |
| xacml:Context | ResourceContent | O |
| xacml:Context | Response | M |
| xacml:Context | Result | M |
| xacml:Context | Status | O |
| xacml:Context | StatusCode | O |
| xacml:Context | StatusDetail | O |
| xacml:Context | StatusMessage | O |
| xacml:Context | Subject | M |

3475 **10.3.2. Identifier Prefixes**

3476 The following identifier prefixes are reserved by XACML.

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

3477 **10.3.3. Algorithms**

3478 The implementation MUST include the rule- and policy-combining algorithms associated with the
3479 following identifiers that are marked "M".

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

3480 **10.3.4. Status Codes**

3481 Implementation support for the urn:oasis:names:tc:xacml:1.0:context:status element is optional, but
3482 if the element is supported, then the following status codes must be supported and must be used in
3483 the way XACML has specified.

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

3484 **10.3.5. Attributes**

3485 The implementation MUST support the attributes associated with the following attribute identifiers
 3486 as specified by XACML. The value for these attributes MUST be provided by the *PDP*, so, unlike
 3487 most other attributes, their semantics are not transparent to the *PDP* implementation.

| 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 |
| urn:oasis:names:tc:xacml:1.0:subject:subject-category | M |

3488 **10.3.6. Identifiers**

3489 The implementation MUST use the attributes associated with the following identifiers in the way
 3490 XACML has defined. This requirement pertains primarily to implementations of a *PAP* or *PEP* that
 3491 use XACML, since the semantics of the attributes are transparent to the *PDP*.

| Identifier | M/O |
|--|-----|
| urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name | O |
| urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address | O |
| urn:oasis:names:tc:xacml:1.0:resource:resource-location | O |
| urn:oasis:names:tc:xacml:1.0:resource:resource-id | O |
| urn:oasis:names:tc:xacml:1.0:resource:scope | O |
| urn:oasis:names:tc:xacml:1.0:resource:simple-file-name | O |
| urn:oasis:names:tc:xacml:1.0::action:action-id | M |
| urn:oasis:names:tc:xacml:1.0::action:implied-action | M |
| 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 |

3492 **10.3.7. Data Types**

3493 The implementation MUST support the data types associated with the following identifiers marked
 3494 "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#date | M |

| | |
|---|---|
| http://www.w3.org/2001/XMLSchema#dateTime | 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 |
| http://www.w3.org/TR/xquery-operators:dayTimeDuration | M |
| http://www.w3.org/TR/xquery-operators:yearMonthDuration | M |
| urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name | M |
| urn:oasis:names:tc:xacml:1.0:data-type:x500Name | M |

3495 **10.3.8. Functions**

3496 The implementation MUST properly process those functions associated with the identifiers marked
3497 with an "M".

3498 xmlns:function="urn:oasis:names:tc:xacml:1.0:function"

| Function | M/O |
|---|-----|
| function:string-equal | M |
| function:boolean-equal | M |
| function:integer-equal | M |
| function:double-equal | M |
| function:date-equal | M |
| function:time-equal | M |
| function:dateTime-equal | M |
| function:anyURI-equal | M |
| function:x500Name-equal | M |
| function:rfc822name-equal | M |
| function:hexBinary-equal | M |
| function:base64Binary-equal | M |
| function:integer-add | M |
| function:double-add | M |
| function:integer-subtract | M |
| function:double-subtract | M |
| function:integer-multiply | M |
| function:double-multiply | M |
| function:integer-divide | M |
| function:double-divide | M |
| function:integer-mod | M |
| function:integer-abs | M |
| function:double-abs | M |
| function:round | M |
| function:floor | M |
| function:string-normalize-space | M |
| function:string-normalize-to-lower-case | M |
| function:double-to-integer | M |
| function:integer-to-double | M |
| function:or | M |
| function:and | M |
| function:n-of | M |
| function:not | M |
| function:present | M |
| function:integer-greater-than | M |
| function:integer-greater-than-or-equal | M |
| function:integer-less-than | M |
| function:integer-less-than-or-equal | M |
| function:double-greater-than | M |
| function:double-greater-than-or-equal | M |

| | |
|--|---|
| function:double-less-than | M |
| function:double-less-than-or-equal | M |
| function:dateTime-add-dayTimeDuration | M |
| function:dateTime-add-yearMonthDuration | M |
| function:dateTime-subtract-dayTimeDuration | M |
| function:dateTime-subtract-yearMonthDuration | M |
| function:date-add-yearMonthDuration | M |
| function:date-subtract-yearMonthDuration | M |
| function:string-greater-than | M |
| function:string-greater-than-or-equal | M |
| function:string-less-than | M |
| function:string-less-than-or-equal | M |
| function:time-greater-than | M |
| function:time-greater-than-or-equal | M |
| function:time-less-than | M |
| function:time-less-than-or-equal | M |
| function:dateTime-greater-than | M |
| function:dateTime-greater-than-or-equal | M |
| function:dateTime-less-than | M |
| function:dateTime-less-than-or-equal | M |
| function:date-greater-than | M |
| function:date-greater-than-or-equal | M |
| function:date-less-than | M |
| function:date-less-than-or-equal | M |
| function:string-one-and-only | M |
| function:string-bag-size | M |
| function:string-is-in | M |
| function:string-bag | M |
| function:boolean-one-and-only | M |
| function:boolean-bag-size | M |
| function:boolean-is-in | M |
| function:boolean-bag | M |
| function:integer-one-and-only | M |
| function:integer-bag-size | M |
| function:integer-is-in | M |
| function:integer-bag | M |
| function:double-one-and-only | M |
| function:double-bag-size | M |
| function:double-is-in | M |
| function:double-bag | M |
| function:date-one-and-only | M |
| function:date-bag-size | M |
| function:date-is-in | M |
| function:date-bag | M |
| function:dateTime-one-and-only | M |
| function:dateTime-bag-size | M |
| function:dateTime-is-in | M |
| function:dateTime-bag | M |
| function:anyURI-one-and-only | M |
| function:anyURI-bag-size | M |
| function:anyURI-is-in | M |
| function:anyURI-bag | M |
| function:hexBinary-one-and-only | M |
| function:hexBinary-bag-size | M |
| function:hexBinary-is-in | M |
| function:hexBinary-bag | M |
| function:base64Binary-one-and-only | M |

| | |
|---|---|
| function:base64Binary-bag-size | M |
| function:base64Binary-is-in | M |
| function:base64Binary-bag | M |
| function:dayTimeDuration-one-and-only | M |
| function:dayTimeDuration-bag-size | M |
| function:dayTimeDuration-is-in | M |
| function:dayTimeDuration-bag | M |
| function:yearMonthDuration-one-and-only | M |
| function:yearMonthDuration-bag-size | M |
| function:yearMonthDuration-is-in | M |
| function:yearMonthDuration-bag | M |
| function:x500Name-one-and-only | M |
| function:x500Name-bag-size | M |
| function:x500Name-is-in | M |
| function:x500Name-bag | M |
| function:rfc822Name-one-and-only | M |
| function:rfc822Name-bag-size | M |
| function:rfc822Name-is-in | M |
| function:rfc822Name-bag | M |
| function:any-of | M |
| function:all-of | M |
| function:any-of-any | M |
| function:all-of-any | M |
| function:any-of-all | M |
| function:all-of-all | M |
| function:map | M |
| function:x500Name-match | M |
| function:rfc822Name-match | M |
| function:xpath-node-count | O |
| function:xpath-node-equal | O |
| function:xpath-node-match | O |

3499 11. References

- 3500 [DS] D. Eastlake et al., *XML-Signature Syntax and Processing*,
3501 <http://www.w3.org/TR/xmlsig-core/>, World Wide Web Consortium.
- 3502 [Haskell] Haskell, a purely functional language. Available at
3503 <http://www.haskell.org/>
- 3504 [Hinton94] Hinton, H, M, Lee,, E, S, The Compatibility of Policies, Proceedings 2nd
3505 ACM Conference on Computer and Communications Security, Nov 1994,
3506 Fairfax, Virginia, USA.
- 3507 [IEEE754] IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-
3508 7653-8, IEEE Product No. SH10116-TBR
- 3509 [Kudo00] Kudo M and Hada S, XML document security based on provisional
3510 authorization, Proceedings of the Seventh ACM Conference on Computer
3511 and Communications Security, Nov 2000, Athens, Greece, pp 87-96.
- 3512 [LDAP-1] RFC2256, A summary of the X500(96) User Schema for use with LDAPv3,
3513 section 5, M Wahl, December 1997 <http://www.ietf.org/rfc/rfc2798.txt>
- 3514 [LDAP-2] RFC2798, Definition of the inetOrgPerson, M. Smith, April 2000
3515 <http://www.ietf.org/rfc/rfc2798.txt>
- 3516 [MathML] Mathematical Markup Language (MathML), Version 2.0, W3C
3517 Recommendation, 21 February 2001. Available at:
3518 <http://www.w3.org/TR/MathML2/>

| | | |
|------|--------------------|--|
| 3519 | [Perritt93] | Perritt, H. Knowbots, Permissions Headers and Contract Law, Conference on Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment, April 1993. Available at: http://www.ifla.org/documents/infopol/copyright/perh2.txt |
| 3520 | | |
| 3521 | | |
| 3522 | | |
| 3523 | [RBAC] | Role-Based Access Controls, David Ferraiolo and Richard Kuhn, 15th National Computer Security Conference, 1992. Available at: http://csrc.nist.gov/rbac |
| 3524 | | |
| 3525 | | |
| 3526 | [RegEx] | XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, Appendix D. Available at: http://www.w3.org/TR/xmlschema-0/ |
| 3527 | | |
| 3528 | [RFC2119] | S. Bradner, <i>Key words for use in RFCs to Indicate Requirement Levels</i> , http://www.ietf.org/rfc/rfc2119.txt , IETF RFC 2119, March 1997 |
| 3529 | | |
| 3530 | [SAML] | Security Assertion Markup Language available from http://www.oasis-open.org/committees/security/#documents |
| 3531 | | |
| 3532 | [Sloman94] | Sloman, M. Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Volume 2, part 4. Plenum Press. 1994. |
| 3533 | | |
| 3534 | | |
| 3535 | [XF] | XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft 16 August 2002. Available at: http://www.w3.org/TR/xquery-operators |
| 3536 | | |
| 3537 | [XS] | XML Schema. Available at: http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/ |
| 3538 | | |
| 3539 | [XPath] | XML Path Language (XPath), Version 1.0, W3C Recommendation 16 November 1999. Available at: http://www.w3.org/TR/xpath |
| 3540 | | |
| 3541 | [XSLT] | XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999. Available at: http://www.w3.org/TR/xslt |
| 3542 | | |
| 3543 | | |

3544 **Appendix A. Standard data types, functions and**
3545 **their semantics (normative)**

3546 **A.1. Introduction**

3547 This section contains a specification of the data-types and functions used in XACML to create
3548 **predicates** for a **rule's condition** and **target** matches.

3549 This specification combines the various standards set forth by IEEE and ANSI for string
3550 representation of numeric values, as well as the evaluation of arithmetic functions.

3551 This section describes the primitive data-types, **bags** and construction of expressions using
3552 XACML constructs. Finally, each standard function is named and its operational semantics are
3553 described.

3554 **A.2. Primitive types**

3555 Although XML instances represent all data-types as strings, an XACML **PDP** must reason about
3556 types of data that, while they have string representations, are not just strings. Types such as
3557 boolean, integer and double **MUST** be converted from their XML string representations to values
3558 that can be compared with values in their domain of discourse, such as numbers. The following
3559 primitive data-types are specified for use with XACML and have explicit data representations:

- 3560 • <http://www.w3.org/2001/XMLSchema#string>
- 3561 • <http://www.w3.org/2001/XMLSchema#boolean>
- 3562 • <http://www.w3.org/2001/XMLSchema#integer>
- 3563 • <http://www.w3.org/2001/XMLSchema#double>
- 3564 • <http://www.w3.org/2001/XMLSchema#date>
- 3565 • <http://www.w3.org/2001/XMLSchema#dateTime>
- 3566 • <http://www.w3.org/2001/XMLSchema#anyURI>
- 3567 • <http://www.w3.org/2001/XMLSchema#hexBinary>
- 3568 • <http://www.w3.org/2001/XMLSchema#base64Binary>
- 3569 • <http://www.w3.org/TR/xquery-operators#dayTimeDuration>
- 3570 • <http://www.w3.org/TR/xquery-operators#yearMonthDuration>
- 3571 • <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
- 3572 • <urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name>

3573 A.3. Structured types

3574 An XACML `<AttributeValue>` element MAY contain an instance of a structured XML data-type,
3575 for example `<ds:KeyInfo>`. XACML 1.0 supports several ways for comparing such
3576 `<AttributeValue>` elements.

3577 1. In some cases, such an `<AttributeValue>` element MAY be compared using one of the
3578 XACML string functions, such as “`regex-string-match`”, described below. This requires
3579 that the structured data `<AttributeValue>` to be given the `DataType="xsi:string"`. For
3580 example, a structured data type that is actually a `ds:KeyInfo/KeyName` would appear in the
3581 Context as:

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

3582 In general, this method will not be adequate unless the structured data type is quite simple.

3583 2. An `<AttributeSelector>` element MAY be used to select the value of a leaf sub-
3584 element of the structured data-type by means of an XPath expression. That value MAY
3585 then be compared using one of the supported XACML functions appropriate for its primitive
3586 data-type. This method requires support by the PDP for the optional XPath expressions
3587 feature.

3588 3. An `<AttributeSelector>` element MAY be used to select the value of any node in the
3589 structured type by means of an XPath expression. This node MAY then be compared
3590 using one of the XPath-based functions described in Section A14.13. This method requires
3591 support by the PDP for the optional XPath expressions and XPath functions features.

3592 A.4. Representations

3593 An XACML *PDP* SHALL be capable of converting string representations into various primitive data
3594 types. For integers and doubles, XACML SHALL use the conversions described in IBM Standard
3595 Decimal Arithmetic [IBMDSA].

3596 This document combines the various standards set forth by IEEE and ANSI for string
3597 representation of numeric values.

3598 XACML defines two additional data-types; these are “`urn:oasis:names:tc:xacml:1.0:data-`
3599 `type:x500Name`” and “`urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name`”. These types
3600 represent identifiers for **subjects** and appear in several standard applications, such as TLS/SSL
3601 and electronic mail.

3602 The “`urn:oasis:names:tc:xacml:1.0:data-type:x500Name`” primitive type represents an X.500
3603 Distinguished Name. The string representation of an X.500 distinguished name is specified in IETF
3604 RFC 2253 “Lightweight Directory Access Protocol (v3): UTF-8 String Representation of
3605 Distinguished Names”.¹

1 An earlier RFC, RFC 1779 “A String Representation of Distinguished Names”, is less restrictive, so `xacml:x500Name` uses the syntax in RFC 2253 for better interoperability.

3606 The "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" primitive type represents electronic mail
3607 addresses, and its string representation is specified by RFC 822.

3608 An RFC822 name consists of a *local-part* followed by "@" followed by a *domain-part*. The *local-*
3609 *part* is case-sensitive, while the *domain-part* (which is usually a DNS host name) is not case-
3610 sensitive.²

3611 A.5. Bags

3612 XACML defines implicit collections of its primitive types. XACML refers to a collection of values that
3613 are of a single primitive type as a **bag**. **Bags** of primitive types are needed because selections of
3614 nodes from an XML **resource** or XACML request **context** may return more than one value.

3615 The <AttributeSelector> element uses an XPath expression to specify the selection of data
3616 from an XML **resource**. The result of an XPath expression is termed a *node-set*, which contains all
3617 the leaf nodes from the XML **resource** that match the predicate in the XPath expression. Based on
3618 the various indexing functions provided in the XPath specification, it SHALL be implied that a
3619 resultant node-set is the collection of the matching nodes. XACML also defines the
3620 <AttributeDesignator> **element** to have the same matching methodology for attributes in the
3621 XACML request **context**.

3622 The values in a **bag** are not ordered, and some of the values may be duplicates. There SHALL be
3623 no notion of a **bag** containing **bags**, or a **bag** containing values of differing types. I.e. a **bag** in
3624 XACML SHALL contain only values that are of the same primitive type.

3625 A.6. Expressions

3626 XACML specifies expressions in terms of the following elements. Each expression evaluates to
3627 one of the primitive types, or a **bag** of one of the primitive types. In addition, XACML defines an
3628 evaluation result of "Indeterminate", which is said to be the result of an invalid expression, or an
3629 operational error occurring during the evaluation of the expression.

3630 XACML defines the following elements to be legal XACML expressions:

- 3631 • <AttributeValue>
- 3632 • <SubjectAttributeDesignator>
- 3633 • <SubjectAttributeSelector>
- 3634 • <QualifiedSubjectAttributeDesignator>
- 3635 • <ResourceAttributeDesignator>
- 3636 • <ActionAttributeDesignator>
- 3637 • <EnvironmentAttributeDesignator>

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

- 3638 • <AttributeSelector>
- 3639 • <Apply>
- 3640 • <Condition>
- 3641 • <Function>

3642 A.7. Element <AttributeValue>

3643 The <AttributeValue> element SHALL represent an explicit value of a primitive type. For
3644 example:

```
3645 <Apply FunctionId="function:integer-equal" >  
3646   <AttributeValue  
3647     DataType="http://www.w3.org/2001/XMLSchema#integer">123</AttributeValue>  
3648   <AttributeValue  
3649     DataType="http://www.w3.org/2001/XMLSchema#integer">123</AttributeValue>  
3650 </Apply>
```

3651 A.8. Elements <AttributeDesignator> and 3652 <AttributeSelector>

3653 The <AttributeDesignator> and <AttributeSelector> elements SHALL evaluate to a **bag**
3654 of a specific primitive type. The type SHALL be inferred from the function in which it appears. Each
3655 element SHALL contain a URI or XPath expression, respectively, to identify the required **attribute**
3656 values. If an operational error were to occur while finding the values, the value of the element
3657 SHALL be set to "Indeterminate". If the required **attribute** cannot be located, then the value of the
3658 element SHALL be set to an empty **bag** of the inferred primitive type.

3659 In the special case of the <QualifiedSubjectAttributeDesignator> element, the sequence
3660 of <SubjectMatch> elements SHALL be evaluated as if each <SubjectMatch> element, while
3661 applied only to one particular **subject**, appeared in a conjunctive sequence.

3662 A.9. Element <Apply>

3663 XACML function calls are represented by the <Apply> element. The function to be applied is
3664 named in the `FunctionId` attribute of this element. The value of the <Apply> element SHALL be
3665 set to either a primitive type or a **bag** of a primitive type, whose type SHALL be inferred from the
3666 `functionId`. The arguments of a function SHALL be the values of the XACML expressions that
3667 are contained as ordered elements in an <Apply> element. The legal number of arguments within
3668 an <Apply> element SHALL depend upon the `functionId`.

3669 A.10. Element <Condition>

3670 The <Condition> element MAY appear in the <Rule> element as the premise for emitting the
3671 corresponding **effect** of the **rule**. The <Condition> element has the same structure as the

3672 <Apply> element, with the restriction that its result SHALL be of type
3673 "http://www.w3.org/2001/XMLSchema#boolean". The evaluation of the <Condition> element
3674 SHALL follow the same evaluation semantics as those of the <Apply> element.

3675 **A.11. Element <Function>**

3676 The <Function> element names a standard XACML function or an extension function in its
3677 FunctionId attribute. The <Function> element MAY be used as an argument in functions that
3678 take a function as an argument.

3679 **A.12. Matching elements**

3680 Matching elements appear in the <Target> element of *rules*, *policies* and *policy sets*. They are
3681 the following:

- 3682 • <SubjectMatch>
- 3683 • <ResourceMatch>
- 3684 • <ActionMatch>
- 3685 • <EnvironmentMatch>

3686 These elements represent boolean expressions over *attributes* of the *subject*, *resource*, *action*
3687 and *environment*, respectively.

3688 The match elements: <SubjectMatch>, <ResourceMatch>, <ActionMatch> and
3689 <EnvironmentMatch> SHALL use functions that match two arguments, returning a result type of
3690 "xs:boolean", to perform the match evaluation. The function used for determining a match is named
3691 in the MatchId attribute of these elements. Each argument to the named function MUST match
3692 the appropriate primitive types for the <AttributeDesignator> or <AttributeSelector>
3693 element and the following explicit *attribute* value, such that the explicit *attribute* value is placed as
3694 the first argument to the function, while an element of the *bag* returned by the
3695 <AttributeDesignator> or <AttributeSelector> element is placed as the second
3696 argument to the function.

3697 The XACML standard functions that may be used as a MatchId attribute value are:

- 3698 function: *type-equal*
- 3699 function: *type-greater-than*
- 3700 function: *type-greater-than-or-equal*
- 3701 function: *type-less-than*
- 3702 function: *type-less-than-or-equal*
- 3703 function: *type-match*

3704 The evaluation semantics for a match is as follows. If an operational error were to occur while
3705 evaluating the <AttributeDesignator> or <AttributeSelector> element, then the result of
3706 the entire expression SHALL be "Indeterminate". If the <AttributeDesignator> or
3707 <AttributeSelector> element were to evaluate to an empty *bag*, then the result of the

3708 expression SHALL be "False". Otherwise, the match function SHALL be applied between the
3709 explicit **attribute** value and each element of the **bag** returned from the <AttributeDesignator>
3710 or <AttributeSelector> element. If at least one of those function applications were to evaluate
3711 to "True", then the result of the entire expression SHALL be "True". Otherwise, if at least one of the
3712 function applications results in "Indeterminate", then the result SHALL be "Indeterminate". Finally,
3713 only if all function applications evaluate to "False", SHALL the result of the entire expression be
3714 "False".

3715 A match can equivalently be expressed in a **target** or a **condition**. For instance, the match
3716 expression that compares a "subject-name" starting with the name "John" can be expressed as
3717 follows:

```
3718 <SubjectMatch MatchId="function:regexp-string-match">  
3719   <SubjectAttributeDesignator AttributeId="subject-name"/>  
3720   <AttributeValue  
3721     DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>  
3722 </SubjectMatch>
```

3723 Alternatively, it can be expressed as an <Apply> element in the **condition** by using the
3724 "function:any-of" function, as follows:

```
3725 <Apply FunctionId="function:any-of">  
3726   <Function FunctionId="function:regexp-string-match"/>  
3727   <AttributeValue  
3728     DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>  
3729   <SubjectAttributeDesignator AttributeId="subject-name"/>  
3730 </Apply>
```

3731 For the match elements: <SubjectMatch>, <ResourceMatch>, <ActionMatch> and
3732 <EnvironmentMatch> that appear in the <Target> element of a <Rule>, <Policy> or
3733 <PolicySet> the value specified by the MatchId attribute SHALL be restricted to the following
3734 functions:

- 3735 • "function:type-equal" (for each primitive *type*),
- 3736 • "function:regexp-string-match",
- 3737 • "function:rfc822Name-match" and
- 3738 • "function:x500Name-match",

3739 and only those functions. Functions that are strictly within an extension to XACML should not
3740 appear as a value to the MatchId attribute in this case. Restricting the MatchId attribute to these
3741 functions facilitates the use of indexing to find the **applicable policy** for a particular **authorization**
3742 **request**.

3743 **A.13. Arithmetic evaluation**

3744 IEEE 754 [IEEE 754] specifies how to evaluate arithmetic functions in a context, which specifies
3745 defaults for precision, rounding, etc. XACML SHALL use this specification for the evaluation of all
3746 integer and double functions relying on the *Extended Default Context*, enhanced with double
3747 precision:

3748 flags - all set to 0

3749 trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the "division-by-zero" trap
3750 enabler, which SHALL be set to 1

3751 precision - is set to the designated double precision
3752 rounding - is set to round-half-even (IEEE 854 §4.1)

3753 **A.14. XACML standard functions**

3754 XACML specifies the following functions that are prefixed with the "function:" relative name space
3755 identifier.

3756 **A14.1 Equality predicates**

3757 The following functions are the *equality* functions for the various primitive types. Each function for a
3758 particular type follows a specified standard convention for that type. If an argument of one of these
3759 functions were to evaluate to "Indeterminate", then the function SHALL be set to "Indeterminate".

3760 • string-equal

3761 This function SHALL take two arguments of "http://www.w3.org/2001/XMLSchema#string"
3762 and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". The function
3763 SHALL return "True" if and only if the value of both of its arguments are of equal length and
3764 each string is determined to be equal byte-by-byte according to the function "integer-equal".

3765 • boolean-equal

3766 This function SHALL take two arguments of
3767 "http://www.w3.org/2001/XMLSchema#boolean" and SHALL return "True" if and only if both
3768 values are equal.

3769 • integer-equal

3770 This function SHALL take two arguments of type
3771 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return an
3772 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on
3773 integers according to IEEE 754 [IEEE 754].

3774 • double-equal

3775 This function SHALL take two arguments of type
3776 "http://www.w3.org/2001/XMLSchema#double" and SHALL return an
3777 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on
3778 doubles according to IEEE 754 [IEEE 754].

3779 • date-equal

3780 This function SHALL take two arguments of type
3781 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
3782 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
3783 according to the "op:date-equal" function [XQO Section 8.3.11].

3784 • time-equal

3785 This function SHALL take two arguments of type
3786 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
3787 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according
3788 to the "op:time-equal" function [XQO Section 8.3.14].

3789 • dateTime-equal

3790 This function SHALL take two arguments of type
3791 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
3792 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
3793 according to the "op:dateTime-equal" function [XQO Section 8.3.8].

3794 • anyURI-equal

3795 This function SHALL take two arguments of type
3796 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return an
3797 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation
3798 according to the "op:anyURI-equal" function [XQO Section 10.2.1].

3799 • x500Name-equal

3800 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-
3801 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It
3802 shall return "True" if and only if each Relative Distinguished Name (RDN) in the two
3803 arguments matches. Two RDNs shall be said to match if and only if the result of the
3804 following operations is "True"³.

- 3805 1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory
3806 Access Protocol (v3): UTF-8 String Representation of Distinguished Names".
- 3807 2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute
3808 ValuePairs in that RDN in ascending order when compared as octet strings
3809 (described in ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").
- 3810 3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key
3811 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section
3812 4.1.2.4 "Issuer".

3813 • rfc822Name-equal

3814 This function SHALL take two arguments of type "urn:oasis:names:tc:xacml:1.0:data-
3815 type:rfc822Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean".
3816 This function SHALL determine whether two "urn:oasis:names:tc:xacml:1.0:data-
3817 type:rfc822Name" arguments are equal. An RFC822 name consists of a *local-part* followed
3818 by "@" followed by a *domain-part*. The *local-part* is case-sensitive, while the *domain-part*
3819 (which is usually a DNS host name) is not case-sensitive. Perform the following
3820 operations:

- 3821 1. Normalize the *domain-part* of each argument to lower case
- 3822 2. Compare the expressions by applying the function "function:string-equal" to the
3823 normalized arguments.

3824 • hexBinary-equal

3825 This function SHALL take two arguments of type
3826 "http://www.w3.org/2001/XMLSchema#hexBinary" and SHALL return an
3827 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL return "True" if the
3828 octet sequences represented by the value of both arguments have equal length and are
3829 equal in a conjunctive, point-wise, comparison using the "function:integer-equal". The

³ ITU-T Rec. X.520 contains rules for matching X500 names, but these are very complex and require knowledge of the syntax of various AttributeTypes. IETF RFC 3280 contains simplified matching rules that the XACML x500Name-equal function uses.

3830 conversion from the string representation to an octet sequence SHALL be as specified in
3831 [XS Section 8.2.15]

3832 • base64Binary-equal

3833 This function SHALL take two arguments of type
3834 "http://www.w3.org/2001/XMLSchema#base64Binary" and SHALL return an
3835 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL return "True" if the
3836 octet sequences represented by the value of both arguments have equal length and are
3837 equal in a conjunctive, point-wise, comparison using the "function:integer-equal". The
3838 conversion from the string representation to an octet sequence SHALL be as specified in
3839 [XS Section 8.2.16]

3840 **A14.2 Arithmetic functions**

3841 All of the following functions SHALL take two arguments of the specified *type*, integer or double,
3842 and SHALL return an element of integer or double type, respectively. However, the "add" functions
3843 MAY take more than two arguments. Each function evaluation SHALL proceed as specified by
3844 their logical counterparts in IEEE 754 [IEEE 754]. In an expression that contains any of these
3845 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3846 "Indeterminate". In the case of the divide functions, if the divisor is zero, then the function SHALL
3847 evaluate to "Indeterminate".

3848 • integer-add

3849 This function MAY have two or more arguments.

3850 • double-add

3851 This function MAY have two or more arguments.

3852 • integer-subtract

3853 • double-subtract

3854 • integer-multiply

3855 • double-multiply

3856 • integer-divide

3857 • double-divide

3858 • integer-mod

3859 The following functions SHALL take a single argument of the specified *type*. The round and floor
3860 functions SHALL take a single argument of type "http://www.w3.org/2001/XMLSchema#double" and
3861 return type "http://www.w3.org/2001/XMLSchema#double". In an expression that contains any of
3862 these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3863 "Indeterminate".

3864 • integer-abs

3865 • double-abs

3866 • round

3867 • floor

3868 **A14.3 String conversion functions**

3869 The following functions convert between values of the XACML
3870 "http://www.w3.org/2001/XMLSchema#string" primitive types. In an expression that contains any of
3871 these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3872 "Indeterminate".

- 3873 • string-normalize-space

3874 This function SHALL take one argument of type
3875 "http://www.w3.org/2001/XMLSchema#string" and SHALL normalize the value by stripping
3876 off all leading and trailing whitespace characters.

- 3877 • string-normalize-to-lower-case

3878 This function SHALL take one argument of "http://www.w3.org/2001/XMLSchema#string"
3879 and SHALL normalize the value by converting each upper case character to its lower case
3880 equivalent.

3881 **A14.4 Numeric type conversion functions**

3882 The following functions convert between the XACML
3883 "http://www.w3.org/2001/XMLSchema#integer" and "http://www.w3.org/2001/XMLSchema#double"
3884 primitive types. In any expression in which the functions defined below are applied, if any argument
3885 while being evaluated results in "Indeterminate", the expression SHALL return "Indeterminate".

- 3886 • double-to-integer

3887 This function SHALL take one argument of type
3888 "http://www.w3.org/2001/XMLSchema#double" and SHALL truncate its numeric value to a
3889 whole number and return an element of type
3890 "http://www.w3.org/2001/XMLSchema#integer".

- 3891 • integer-to-double

3892 This function SHALL take one argument of type
3893 "http://www.w3.org/2001/XMLSchema#integer" and SHALL promote its value to an element
3894 of type "http://www.w3.org/2001/XMLSchema#double" of the same numeric value.

3895 **A14.5 Logical functions**

3896 This section contains the specification for logical functions that operate on arguments of the
3897 "http://www.w3.org/2001/XMLSchema#boolean" type.

- 3898 • or

3899 This function SHALL return "False" if it has no arguments and SHALL return "True" if one of
3900 its arguments evaluates to "True". The order of evaluation SHALL be from first argument to
3901 last. The evaluation SHALL stop with a result of "True" if any argument evaluates to "True",
3902 leaving the rest of the arguments unevaluated. In an expression that contains any of these
3903 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3904 "Indeterminate".

- 3905 • and

3906 This function SHALL return "True" if it has no arguments and SHALL return "False" if one of
3907 its arguments evaluates to "False". The order of evaluation SHALL be from first argument
3908 to last. The evaluation SHALL stop with a result of "False" if any argument evaluates to

3909 "False", leaving the rest of the arguments unevaluated. In an expression that contains any
3910 of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate
3911 to "Indeterminate".

3912 • n-of

3913 The first argument to this function SHALL be of type
3914 "http://www.w3.org/2001/XMLSchema#integer", specifying the number of the remaining
3915 arguments that MUST evaluate to "True" for the expression to be considered "True". If the
3916 first argument is 0, the result SHALL be "True". If the number of arguments after the first
3917 one is less than the value of the first argument, then the expression SHALL result in
3918 "Indeterminate". The order of evaluation SHALL be: first evaluate the integer value, then
3919 evaluate each subsequent argument. The evaluation SHALL stop and return "True" if the
3920 specified number of arguments evaluate to "True". The evaluation of arguments SHALL
3921 stop if it is determined that evaluating the remaining arguments will not satisfy the
3922 requirement. In an expression that contains any of these functions, if any argument is
3923 "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

3924 • not

3925 This function SHALL take one logical argument. If the argument evaluates to "True", then
3926 the result of the expression SHALL be "False". If the argument evaluates to "False", then
3927 the result of the expression SHALL be "True". In an expression that contains any of these
3928 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3929 "Indeterminate".

3930 • present

3931 This function SHALL take an attribute value of type
3932 "http://www.w3.org/2001/XMLSchema#anyURI" as used as the `AttributeId` in an
3933 `<AttributeDesignator>` element. This expression SHALL return "True" if the named
3934 **attribute** can be located in the request **context**, which means that an
3935 `<AttributeDesignator>` or `<AttributeSelector>` element for this named **attribute**
3936 will return a **bag** containing at least one value. If it cannot be determined that the **attribute**
3937 is present in the request **context**, or the attribute cannot be evaluated, then the expression
3938 SHALL result in "Indeterminate".

3939 **A14.6 Arithmetic comparison functions**

3940 These functions form a minimal set for comparing two numbers, yielding a boolean result. They
3941 SHALL comply with the rules governed by IEEE 754 [IEEE 754]. In an expression that contains
3942 any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to
3943 "Indeterminate".

3944 • integer-greater-than

3945 • integer-greater-than-or-equal

3946 • integer-less-than

3947 • integer-less-than-or-equal

3948 • double-greater-than

3949 • double-greater-than-or-equal

3950 • double-less-than

3951 • double-less-than-or-equal

3952

A14.7 Date and time arithmetic functions

3953 These functions perform arithmetic operations with the date and time. In an expression that
3954 contains any of these functions, if any argument is "Indeterminate", then the expression SHALL
3955 evaluate to "Indeterminate".

- 3956 • `dateTime-add-dayTimeDuration`

3957 This function SHALL take two arguments, the first is of type
3958 "`http://www.w3.org/2001/XMLSchema#dateTime`" and the second is of type
3959 "`xf:dayTimeDuration`". It SHALL return a result of
3960 "`http://www.w3.org/2001/XMLSchema#dateTime`". This function SHALL return the value by
3961 adding the second argument to the first argument according to the specification of adding
3962 durations to date and time [XS Appendix E].

- 3963 • `dateTime-add-yearMonthDuration`

3964 This function SHALL take two arguments, the first is a
3965 "`http://www.w3.org/2001/XMLSchema#dateTime`" and the second is a
3966 "`xf:yearMonthDuration`". It SHALL return a result of
3967 "`http://www.w3.org/2001/XMLSchema#dateTime`". This function SHALL return the value by
3968 adding the second argument to the first argument according to the specification of adding
3969 durations to date and time [XS Appendix E].

- 3970 • `dateTime-subtract-dayTimeDuration`

3971 This function SHALL take two arguments, the first is a
3972 "`http://www.w3.org/2001/XMLSchema#dateTime`" and the second is a
3973 "`xf:dayTimeDuration`". It SHALL return a result of
3974 "`http://www.w3.org/2001/XMLSchema#dateTime`". If the second argument is a positive
3975 duration, then this function SHALL return the value by adding the corresponding negative
3976 duration, as per the specification [XS Appendix E]. If the second argument is a negative
3977 duration, then the result SHALL be as if the function "`function:dateTime-add-`
3978 `dayTimeDuration`" had been applied to the corresponding positive duration.

- 3979 • `dateTime-subtract-yearMonthDuration`

3980 This function SHALL take two arguments, the first is a
3981 "`http://www.w3.org/2001/XMLSchema#dateTime`" and the second is a
3982 "`xf:yearMonthDuration`". It SHALL return a result of
3983 "`http://www.w3.org/2001/XMLSchema#dateTime`". If the second argument is a positive
3984 duration, then this function SHALL return the value by adding the corresponding negative
3985 duration, as per the specification [XS Appendix E]. If the second argument is a negative
3986 duration, then the result SHALL be as if the function "`function:dateTime-add-`
3987 `yearMonthDuration`" had been applied to the corresponding positive duration.

- 3988 • `date-add-yearMonthDuration`

3989 This function SHALL take two arguments, the first is a
3990 "`http://www.w3.org/2001/XMLSchema#date`" and the second is a "`xf:yearMonthDuration`". It
3991 return a result of "`http://www.w3.org/2001/XMLSchema#date`". This function SHALL return
3992 the value by adding the second argument to the first argument according to the
3993 specification of adding durations to date [XS Appendix E].

- 3994 • `date-subtract-yearMonthDuration`

3995 This function SHALL take two arguments, the first is a
3996 "`http://www.w3.org/2001/XMLSchema#date`" and the second is a "`xf:yearMonthDuration`". It
3997 SHALL return a result of "`http://www.w3.org/2001/XMLSchema#date`". If the second

3998 argument is a positive duration, then this function SHALL return the value by adding the
3999 corresponding negative duration, as per the specification [XS Appendix E]. If the second
4000 argument is a negative duration, then the result SHALL be as if the function “function:date-
4001 add-yearMonthDuration” had been applied to the corresponding positive duration.

4002 **A14.8 Non-numeric comparison functions**

4003 These functions perform comparison operations on two arguments of non-numerical types. In an
4004 expression that contains any of these functions, if any argument is "Indeterminate", then the
4005 expression SHALL evaluate to "Indeterminate".

4006 • string-greater-than

4007 This function SHALL take two arguments of type
4008 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
4009 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if and only if the
4010 arguments are compared byte by byte and, after an initial prefix of corresponding bytes
4011 from both arguments that are considered equal by “function:integer-equal”, the next byte by
4012 byte comparison is such that the byte from the first argument is greater than the byte from
4013 the second argument by the use of the function “function:integer-equal”.

4014 • string-greater-than-or-equal

4015 This function SHALL take two arguments of type
4016 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
4017 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated
4018 with the logical function “function:or” with two arguments containing the functions
4019 “function:string-greater-than” and “function:string-equal” containing the original arguments

4020 • string-less-than

4021 This function SHALL take two arguments of type
4022 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
4023 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if and only if the
4024 arguments are compared byte by byte and, after an initial prefix of corresponding bytes
4025 from both arguments are considered equal by “function:integer-equal”, the next byte by
4026 byte comparison is such that the byte from the first argument is less than the byte from the
4027 second argument by the use of the function “function:integer-less-than”.

4028 • string-less-than-or-equal

4029 This function SHALL take two arguments of type
4030 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
4031 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated
4032 with the function “function:or” with two arguments containing the functions “function:string-
4033 less-than” and “function:string-equal” containing the original arguments.

4034 • time-greater-than

4035 This function SHALL take two arguments of type
4036 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an
4037 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
4038 argument is greater than the second argument according to the order relation specified for
4039 “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].

4040 • time-greater-than-or-equal

4041 This function SHALL take two arguments of type
4042 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an

4043 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
4044 argument is greater than or equal to the second argument according to the order relation
4045 specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

4046 • time-less-than

4047 This function SHALL take two arguments of type
4048 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
4049 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
4050 argument is less than the second argument according to the order relation specified for
4051 "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

4052 • time-less-than-or-equal

4053 This function SHALL take two arguments of type
4054 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
4055 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
4056 argument is less than or equal to the second argument according to the order relation
4057 specified for "http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

4058 • dateTime-greater-than

4059 This function SHALL take two arguments of type
4060 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
4061 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
4062 argument is greater than the second argument according to the order relation specified for
4063 "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

4064 • dateTime-greater-than-or-equal

4065 This function SHALL take two arguments of type
4066 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
4067 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
4068 argument is greater than or equal to the second argument according to the order relation
4069 specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

4070 • dateTime-less-than

4071 This function SHALL take two arguments of type
4072 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
4073 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
4074 argument is less than the second argument according to the order relation specified for
4075 "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

4076 • dateTime-less-than-or-equal

4077 This function SHALL take two arguments of type "http://www.w3.org/2001/XMLSchema#
4078 dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It
4079 SHALL return "True" if the first argument is less than or equal to the second argument
4080 according to the order relation specified for
4081 "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

4082 • date-greater-than

4083 This function SHALL take two arguments of type
4084 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4085 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
4086 argument is greater than the second argument according to the order relation specified for
4087 "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

4088 • date-greater-than-or-equal
4089 This function SHALL take two arguments of type
4090 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4091 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
4092 argument is greater than or equal to the second argument according to the order relation
4093 specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

4094 • date-less-than
4095 This function SHALL take two arguments of type
4096 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4097 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
4098 argument is less than the second argument according to the order relation specified for
4099 "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

4100 • date-less-than-or-equal
4101 This function SHALL take two arguments of type
4102 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4103 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
4104 argument is less than or equal to the second argument according to the order relation
4105 specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

4106 **A14.9 Bag functions**

4107 These functions operate on a **bag** of *type* values, where *type* is one of the primitive types. In an
4108 expression that contains any of these functions, if any argument is "Indeterminate", then the
4109 expression SHALL evaluate to "Indeterminate". Some additional conditions defined for each
4110 function below SHALL cause the expression to evaluate to "Indeterminate".

4111 • *type*-one-and-only
4112 This function SHALL take an argument of a **bag** of *type* values and SHALL return a value
4113 of *type*. It SHALL return the only value in the **bag**. If the **bag** does not have one and only
4114 one value, then the expression SHALL evaluate to "Indeterminate".

4115 • *type*-bag-size
4116 This function SHALL take a **bag** of *type* values as an argument and SHALL return an
4117 "http://www.w3.org/2001/XMLSchema#integer" indicating the number of values in the **bag**.

4118 • *type*-is-in
4119 This function SHALL take an argument of type *type* as the first argument and a **bag** of *type*
4120 values as the second argument. The expression SHALL evaluate to "True" if the first
4121 argument matches by the "function:type-equal" to any value in the **bag**.

4122 • *type*-bag
4123 This function SHALL take any number of arguments of a single type and return a **bag** of
4124 *type* values containing the values of the arguments. An application of this function to zero
4125 arguments SHALL produce an empty **bag** of the specified type.

4126 **A14.10 Set functions**

4127 These functions operate on **bags** mimicking sets by eliminating duplicate elements from a **bag**. In
4128 an expression that contains any of these functions, if any argument is "Indeterminate", then the
4129 expression SHALL evaluate to "Indeterminate".

4130 • *type-intersection*

4131 This function SHALL take two arguments that are both a **bag** of *type* values. The
4132 expression SHALL return a **bag** of *type* values such that it contains only elements that are
4133 common between the two **bags**, which is determined by "function:type-equal". No
4134 duplicates as determined by "function:type-equal" SHALL exist in the result.

4135 • *type-at-least-one-member-of*

4136 This function SHALL take two arguments that are both a **bag** of *type* values. The
4137 expression SHALL evaluate to "True" if at least one element of the first argument is
4138 contained in the second argument as determined by "function:type-is-in".

4139 • *type-union*

4140 This function SHALL take two arguments that are both a **bag** of *type* values. The
4141 expression SHALL return a **bag** of *type* such that it contains all elements of both **bags**. No
4142 duplicates as determined by "function:type-equal" SHALL exist in the result.

4143 • *type-subset*

4144 This function SHALL take two arguments that are both a **bag** of *type* values. It SHALL
4145 return "True" if the first argument is a subset of the second argument. Each argument is
4146 considered to have its duplicates removed as determined by "function:type-equal" before
4147 subset calculation.

4148 • *type-set-equals*

4149 This function SHALL take two arguments that are both a **bag** of *type* values and SHALL
4150 return the result of applying "function:and" to the application of "function:type-subset" to the
4151 first and second arguments and the application of "function:type-subset" to the second and
4152 first arguments.

4153 **A14.11 Higher-order bag functions**

4154 This section describes functions in XACML that perform operations on **bags** such that functions
4155 may be applied to the **bags** in general.

4156 In this section, a general-purpose functional language called Haskell [**Haskell**] is used to formally
4157 specify the semantics of these functions. Although the English description is adequate, a formal
4158 specification of the semantics is helpful.

4159 For a quick summary, in the following Haskell notation, a function definition takes the form of
4160 clauses that are applied to patterns of structures, namely lists. The symbol "[]" denotes the empty
4161 list, whereas the expression "(x:xs)" matches against an argument of a non-empty list of which "x"
4162 represents the first element of the list, and "xs" is the rest of the list, which may be an empty list. We
4163 use the Haskell notion of a list, which is an ordered collection of elements, to model the XACML
4164 **bags** of values.

4165 A simple Haskell definition of a familiar function "function:and" that takes a list of booleans is
4166 defined as follows:

4167 and:: [Bool] -> Bool

4168 and [] = "True"

4169 and (x:xs) = x && (and xs)

4170 The first definition line denoted by a ":" formally describes the type of the function, which takes a
4171 list of booleans, denoted by "[Bool]", and returns a boolean, denoted by "Bool". The second
4172 definition line is a clause that states that the function "and" applied to the empty list is "True". The
4173 second definition line is a clause that states that for a non-empty list, such that the first element is
4174 "x", which is a value of type Bool, the function "and" applied to x SHALL be combined with, using
4175 the logical conjunction function, which is denoted by the infix symbol "&&", the result of recursively
4176 applying the function "and" to the rest of the list. Of course, an application of the "and" function is
4177 "True" if and only if the list to which it is applied is empty or every element of the list is "True". For
4178 example, the evaluation of the following Haskell expressions,

4179 (and []), (and ["True"]), (and ["True", "True"]), (and ["True", "True", "False"])

4180 evaluate to "True", "True", "True", and "False", respectively.

4181 In an expression that contains any of these functions, if any argument is "Indeterminate", then the
4182 expression SHALL evaluate to "Indeterminate".

4183 • any-of

4184 This function applies a boolean function between a specific primitive value and a **bag** of
4185 values, and SHALL return "True" if and only if the predicate is "True" for at least one
4186 element of the **bag**.

4187 This function SHALL take three arguments. The first argument SHALL be a <Function>
4188 element that names a boolean function that takes two arguments of primitive types. The
4189 second argument SHALL be a value of a primitive type. The third argument SHALL be a
4190 **bag** of a primitive type. The expression SHALL be evaluated as if the function named in
4191 the <Function> element is applied to the second argument and each element of the third
4192 argumane (the **bag**) and the results are combined with "function:or".

4193 In Haskell, the semantics of this operation are as follows:

```
4194           any_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
4195           any_of f a []       = "False"
4196           any_of f a (x:xs) = (f a x) || (any_of f a xs)
```

4197 In the above notation, "f" is the function name to be applied, "a" is the primitive value, and
4198 "(x:xs)" represents the first element of the list as "x" and the rest of the list as "xs".

4199 For example, the following expression SHALL return "True":

```
4200       <Apply FunctionId="function:any-of">
4201        <Function FunctionId="function:string-equal" />
4202        <AttributeValue
4203        DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4204        <Apply FunctionId="function:string-bag">
4205        <AttributeValue
4206        DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4207        <AttributeValue
4208        DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4209        <AttributeValue
4210        DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4211        <AttributeValue
4212        DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4213        </Apply>
4214       </Apply>
```


4215 This expression is "True" because the first argument is equal to at least one of the
4216 elements of the **bag**.

4217 • all-of

4218 This function applies a boolean function between a specific primitive value and a **bag** of
4219 values, and returns "True" if and only if the predicate is "True" for every element of the **bag**.

4220 This function SHALL take three arguments. The first argument SHALL be a <Function>
4221 element that names a boolean function that takes two arguments of primitive types. The
4222 second argument SHALL be a value of a primitive type. The third argument SHALL be a
4223 **bag** of a primitive type. The expression SHALL be evaluated as if the function named in
4224 the <Function> element were applied to the second argument and each element of the
4225 third argument (the **bag**) and the results were combined using "function:and".

4226 In Haskell, the semantics of this operation are as follows:

```
4227 all_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
4228 all_of f a [] = "False"
4229 all_of f a (x:xs) = (f a x) && (all_of f a xs)
```

4230 In the above notation, "f" is the function name to be applied, "a" is the primitive value, and
4231 "(x:xs)" represents the first element of the list as "x" and the rest of the list as "xs".

4232 For example, the following expression SHALL evaluate to "True":

```
4233 <Apply FunctionId="function:all-of">
4234   <Function FunctionId="function:integer-greater" />
4235   <AttributeValue
4236     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4237   <Apply FunctionId="function:integer-bag">
4238     <AttributeValue
4239       DataType="http://www.w3.org/2001/XMLSchema#integer">9</AttributeValue>
4240     <AttributeValue
4241       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4242     <AttributeValue
4243       DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4244     <AttributeValue
4245       DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4246   </Apply>
4247 </Apply>
```

4248 This expression is "True" because the first argument is greater than *all* of the elements of
4249 the **bag**.

4250 • any-of-any

4251 This function applies a boolean function between each element of a **bag** of values and
4252 each element of another **bag** of values, and returns "True" if and only if the predicate is
4253 "True" for at least one comparison.

4254 This function SHALL take three arguments. The first argument SHALL be a <Function>
4255 element that names a boolean function that takes two arguments of primitive types. The
4256 second argument SHALL be a **bag** of a primitive type. The third argument SHALL be a
4257 **bag** of a primitive type. The expression SHALL be evaluated as if the function named in
4258 the <Function> element were applied between *every* element in the second argument
4259 and *every* element of the third argument (the **bag**) and the results were combined using
4260 "function:or". The semantics are that the result of the expression SHALL be "True" if and
4261 only if the applied predicate is "True" for *any* comparison of elements from the two **bags**.

4262 In Haskell, taking advantage of the “any_of” function defined above, the semantics of the
4263 “any_of_any” function are as follows:

```
4264 any_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4265 any_of_any f [] ys = "False"
4266 any_of_any f (x:xs) ys = (any_of f x ys) || (any_of_any f xs ys)
```

4267 In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first
4268 element of the list as “x” and the rest of the list as “xs”.

4269 For example, the following expression SHALL evaluate to "True":

```
4270 <Apply FunctionId="function:any-of-any">
4271   <Function FunctionId="function:string-equal"/>
4272   <Apply FunctionId="function:string-bag">
4273     <AttributeValue
4274       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4275     <AttributeValue
4276       DataType="http://www.w3.org/2001/XMLSchema#string">Mary</AttributeValue>
4277   </Apply>
4278   <Apply FunctionId="function:string-bag">
4279     <AttributeValue
4280       DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4281     <AttributeValue
4282       DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4283     <AttributeValue
4284       DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4285     <AttributeValue
4286       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4287   </Apply>
4288 </Apply>
```

4289 This expression is "True" because at least one of the elements of the first **bag**, namely
4290 “Ringo”, is equal to at least one of the string values of the second **bag**.

4291 • all-of-any

4292 This function applies a boolean function between the elements of two **bags**. The
4293 expression is "True" if and only if the predicate is "True" between each and all of the
4294 elements of the first **bag** collectively against at least one element of the second **bag**.

4295 This function SHALL take three arguments. The first argument SHALL be a <Function>
4296 element that names a boolean function that takes two arguments of primitive types. The
4297 second argument SHALL be a **bag** of a primitive type. The third argument SHALL be a
4298 **bag** of a primitive type. The expression SHALL be evaluated as if function named in the
4299 <Function> element were applied between every element in the second argument and
4300 every element of the third argument (the **bag**) using “function:and”. The semantics are that
4301 the result of the expression SHALL be "True" if and only if the applied predicate is "True"
4302 for each element of the first **bag** and any element of the second **bag**.

4303 In Haskell, taking advantage of the “any_of” function defined in Haskell above, the
4304 semantics of the “all_of_any” function are as follows:

```
4305 all_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4306 all_of_any f [] ys = "False"
4307 all_of_any f (x:xs) ys = (any_of f x ys) && (all_of_any f xs ys)
```

4308 In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first
4309 element of the list as “x” and the rest of the list as “xs”.

4310 For example, the following expression SHALL evaluate to "True":

```

4311 <Apply FunctionId="function:all-of-any">
4312   <Function FunctionId="function:integer-greater" />
4313   <Apply FunctionId="function:integer-bag">
4314     <AttributeValue
4315       DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4316     <AttributeValue
4317       DataType="http://www.w3.org/2001/XMLSchema#integer">20</AttributeValue>
4318   </Apply>
4319   <Apply FunctionId="function:integer-bag">
4320     <AttributeValue
4321       DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4322     <AttributeValue
4323       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4324     <AttributeValue
4325       DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4326     <AttributeValue
4327       DataType="http://www.w3.org/2001/XMLSchema#integer">21</AttributeValue>
4328   </Apply>
4329 </Apply>

```

4330 This expression is "True" because all of the elements of the first **bag**, each "10" and "20",
4331 are greater than at least one of the integer values "1", "3", "5", "21" of the second **bag**.

4332 • any-of-all

4333 This function applies a boolean function between the elements of two **bags**. The
4334 expression SHALL be "True" if and only if the predicate is "True" between at least one of
4335 the elements of the first **bag** collectively against all the elements of the second **bag**.

4336 This function SHALL take three arguments. The first argument SHALL be a <Function>
4337 element that names a boolean function that takes two arguments of primitive types. The
4338 second argument SHALL be a **bag** of a primitive type. The third argument SHALL be a
4339 **bag** of a primitive type. The expression SHALL be evaluated as if the function named in
4340 the <Function> element were applied between every element in the second argument
4341 and every element of the third argument (the **bag**) and the results were combined using
4342 "function:or". The semantics are that the result of the expression SHALL be "True" if and
4343 only if the applied predicate is "True" for any element of the first **bag** compared to all the
4344 elements of the second **bag**.

4345 In Haskell, taking advantage of the "all_of" function defined in Haskell above, the semantics
4346 of the "any_of_all" function are as follows:

```

4347 any_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4348 any_of_all f [] ys = "False"
4349 any_of_all f (x:xs) ys = (all_of f x ys) || ( any_of_all f xs ys)

```

4350 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4351 element of the list as "x" and the rest of the list as "xs".

4352 For example, the following expression SHALL evaluate to "True":

```

4353 <Apply FunctionId="function:any-of-all">
4354   <Function FunctionId="function:integer-greater"/>
4355   <Apply FunctionId="function:integer-bag">
4356     <AttributeValue
4357       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4358     <AttributeValue
4359       DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4360   </Apply>
4361   <Apply FunctionId="function:integer-bag">
4362     <AttributeValue
4363       DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4364     <AttributeValue
4365       DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4366     <AttributeValue
4367       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4368     <AttributeValue
4369       DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4370   </Apply>
4371 </Apply>

```

4372 This expression is "True" because at least one element of the first **bag**, namely "5", is
4373 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4374 • all-of-all

4375 This function applies a boolean function between the elements of two **bags**. The
4376 expression SHALL be "True" if and only if the predicate is "True" between each and all of
4377 the elements of the first **bag** collectively against all the elements of the second **bag**.

4378 This function SHALL take three arguments. The first argument SHALL be a <Function>
4379 element that names a boolean function that takes two arguments of primitive types. The
4380 second argument SHALL be a **bag** of a primitive type. The third argument SHALL be a
4381 **bag** of a primitive type. The expression is evaluated as if the function named in the
4382 <Function> element were applied between *every* element in the second argument and
4383 *every* element of the third argument (the **bag**) and the results were combined using
4384 "function:and". The semantics are that the result of the expression is "True" if and only if
4385 the applied predicate is "True" for *all* elements of the first **bag** compared to *all* the elements
4386 of the second **bag**.

4387 In Haskell, taking advantage of the "all_of" function defined in Haskell above, the semantics
4388 of the "all_of_all" function is as follows:

```

4389 all_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4390 all_of_all f [] ys = "False"
4391 all_of_all f (x:xs) ys = (all_of f x ys) && (all_of_all f xs ys)

```

4392 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4393 element of the list as "x" and the rest of the list as "xs".

4394 For example, the following expression SHALL evaluate to "True":

```

4395 <Apply FunctionId="function:all-of-all">
4396   <Function FunctionId="function:integer-greater"/>
4397   <Apply FunctionId="function:integer-bag">
4398     <AttributeValue
4399       DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>
4400     <AttributeValue
4401       DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4402   </Apply>
4403   <Apply FunctionId="function:integer-bag">
4404     <AttributeValue
4405       DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4406     <AttributeValue
4407       DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4408     <AttributeValue
4409       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4410     <AttributeValue
4411       DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4412   </Apply>
4413 </Apply>

```

4414 This expression is "True" because all elements of the first **bag**, "5" and "6", are each
4415 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4416 • map

4417 This function converts a **bag** of values to another **bag** of values.

4418 This function SHALL take two arguments. The first function SHALL be a <Function>
4419 element naming a function that takes a single argument of a primitive type and returns a
4420 value of a primitive type. The second argument SHALL be a **bag** of a primitive type. The
4421 expression SHALL be evaluated as if the function named in the <Function> element were
4422 applied to each element in the **bag** resulting in a **bag** of the converted value. The result
4423 SHALL be a **bag** of the primitive type that is the same type that is returned by the function
4424 named in the <Function> element.

4425 In Haskell, this function is defined as follows:

```

4426     map:: (a -> b) -> [a] -> [b]
4427     map f []     = []
4428     map f (x:xs) = (f x) : (map f xs)

```

4429 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4430 element of the list as "x" and the rest of the list as "xs".

4431 For example, the following expression,

```

4432 <Apply FunctionId="function:map">
4433   <Function FunctionId="function:string-normalize-to-lower-case">
4434   <Apply FunctionId="function:string-bag">
4435     <AttributeValue
4436       DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>
4437     <AttributeValue
4438       DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>
4439   </Apply>
4440 </Apply>

```

4441 evaluates to a **bag** containing "hello" and "world!".

4442 **A14.12 Special match functions**

4443 These functions operate on various types and evaluate to
4444 "http://www.w3.org/2001/XMLSchema#boolean" based on the specified standard matching
4445 algorithm. In an expression that contains any of these functions, if any argument is "Indeterminate",
4446 then the expression SHALL evaluate to "Indeterminate".

4447 • regex-string-match

4448 This function decides a regular expression match. It SHALL take two arguments of
4449 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4450 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4451 expression and the second argument SHALL be a general string. The function
4452 specification SHALL be that of the "xf:match" function with the arguments reversed [XF
4453 Section 6.3.15.1].

4454 • x500Name-match

4455 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-
4456 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It
4457 shall return "True" if and only if the first argument matches some terminal sequence of
4458 RDNs from the second argument when compared using x500Name-equal.

4459 • rfc822Name-match

4460 This function SHALL take two arguments, the first is of type
4461 "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4462 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an
4463 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if
4464 the first argument matches the second argument according to the following specification.

4465 An RFC822 name consists of a local-part followed by "@" followed by domain-part. The
4466 local-part is case-sensitive, while the domain-part (which is usually a DNS name) is not
4467 case-sensitive.⁴

4468 The second argument contains a complete rfc822Name. The first argument is a complete
4469 or partial rfc822Name used to select appropriate values in the second argument as follows.

4470 In order to match a particular mailbox in the second argument, the first argument must
4471 specify the complete mail address to be matched. For example, if the first argument is
4472 "Anderson@sun.com", this matches a value in the second argument of
4473 "Anderson@sun.com" and "Anderson@SUN.COM", but not "Anne.Anderson@sun.com",
4474 "anderson@sun.com" or "Anderson@east.sun.com".

4475 In order to match any mail address at a particular domain in the second argument, the first
4476 argument must specify only a domain name (usually a DNS name). For example, if the first
4477 argument is "sun.com", this matches a value in the first argument of "Anderson@sun.com?"
4478 or "Baxter@SUN.COM", but not "Anderson@east.sun.com".

4479 In order to match any mail address in a particular domain in the second argument, the first
4480 argument must specify the desired domain-part with a leading ".". For example, if the first
4481 argument is ".east.sun.com", this matches a value in the second argument of

4 According to IETF RFC822 and its successor specifications [RFC2821], case is significant in the *local-part*. Many mail systems, as well as the IETF PKIX specification, treat the *local-part* as case-insensitive. This anomaly is considered an error by mail-system designers and is not encouraged. For this reason, rfc822Name-match treats *local-part* as case sensitive.

4482 "Anderson@east.sun.com" and "anne.anderson@ISRG.EAST.SUN.COM" but not
4483 "Anderson@sun.com".

4484 **A14.13 XPath-based functions**

4485 This section specifies functions that take XPath expressions for arguments. An XPath expression
4486 evaluates to a *node-set*, which is a set of XML nodes that match the expression. A node or node-
4487 set is not in the formal type system of XACML. All comparison or other operations on node-sets are
4488 performed in the isolation of the particular function specified. The XPath expressions in these
4489 functions are restricted to the XACML request *context*. The following functions are defined:

- 4490 • xpath-node-count

4491 This function SHALL take an "http://www.w3.org/2001/XMLSchema#string" as an
4492 argument, which SHALL be interpreted as an XPath expression, and evaluates to an
4493 "http://www.w3.org/2001/XMLSchema#integer". The value returned from the function
4494 SHALL be the count of the nodes within the node-set that matches the given XPath
4495 expression.

- 4496 • xpath-node-equal

4497 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,
4498 which SHALL be interpreted as XPath expressions, and SHALL return an
4499 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if any
4500 XML node from the node-set matched by the first argument equals according to the
4501 "op:node-equal" function [XQO] any XML node from the node-set matched by the second
4502 argument.

- 4503 • xpath-node-match

4504 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,
4505 which SHALL be interpreted as XPath expressions and SHALL return an
4506 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL first extend the first
4507 argument to match an XML document in a hierarchical fashion. If *a* is an XPath expression
4508 and it is specified as the first argument, it SHALL be interpreted to mean match the set of
4509 nodes specified by the enhanced XPath expression "*a* | *a*/* | *a*/*@*". In other words, the
4510 expression *a* SHALL match all elements and attributes below the element specified by *a*.
4511 This function SHALL evaluate to "True" if any XML node that matches the enhanced XPath
4512 expression is equal according to "op:node-equal" [XQO] to any XML node from the node-
4513 set matched by the second argument.

4514 **A14.14 Extension functions and primitive types**

4515 Functions and primitive types are specified by string identifiers allowing for the introduction of
4516 functions in addition to those specified by XACML. This approach allows one to extend the XACML
4517 module with special functions and special primitive data types.

4518 In order to preserve some integrity to the XACML evaluation strategy, the result of all function
4519 applications SHALL depend only on the values of its arguments. Global and hidden parameters
4520 SHALL NOT affect the evaluation of an expression. Functions SHALL NOT have side effects, as
4521 evaluation order cannot be guaranteed in a standard way.

4522 Appendix B. XACML identifiers (normative)

4523 This section defines standard identifiers for commonly used entities. All XACML-defined identifiers
4524 have the common base:

4525 `urn:oasis:names:tc:xacml:1.0`

4526 B.1. XACML namespaces

4527 There are currently two defined XACML namespaces.

4528 Policies are defined using this identifier.

4529 `urn:oasis:names:tc:xacml:1.0:policy`

4530 Request and response *contexts* are defined using this identifier.

4531 `urn:oasis:names:tc:xacml:1.0:context`

4532 XACML data-types are defined using this identifier.

4533 `urn:oasis:names:tc:xacml:1.0:data-type`

4534 B.2. Access subject categories

4535 This identifier indicates the system entity that is directly requesting **access**. That is, the final entity
4536 in a request chain. If **subject** category is not specified, this is the default value.

4537 `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

4538 This identifier indicates the system entity that will receive the results of the request. Used when it is
4539 distinct from the access-subject.

4540 `urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject`

4541 This identifier indicates a system entity through which the **access** request was passed. There may
4542 be more than one. No means is provided to specify the order in which they passed the message.

4543 `urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject`

4544 This identifier indicates a system entity associated with a local or remote codebase that generated
4545 the request. Corresponding **subject attributes** might include the URL from which it was loaded
4546 and/or the identity of the code-signer. There may be more than one. No means is provided to
4547 specify the order they processed the request.

4548 `urn:oasis:names:tc:xacml:1.0:subject-category:codebase`

4549 This identifier indicates a system entity associated with the computer that initiated the **access**
4550 request. An example would be an IPsec identity.

4551 `urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine`

4552 B.3. XACML functions

4553 This identifier is the base for all the identifiers in the table of functions. See Section A.1.

4554 urn:oasis:names:tc:xacml:1.0:function

4555 B.4. Data types

4556 The following identifiers indicate useful data-types.

4557 X.500 distinguished name

4558 urn:oasis:names:tc:xacml:1.0:data-type:x500Name

4559 An x500Name contains an ITU-T Rec. X.520 Distinguished Name. The valid syntax for such a
4560 name is described in IETF RFC 2253 "Lightweight Directory Access Protocol (v3): UTF-8 String
4561 Representation of Distinguished Names".

4562 RFC822 Name

4563 urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name

4564 An rfc822Name contains an "e-mail name". The valid syntax for such a name is described in IETF
4565 RFC 2821, Section 4.1.2, Command Argument Syntax, under the term "Mailbox".

4566 The following data type identifiers are defined by XML Schema and XQuery.

4567 <http://www.w3.org/2001/XMLSchema:string>
4568 <http://www.w3.org/2001/XMLSchema:boolean>
4569 <http://www.w3.org/2001/XMLSchema:integer>
4570 <http://www.w3.org/2001/XMLSchema:double>
4571 <http://www.w3.org/2001/XMLSchema:date>
4572 <http://www.w3.org/2001/XMLSchema:dateTime>
4573 <http://www.w3.org/2001/XMLSchema:anyURI>
4574 <http://www.w3.org/2001/XMLSchema:hexBinary>
4575 <http://www.w3.org/2001/XMLSchema:base64Binary>
4576 <http://www.w3.org/2002/08/xquery-functions:dayTimeDuration>
4577 <http://www.w3.org/2002/08/xquery-functions:yearMonthDuration>

4578 B.5. Subject attributes

4579 These identifiers indicate *attributes* of a *subject*. When used, they SHALL appear within a
4580 <Subject> element of the request *context*. They SHALL be accessed via a
4581 <SubjectAttributeDesignator>, a <QualifiedSubjectAttributeDesignator> or an
4582 <AttributeSelector> element pointing into a <Subject> element of the request *context*.

4583 At most one of each of these attributes is associated with each subject. Each attribute associated
4584 with authentication included within a single <Subject> element relates to the same authentication
4585 event.

4586 This identifier indicates the name of the *subject*. The default format is
4587 <http://www.w3.org/2001/XMLSchema:string>. To indicate other formats, use `Data Type` attributes
4588 listed in B.4

4589 urn:oasis:names:tc:xacml:1.0:subject:subject-id

4590 This identifier indicates the *subject* category. "access-subject" is the default.

4591 urn:oasis:names:tc:xacml:1.0:subject:subject-category

4592 This identifier indicates the security domain of the *subject*. It identifies the administrator and policy
4593 that manages the name-space in which the *subject* id is administered.

| | |
|------|--|
| 4594 | urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier |
| 4595 | This identifier indicates a public key used to confirm the subject's identity. |
| 4596 | urn:oasis:names:tc:xacml:1.0:subject:key-info |
| 4597 | This identifier indicates the time at which the subject was authenticated. |
| 4598 | urn:oasis:names:tc:xacml:1.0:subject:authentication-time |
| 4599 | This identifier indicates the method used to authenticate the subject . |
| 4600 | urn:oasis:names:tc:xacml:1.0:subject:authentication-method |
| 4601 | This identifier indicates the time at which the subject initiated the access request, according to the |
| 4602 | PEP . |
| 4603 | urn:oasis:names:tc:xacml:1.0:subject:request-time |
| 4604 | This identifier indicates the time at which the subject's current session began, according to the |
| 4605 | PEP . |
| 4606 | urn:oasis:names:tc:xacml:1.0:subject:session-start-time |
| 4607 | The following identifiers indicate the location where authentication credentials were activated. They |
| 4608 | are intended to support the corresponding entities from the SAML authentication statement. |
| 4609 | This identifier indicates that the location is expressed as an IP address. |
| 4610 | urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address |
| 4611 | This identifier indicates that the location is expressed as a DNS name. |
| 4612 | urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name |
| 4613 | Where a suitable attribute is already defined in LDAP [LDAP-1, LDAP-2], the XACML identifier |
| 4614 | SHALL be formed by adding the attribute name to the URI of the LDAP specification. For |
| 4615 | example, the attribute name for the userPassword defined in the rfc2256 SHALL be: |
| 4616 | http://www.ietf.org/rfc/rfc2256.txt#userPassword |

4617 B.6. Resource attributes

4618 These identifiers indicate **attributes** of the **resource**. When used, they SHALL appear within the
4619 <Resource> element of the request **context**. They SHALL be accessed via a
4620 <ResourceAttributeDesignator> or an <AttributeSelector> element pointing into the
4621 <Resource> element of the request **context**.

4622 This identifier indicates the entire URI of the **resource**.

4623 urn:oasis:names:tc:xacml:1.0:resource:resource-id

4624 A **resource attribute** used to indicate values extracted from the **resource**.

4625 urn:oasis:names:tc:xacml:1.0:resource:resource-content

4626 This identifier indicates the last (rightmost) component of the file name. For example, if the URI is:
4627 "file://home/my/status#pointer", the simple-file-name is "status".

4628 urn:oasis:names:tc:xacml:1.0:resource:simple-file-name

4629 This identifier indicates that the **resource** is specified by an XPath expression.

4630 urn:oasis:names:tc:xacml:1.0:resource:xpath

| | |
|------|--|
| 4631 | This identifier indicates a UNIX file-system path. |
| 4632 | <code>urn:oasis:names:tc:xacml:1.0:resource:ufs-path</code> |
| 4633 | This identifier indicates the scope of the resource , as described in Section 7.8. |
| 4634 | <code>urn:oasis:names:tc:xacml:1.0:resource:scope</code> |
| 4635 | The allowed value for this attribute is of type <code>http://www.w3.org/2001/XMLSchema#string</code> , and is either "Immediate", "Children" or "Descendants". |
| 4636 | |

4637 B.7. Action attributes

| | |
|------|---|
| 4638 | These identifiers indicate attributes of the action being requested. When used, they SHALL appear within the <Action> element of the request context . They SHALL be accessed via an |
| 4639 | <ActionAttributeDesignator> or an <AttributeSelector> element pointing into the |
| 4640 | <Action> element of the request context . |
| 4641 | |
| 4642 | <code>urn:oasis:names:tc:xacml:1.0:action:action-id</code> |
| 4643 | Action namespace |
| 4644 | <code>urn:oasis:names:tc:xacml:1.0:action:action-namespace</code> |
| 4645 | Implied action. This is the value for action-id attribute when action is implied. |
| 4646 | <code>urn:oasis:names:tc:xacml:1.0:action:implied-action</code> |

4647 B.8. Environment attributes

| | |
|------|--|
| 4648 | These identifiers indicate attributes of the environment within which the decision request is to be |
| 4649 | evaluated. When used, they SHALL appear within the <Resource> element of the request |
| 4650 | context . They SHALL be accessed via an <EnvironmentAttributeDesignator> or an |
| 4651 | <AttributeSelector> element pointing into the <Environment> element of the request |
| 4652 | context . |
| 4653 | This identifier indicates the current time at the PDP . In practice it is the time at which the request |
| 4654 | context was created. |
| 4655 | <code>urn:oasis:names:tc:xacml:1.0:environment:current-time</code> |
| 4656 | <code>urn:oasis:names:tc:xacml:1.0:environment:current-date</code> |
| 4657 | <code>urn:oasis:names:tc:xacml:1.0:environment:current-dateTime</code> |

4658 B.9. Status codes

| | |
|------|---|
| 4659 | The following status code identifiers are defined. |
| 4660 | This identifier indicates success. |
| 4661 | <code>urn:oasis:names:tc:xacml:1.0:status:ok</code> |
| 4662 | This identifier indicates that attributes necessary to make a policy decision were not available. |
| 4663 | <code>urn:oasis:names:tc:xacml:1.0:status:missing-attribute</code> |
| 4664 | This identifier indicates that some attribute value contained a syntax error, such as a letter in a |
| 4665 | numeric field. |

| | |
|--------------|--|
| 4666 | <code>urn:oasis:names:tc:xacml:1.0:status:syntax-error</code> |
| 4667 4668 | This identifier indicates that an error occurred during policy evaluation. An example would be division by zero. |
| 4669 | <code>urn:oasis:names:tc:xacml:1.0:status:processing-error</code> |

4670 **B.10. Combining algorithms**

| | |
|--------------|---|
| 4671 | The deny-overrides rule-combining algorithm has the following value for <code>ruleCombiningAlgId</code> : |
| 4672 | <code>urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides</code> |
| 4673 4674 | The deny-overrides policy-combining algorithm has the following value for <code>policyCombiningAlgId</code> : |
| 4675 | <code>urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides</code> |
| 4676 | The permit-overrides rule-combining algorithm has the following value for <code>ruleCombiningAlgId</code> : |
| 4677 | <code>urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides</code> |
| 4678 4679 | The permit-overrides policy-combining algorithm has the following value for <code>policyCombiningAlgId</code> : |
| 4680 | <code>urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides</code> |
| 4681 | The first-applicable rule-combining algorithm has the following value for <code>ruleCombiningAlgId</code> : |
| 4682 | <code>urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable</code> |
| 4683 4684 | The first-applicable policy-combining algorithm has the following value for <code>policyCombiningAlgId</code> : |
| 4685 | <code>urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable</code> |
| 4686 4687 | The only-one-applicable-policy policy-combining algorithm has the following value for <code>policyCombiningAlgId</code> : |
| 4688 | <code>urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable-policy</code> |

4689

Appendix C. Combining algorithms (normative)

4690

This section contains a description of the rule-combining and policy-combining algorithms specified by XACML.

4691

4692

C.1. Deny-overrides

4693

The following specification defines the “Deny-overrides” *rule-combining algorithm* of a *policy*.

4694

4695

4696

4697

4698

4699

In the entire set of *rules* in the *policy*, if any *rule* evaluates to "Deny", then the result of the *rule* combination SHALL be "Deny". If any *rule* evaluates to "Permit" and all other *rules* evaluate to "Not-applicable", then the result of the *rule* combination SHALL be "Permit". In other words, "Deny" takes precedence, regardless of the result of evaluating any of the other *rules* in the combination. If all *rules* are found to be "Not-applicable" to the *decision request*, then the *rule* combination SHALL evaluate to "Not-applicable".

4700

4701

4702

4703

If an error occurs while evaluating the *target* or *condition* of a *rule* that contains an *effect* value of "Deny" then the evaluation SHALL continue to evaluate subsequent *rules*, looking for a result of "Deny". If no other *rule* evaluates to "Deny", then the combination SHALL evaluate to "Indeterminate".

4704

4705

4706

If at least one *rule* evaluates to "Permit", all other *rules* that do not have evaluation errors evaluate to "Permit" or "Not-applicable" and all *rules* that do have evaluation errors contain *effects* of "Permit", then the result of the combination SHALL be "Permit".

4707

The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

4708

4709

4710

4711

4712

4713

4714

4715

4716

4717

4718

4719

4720

4721

4722

4723

4724

4725

4726

4727

4728

4729

4730

4731

4732

4733

4734

4735

4736

4737

```
Decision denyOverridesRuleCombiningAlgorithm(Rule rule[])
{
  Boolean atLeastOneError = false;
  Boolean potentialDeny = false;
  Boolean atLeastOnePermit = false;
  for( i=0 ; i < lengthOf(rules) ; i++ )
  {
    Decision decision = evaluate(rule[i]);
    if (decision == Deny)
    {
      return Deny;
    }
    if (decision == Permit)
    {
      atLeastOnePermit = true;
      continue;
    }
    if (decision == Not-applicable)
    {
      continue;
    }
    if (decision == Indeterminate)
    {
      atLeastOneError = true;

      if (effect(rule[i]) == Deny)
      {
        potentialDeny = true;
      }
    }
    continue;
  }
}
```

```

4738     }
4739   }
4740   if (potentialDeny)
4741   {
4742     return Indeterminate;
4743   }
4744   if (atLeastOnePermit)
4745   {
4746     return Permit;
4747   }
4748   if (atLeastOneError)
4749   {
4750     return Indeterminate;
4751   }
4752   return Not-applicable;
4753 }

```

4754 The following specification defines the “Deny-overrides” *policy-combining algorithm* of a *policy*
4755 *set*.

4756 In the entire set of *policies* in the *policy set*, if any *policy* evaluates to "Deny", then the
4757 result of the *policy* combination SHALL be "Deny". In other words, "Deny" takes
4758 precedence, regardless of the result of evaluating any of the other *policies* in the *policy*
4759 *set*. If all *policies* are found to be "Not-applicable" to the *decision request*, then the
4760 *policy set* SHALL evaluate to "Not-applicable".

4761 If an error occurs while evaluating the *target* of a *policy*, or a reference to a *policy* is
4762 considered invalid or the *policy* evaluation results in "Indeterminate", then the *policy set*
4763 SHALL evaluate to "Deny".

4764 The following pseudo-code represents the evaluation strategy of this *policy-combining algorithm*.

```

4765 Decision denyOverridesPolicyCombiningAlgorithm(Policy policy[])
4766 {
4767   Boolean atLeastOnePermit = false;
4768   for( i=0 ; i < lengthOf(policy) ; i++ )
4769   {
4770     Decision decision = evaluate(policy[i]);
4771     if (decision == Deny)
4772     {
4773       return Deny;
4774     }
4775     if (decision == Permit)
4776     {
4777       atLeastOnePermit = true;
4778       continue;
4779     }
4780     if (decision == Not-applicable)
4781     {
4782       continue;
4783     }
4784     if (decision == Indeterminate)
4785     {
4786       return Deny;
4787     }
4788   }
4789   if (atLeastOnePermit)
4790   {
4791     return Permit;
4792   }
4793   return Not-applicable;
4794 }

```

4795 **Obligations** of the individual **policies** shall be combined as described in Section 3.3.2.3.

4796 C.2. Permit-overrides

4797 The following specification defines the "Permit-overrides" **rule-combining algorithm** of a **policy**.

4798 In the entire set of **rules** in the **policy**, if any **rule** evaluates to "Permit", then the result of
4799 the **rule** combination SHALL be "Permit". If any **rule** evaluates to "Deny" and all other
4800 **rules** evaluate to "Not-applicable", then the **policy** SHALL evaluate to "Deny". In other
4801 words, "Permit" takes precedence, regardless of the result of evaluating any of the other
4802 **rules** in the **policy**. If all **rules** are found to be "Not-applicable" to the **decision request**,
4803 then the **policy** SHALL evaluate to "Not-applicable".

4804 If an error occurs while evaluating the **target** or **condition** of a **rule** that contains an **effect**
4805 of "Permit" then the evaluation SHALL continue looking for a result of "Permit". If no other
4806 **rule** evaluates to "Permit", then the **policy** SHALL evaluate to "Indeterminate".

4807 If at least one **rule** evaluates to "Deny", all other **rules** that do not have evaluation errors
4808 evaluate to "Deny" or "Not-applicable" and all **rules** that do have evaluation errors contain
4809 an **effect** value of "Deny", then the **policy** SHALL evaluate to "Deny".

4810 The following pseudo-code represents the evaluation strategy of this **rule-combining algorithm**.

```
4811 Decision permitOverridesRuleCombiningAlgorithm(Rule rule[])
4812 {
4813     Boolean atLeastOneError = false;
4814     Boolean potentialPermit = false;
4815     Boolean atLeastOneDeny = false;
4816     for( i=0 ; i < lengthOf(rule) ; i++ )
4817     {
4818         Decision decision = evaluate(rule[i]);
4819         if (decision == Deny)
4820         {
4821             atLeastOneDeny = true;
4822             continue;
4823         }
4824         if (decision == Permit)
4825         {
4826             return Permit;
4827         }
4828         if (decision == Not-applicable)
4829         {
4830             continue;
4831         }
4832         if (decision == Indeterminate)
4833         {
4834             atLeastOneError = true;
4835
4836             if (effect(rule[i]) == Permit)
4837             {
4838                 potentialPermit = true;
4839             }
4840             continue;
4841         }
4842     }
4843     if (potentialPermit)
4844     {
4845         return Indeterminate;
4846     }
```

```

4847     if (atLeastOneDeny)
4848     {
4849         return Deny;
4850     }
4851     if (atLeastOneError)
4852     {
4853         return Indeterminate;
4854     }
4855     return Not-applicable;
4856 }

```

4857 The following specification defines the "Permit-overrides" *policy-combining algorithm* of a *policy*
4858 *set*.

4859 In the entire set of *policies* in the *policy set*, if any *policy* evaluates to "Permit", then the
4860 result of the *policy* combination SHALL be "Permit". In other words, "Permit" takes
4861 precedence, regardless of the result of evaluating any of the other *policies* in the *policy*
4862 *set*. If all *policies* are found to be "Not-applicable" to the *decision request*, then the
4863 *policy set* SHALL evaluate to "Not-applicable".

4864 If an error occurs while evaluating the *target* of a *policy*, a reference to a *policy* is
4865 considered invalid or the *policy* evaluation results in "Indeterminate", then the *policy set*
4866 SHALL evaluate to "Indeterminate" provided no other *policies* evaluate to "Permit" or
4867 "Deny".

4868 The following pseudo-code represents the evaluation strategy of this *policy-combining algorithm*.

```

4869 Decision permitOverridesPolicyCombiningAlgorithm(Policy policy[])
4870 {
4871     Boolean atLeastOneError = false;
4872     Boolean atLeastOneDeny = false;
4873     for( i=0 ; i < lengthOf(policy) ; i++ )
4874     {
4875         Decision decision = evaluate(policy[i]);
4876         if (decision == Deny)
4877         {
4878             atLeastOneDeny = true;
4879             continue;
4880         }
4881         if (decision == Permit)
4882         {
4883             return Permit;
4884         }
4885         if (decision == Not-applicable)
4886         {
4887             continue;
4888         }
4889         if (decision == Indeterminate)
4890         {
4891             atLeastOneError = true;
4892             continue;
4893         }
4894     }
4895     if (atLeastOneDeny)
4896     {
4897         return Deny;
4898     }
4899     if (atLeastOneError)
4900     {
4901         return Indeterminate;
4902     }
4903     return Not-applicable;

```


4904

```
}
```

4905 **Obligations** of the individual policies shall be combined as described in Section 3.3.2.3.

4906 C.3. First-applicable

4907 The following specification defines the "First-Applicable " **rule-combining algorithm** of a **policy**.

4908 Each **rule** SHALL be evaluated in the order in which it is listed in the **policy**. For a
4909 particular **rule**, if the **target** matches and the **condition** evaluates to "True", then the
4910 evaluation of the **policy** SHALL halt and the corresponding **effect** of the **rule** SHALL be the
4911 result of the evaluation of the **policy** (i.e. "Permit" or "Deny"). For a particular **rule** selected
4912 in the evaluation, if the **target** evaluates to "False" or the **condition** evaluates to "False",
4913 then the next **rule** in the order SHALL be evaluated. If no further **rule** in the order exists,
4914 then the **policy** SHALL evaluate to "Not-applicable".

4915 If an error occurs while evaluating the **target** or **condition** of a **rule**, then the evaluation
4916 SHALL halt, and the **policy** shall evaluate to "Indeterminate", with the appropriate error
4917 status.

4918 The following pseudo-code represents the evaluation strategy of this **rule-combining algorithm**.

```
4919 Decision firstApplicableEffectRuleCombiningAlgorithm(Rule rule[])
4920 {
4921     for( i = 0 ; i < lengthOf(rule) ; i++ )
4922     {
4923         Decision decision = evaluate(rule[i]);
4924         if (decision == Deny)
4925         {
4926             return Deny;
4927         }
4928         if (decision == Permit)
4929         {
4930             return Permit;
4931         }
4932         if (decision == Not-applicable)
4933         {
4934             continue;
4935         }
4936         if (decision == Indeterminate)
4937         {
4938             return Indeterminate;
4939         }
4940     }
4941     return Not-applicable;
4942 }
```

4943 The following specification defines the "First-applicable" **policy-combining algorithm** of a **policy**
4944 **set**.

4945 Each **policy** is evaluated in the order that it appears in the **policy set**. For a particular
4946 **policy**, if the **target** evaluates to "True" and the **policy** evaluates to a determinate value of
4947 "Permit" or "Deny", then the evaluation SHALL halt and the **policy set** SHALL evaluate to
4948 the **effect** value of that **policy**. For a particular **policy**, if the **target** evaluate to "False", or
4949 the **policy** evaluates to "Not-applicable", then the next **policy** in the order SHALL be
4950 evaluated. If no further **policy** exists in the order, then the **policy set** SHALL evaluate to
4951 "Not-applicable".

4952 If an error occurs while evaluating the **target** or the **policy**, or a reference to a **policy** is
4953 considered invalid, then the evaluation SHALL continue looking for an **applicable policy**, if
4954 no **applicable policy** is found, then the **policy set** SHALL evaluate to "Indeterminate".

4955 If an error were to occur when evaluating the **target**, or when evaluating a specific **policy**,
4956 the reference to the **policy** is considered invalid, or the **policy** itself evaluates to
4957 "Indeterminate", then the evaluation of the **policy-combining algorithm** shall halt, and the
4958 **policy set** shall evaluate to "Indeterminate" **with an appropriate error status**.

4959 The following pseudo-code represents the evaluation strategy of this **policy-combination**
4960 **algorithm**.

```
4961 Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy policy[])
4962 {
4963     for( i = 0 ; i < lengthOf(policy) ; i++ )
4964     {
4965         Decision decision = evaluate(policy[i]);
4966         if(decision == Deny)
4967         {
4968             return Deny;
4969         }
4970         if(decision == Permit)
4971         {
4972             return Permit;
4973         }
4974         if (decision == Not-applicable)
4975         {
4976             continue;
4977         }
4978         if (decision == Indeterminate)
4979         {
4980             return Indeterminate;
4981         }
4982     }
4983     return Not-applicable;
4984 }
```

4985 **Obligations** of the individual policies shall be combined as described in Section 3.3.2.3

4986 C.4. Only-one-applicable

4987 The following specification defines the "Only-one-applicable" **policy-combining algorithm** of a
4988 **policy set**.

4989 In the entire set of policies in the **policy set**, if no **policy** is considered applicable by virtue of their
4990 **targets**, then the result of the policy combination algorithm SHALL be "Not-applicable". If more than
4991 one policy is considered applicable by virtue of their **targets**, then the result of the policy
4992 combination algorithm SHALL be "Indeterminate".

4993 If only one **policy** is considered applicable by evaluation of the **policy targets**, then the result of
4994 the **policy-combining algorithm** SHALL be the result of evaluating the **policy**.

4995 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is considered
4996 invalid or the **policy** evaluation results in "Indeterminate", then the **policy set** SHALL evaluate to
4997 "Indeterminate".

4998 The following pseudo-code represents the evaluation strategy of this policy combining algorithm.

```
4999 Decision onlyOneApplicablePolicyPolicyCombiningAlogrithm(Policy policy[])
```

```
5000 {
5001     Boolean          atLeastOne      = false;
5002     Policy           selectedPolicy = null;
5003     ApplicableResult appResult;
5004
5005     for ( i = 0; i < lengthOf(policy) ; i++ )
5006     {
5007         appResult = isApplicable(policy[i]);
5008
5009         if ( appResult == Indeterminate )
5010         {
5011             return Indeterminate;
5012         }
5013         if( appResult == Applicable )
5014         {
5015             if ( atLeastOne )
5016             {
5017                 return Indeterminate;
5018             }
5019             else
5020             {
5021                 atLeastOne      = true;
5022                 selectedPolicy = policy[i];
5023             }
5024         }
5025         if ( appResult == NotApplicable )
5026         {
5027             continue;
5028         }
5029     }
5030     if ( atLeastOne )
5031     {
5032         return evaluate(selectedPolicy);
5033     }
5034     else
5035     {
5036         return NotApplicable;
5037     }
5038 }
```

5039

5040

Appendix D. Acknowledgments

5041 The following individuals were voting members of the XACML committee at the time that this
5042 version of the specification was issued:

5043 Affinitex James MacLean JMaclean@affinitex.com
5044 CrossLogix Ken Yagen kyagen@crosslogix.com
5045 CrossLogix Daniel Engovatov dengovatov@crosslogix.com
5046 Entegrity Hal Lockhart hal.lockhart@entegrity.com
5047 Entrust Carlisle Adams carlisle.adams@entrust.com
5048 Entrust Tim Moses tim.moses@entrust.com
5049 Quadrasis Don Flinn Don.Flinn@hitachisoftware.com
5050 Quadrasis Konstantin Beznosov konstantin.beznosov@quadrasis.com
5051 OpenNetwork Steve Andersen sanderson@opennetwork.com
5052 Overxeer Bill Parducci bill.parducci@overxeer.com
5053 Overxeer Simon Godik simon.godik@overxeer.com
5054 IBM Michiharu Kudo kudo@jp.ibm.com
5055 Self Polar Humenn polar@syr.edu
5056 Sterling Commerce Suresh Damodaran Suresh_Damodaran@stercomm.com
5057 Sun Microsystems Anne Anderson Anne.Anderson@Sun.com
5058 Sun Microsystems Pirasenna Velandai Thiyagarajan Pirasenna.Thiyagarajan@Sun.com
5059 Xtradyne Gerald Brose Gerald.Brose@xtradyne.com

5060

Appendix E. Revision history

| Rev | Date | By whom | What |
|------|------------|---------------------------------|--------------------------------|
| V1.0 | 6 Nov 2002 | XACML Technical Committee | First committee specification. |
| | | | |
| | | | |
| | | | |
| | | | |

5061

Appendix F. Notices

5062

5063 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
5064 that might be claimed to pertain to the implementation or use of the technology described in this
5065 document or the extent to which any license under such rights might or might not be available;
5066 neither does it represent that it has made any effort to identify any such rights. Information on
5067 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
5068 website. Copies of claims of rights made available for publication and any assurances of licenses to
5069 be made available, or the result of an attempt made to obtain a general license or permission for
5070 the use of such proprietary rights by implementors or users of this specification, can be obtained
5071 from the OASIS Executive Director.

5072 OASIS has been notified of intellectual property rights claimed in regard to some or all of the
5073 contents of this specification. For more information consult the online list of claimed rights.

5074 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
5075 applications, or other proprietary rights which may cover technology that may be required to
5076 implement this specification. Please address the information to the OASIS Executive Director.

5077 Copyright (C) OASIS Open 2002. All Rights Reserved.

5078 This document and translations of it may be copied and furnished to others, and derivative works
5079 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
5080 published and distributed, in whole or in part, without restriction of any kind, provided that the above
5081 copyright notice and this paragraph are included on all such copies and derivative works. However,
5082 this document itself may not be modified in any way, such as by removing the copyright notice or
5083 references to OASIS, except as needed for the purpose of developing OASIS specifications, in
5084 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights
5085 document must be followed, or as required to translate it into languages other than English.

5086 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
5087 successors or assigns.

5088 This document and the information contained herein is provided on an "AS IS" basis and OASIS
5089 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
5090 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY
5091 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
5092 PARTICULAR PURPOSE.