eXtensible Access Control Markup Language (XACML) Version 1.1

Committee Specification, 07 August 2003

Document identifier: cs-xacml-specification-1.1.pdf
Location: http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf
Send comments to: xacml-comment@lists.oasis-open.org

Editors:
Simon Godik, Overxeer
Tim Moses, Entrust

Committee members:
Anne Anderson, Sun Microsystems
Antony Nadalin, IBM
Bill Parducci, Overxeer
Daniel Engovatov, BEA Systems
Hal Lockhart, BEA Systems
Michiharu Kudo, IBM
Polar Humenn, Self
Simon Godik, Overxeer
Steve Anderson, OpenNetwork
Steve Crocker, Pervasive Security Systems
Tim Moses, Entrust

Abstract:
This specification defines an XML schema for an extensible access-control policy language.

Status:
This version of the specification is a Committee Specification.

If you are on the xacml@lists.oasis-open.org list for committee members, send comments there. If you are not on that list, subscribe to the xacml-comment@lists.oasis-open.org list and send comments there. To subscribe, send an email message to xacml-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.
Copyright (C) OASIS Open 2003. All Rights Reserved.
# Table of contents

1. Introduction (non-normative) ................................................................. 10  
   1.1. Glossary ................................................................................. 10  
      1.1.1 Preferred terms .............................................................. 10  
      1.1.2 Related terms ................................................................. 11  
   1.2. Notation .................................................................................. 12  
   1.3. Schema organization and namespaces ..................................... 12  
2. Background (non-normative) ................................................................. 13  
   2.1. Requirements ......................................................................... 13  
   2.2. Rule and policy combining ...................................................... 14  
   2.3. Combining algorithms ............................................................. 14  
   2.4. Multiple subjects ................................................................... 15  
   2.5. Policies based on subject and resource attributes .................... 15  
   2.6. Multi-valued attributes .......................................................... 15  
   2.7. Policies based on resource contents ....................................... 16  
   2.8. Operators .............................................................................. 16  
   2.9. Policy distribution ................................................................. 17  
   2.10. Policy indexing .................................................................... 17  
   2.11. Abstraction layer .................................................................. 17  
   2.12. Actions performed in conjunction with enforcement .............. 18  
3. Models (non-normative) ....................................................................... 18  
   3.1. Data-flow model .................................................................... 18  
   3.2. XACML context ..................................................................... 20  
   3.3. Policy language model ............................................................ 20  
      3.3.1 Rule .................................................................................. 21  
      3.3.2 Policy ............................................................................... 23  
      3.3.3 Policy set ......................................................................... 24  
4. Examples (non-normative) ................................................................... 25  
   4.1. Example one ......................................................................... 25  
      4.1.1 Example policy ................................................................. 25  
      4.1.2 Example request context ................................................... 27  
      4.1.3 Example response context ............................................... 28  
   4.2. Example two ......................................................................... 28  
      4.2.1 Example medical record instance ...................................... 29  
      4.2.2 Example request context ................................................... 30  
      4.2.3 Example plain-language rules .......................................... 32
4.2.4 Example XACML rule instances .................................................................................................32
5. Policy syntax (normative, with the exception of the schema fragments) .........................................46
  5.1. Element <PolicySet> .................................................................................................................46
  5.2. Element <Description> ............................................................................................................47
  5.3. Element <PolicySetDefaults> .................................................................................................47
  5.4. Element <XPathVersion> ........................................................................................................48
  5.5. Element <Target> .....................................................................................................................48
  5.6. Element <Subjects> ...................................................................................................................49
  5.7. Element <Subject> ....................................................................................................................49
  5.8. Element <AnySubject> .............................................................................................................49
  5.9. Element <SubjectMatch> .........................................................................................................49
  5.10. Element <Resources> ............................................................................................................50
  5.11. Element <Resource> ...............................................................................................................50
  5.12. Element <AnyResource> .......................................................................................................51
  5.13. Element <ResourceMatch> ....................................................................................................51
  5.14. Element <Actions> ...............................................................................................................52
  5.15. Element <Action> ..................................................................................................................52
  5.16. Element <AnyAction> ............................................................................................................52
  5.17. Element <ActionMatch> .......................................................................................................52
  5.18. Element <PolicySetIdReference> ..........................................................................................53
  5.19. Element <PolicyIdReference> ...............................................................................................53
  5.20. Element <Policy> ...................................................................................................................53
  5.21. Element <PolicyDefaults> .....................................................................................................55
  5.22. Element <Rule> .....................................................................................................................55
  5.23. Simple type EffectType ..........................................................................................................56
  5.24. Element <Condition> ............................................................................................................56
  5.25. Element <Apply> ....................................................................................................................56
  5.26. Element <Function> ..............................................................................................................57
  5.27. Complex type AttributeDesignatorType ...............................................................................57
  5.28. Element <SubjectAttributeDesignator> ..............................................................................58
  5.29. Element <ResourceAttributeDesignator> ...........................................................................59
  5.30. Element <ActionAttributeDesignator> ..............................................................................60
  5.31. Element <EnvironmentAttributeDesignator> ....................................................................60
  5.32. Element <AttributeSelector> ..............................................................................................61
  5.33. Element <AttributeValue> ....................................................................................................62
  5.34. Element <Obligations> ............................................................................................................63
  5.35. Element <Obligation> .............................................................................................................63
109 5.36. Element <AttributeAssignment> ................................................................. 64
110 6. Context syntax (normative with the exception of the schema fragments) .................... 64
111 6.1. Element <Request> .................................................................................. 64
112 6.2. Element <Subject> .................................................................................. 65
113 6.3. Element <Resource> ................................................................................ 66
114 6.4. Element <ResourceContent> .................................................................... 66
115 6.5. Element <Action> ................................................................................... 67
116 6.6. Element <Environment> ......................................................................... 67
117 6.7. Element <Attribute> ............................................................................... 67
118 6.8. Element <AttributeValue> ........................................................................ 68
119 6.9. Element <Response> ............................................................................... 68
120 6.10. Element <Result> ................................................................................... 69
121 6.11. Element <Decision> ............................................................................. 70
122 6.12. Element <Status> ................................................................................. 70
123 6.13. Element <StatusCode> .......................................................................... 71
125 6.15. Element <StatusDetail> ................................................................. 71
126 7. Functional requirements (normative) ................................................................ 72
127 7.1. Policy enforcement point ........................................................................ 72
128 7.2. Base policy .............................................................................................. 72
129 7.3. Target evaluation .................................................................................... 73
130 7.4. Condition evaluation ............................................................................... 73
131 7.5. Rule evaluation ....................................................................................... 73
132 7.6. Policy evaluation ..................................................................................... 73
133 7.7. Policy Set evaluation ............................................................................... 74
134 7.8. Hierarchical resources ........................................................................... 75
135 7.9. Attributes ................................................................................................ 76
136 7.9.1. Attribute Matching ............................................................................. 76
137 7.9.2. Attribute Retrieval .............................................................................. 76
138 7.9.3. Environment Attributes ...................................................................... 77
139 7.10. Authorization decision ......................................................................... 77
140 7.11. Obligations ............................................................................................. 77
141 7.12. Unsupported functionality ...................................................................... 78
142 7.13. Syntax and type errors ........................................................................... 78
143 8. XACML extensibility points (non-normative) ......................................................... 78
144 8.1. Extensible XML attribute types .............................................................. 78
145 8.2. Structured attributes ............................................................................... 79
9. Security and privacy considerations (non-normative)...........................................................................79
  9.1. Threat model .................................................................................................................................79
    9.1.1. Unauthorized disclosure ........................................................................................................80
    9.1.2. Message replay .......................................................................................................................80
    9.1.3. Message insertion ....................................................................................................................80
    9.1.4. Message deletion .....................................................................................................................80
    9.1.5. Message modification ..............................................................................................................80
    9.1.6. NotApplicable results ...........................................................................................................81
    9.1.7. Negative rules .........................................................................................................................81
  9.2. Safeguards ........................................................................................................................................82
    9.2.1. Authentication ........................................................................................................................82
    9.2.2. Policy administration ............................................................................................................82
    9.2.3. Confidentiality .......................................................................................................................82
    9.2.4. Policy integrity .......................................................................................................................83
    9.2.5. Policy identifiers ....................................................................................................................83
    9.2.6. Trust model ............................................................................................................................84
    9.2.7. Privacy ........................................................................................................................................84
  10. Conformance (normative)..................................................................................................................84
    10.1. Introduction ...............................................................................................................................84
    10.2. Conformance tables ..................................................................................................................84
      10.2.1. Schema elements ...............................................................................................................85
      10.2.2. Identifier Prefixes ..............................................................................................................86
      10.2.3. Algorithms ........................................................................................................................86
      10.2.4. Status Codes .......................................................................................................................86
      10.2.5. Attributes ..........................................................................................................................87
      10.2.6. Identifiers ............................................................................................................................87
      10.2.7. Data-types ..........................................................................................................................87
      10.2.8. Functions ............................................................................................................................88
  11. References ........................................................................................................................................92
  Appendix A. Standard data-types, functions and their semantics (normative) .......................................94
    A.1. Introduction ...................................................................................................................................94
    A.2. Primitive types ............................................................................................................................94
    A.3. Structured types ............................................................................................................................95
    A.4. Representations ............................................................................................................................95
    A.5. Bags ................................................................................................................................................96
    A.6. Expressions ....................................................................................................................................96
    A.7. Element <AttributeValue> ..........................................................................................................97
Errata

Errata can be found at the following location:

http://www.oasis-open.org/committees/xacml/repository/errata-001.pdf
1. Introduction (non-normative)

1.1. Glossary

1.1.1 Preferred terms

**Access** - Performing an *action*

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

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

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

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

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

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

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

**Conjunctive sequence** - a sequence of boolean elements combined using the logical ‘AND’ operation

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

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

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

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

**Disjunctive sequence** - a sequence of boolean elements combined using the logical ‘OR’ operation

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

**Environment** - The set of *attributes* that are relevant to an *authorization decision* and are independent of a particular *subject, resource* or *action*
Obligation - An operation specified in a policy or policy set that should be performed in conjunction with the enforcement of an authorization decision

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

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

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

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

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

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

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

Predicate - A statement about attributes whose truth can be evaluated

Resource - Data, service or system component

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

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

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

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

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

1.1.2 Related terms

In the field of access control and authorization there are several closely related terms in common use. For purposes of precision and clarity, certain of these terms are not used in this specification. For instance, the term attribute is used in place of the terms: group and role.

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

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

Requestors and initiators are covered by the term subject.
1.2. Notation

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

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

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

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

Listings of XACML schemas appear like this.

Example code listings appear like this.

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

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

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

1.3. Schema organization and namespaces

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

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

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

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

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

http://www.w3.org/2000/09/xmldsig#
2. Background (non-normative)

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

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

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

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

2.1. Requirements

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

- To provide a method for combining individual rules and policies into a single policy set that
  applies to a particular decision request.

- To provide a method for flexible definition of the procedure by which rules and policies are
  combined.

- To provide a method for dealing with multiple subjects acting in different capacities.

- To provide a method for basing an authorization decision on attributes of the subject and
  resource.

- To provide a method for dealing with multi-valued attributes.

- To provide a method for basing an authorization decision on the contents of an information
  resource.

- To provide a set of logical and mathematical operators on attributes of the subject, resource
  and environment.
• To provide a method for handling a distributed set of policy components, while abstracting the
method for locating, retrieving and authenticating the policy components.

• To provide a method for rapidly identifying the policy that applies to a given action, based upon
the values of attributes of the subjects, resource and action.

• To provide an abstraction-layer that insulates the policy-writer from the details of the application
environment.

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

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

2.2. Rule and policy combining

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

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

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

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

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

2.3. Combining algorithms

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

• Deny-overrides (Ordered and Unordered),

• Permit-overrides (Ordered and Unordered),
First applicable and

Only-one-applicable.

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

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

2.4. Multiple subjects

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

2.5. Policies based on subject and resource attributes

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

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

2.6. Multi-valued attributes

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

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

2.7. Policies based on resource contents

In many applications, it is required to base an authorization decision on data contained in the
information resource to which access is requested. For instance, a common component of privacy
policy is that a person should be allowed to read records for which he or she is the subject. The

Corresponding policy must contain a reference to the subject identified in the information resource
itself.

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

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

2.8. Operators

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

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

Policy. Hence the need for set operations.

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

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

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

Also noteworthy are the operators over boolean data-types, which permit the logical combination of
predicates in a rule. For example, a rule may contain the statement that access may be
permitted during business hours AND from a terminal on business premises.
The XACML method of representing functions borrows from MathML [MathML] and from the
XQuery 1.0 and XPath 2.0 Functions and Operators specification [XF].

2.9. Policy distribution

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

Policy elements may be attached to the information resources to which they apply, as
described by Prettit [Prettit93]. Alternatively, Policy elements may be maintained in one or
more locations from which they are retrieved for evaluation. In such cases, the applicable policy
may be referenced by an identifier or locator closely associated with the information resource.

2.10. Policy indexing

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

Two approaches are supported:

1. Policy statements may be stored in a database, whose data-model is congruent with that of the
   <Target> element. The PDP should use the contents of the decision request that it is
   processing to form the database read command by which applicable policy statements are
   retrieved. Nevertheless, the PDP should still evaluate the <Target> element of the retrieved
   policy or policy set statements as defined by the XACML specification.

2. Alternatively, the PDP may evaluate the <Target> element from each of the policies or
   policy sets that it has available to it, in the context of a particular decision request, in order to
   identify the policies and policy sets that are applicable to that request.

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

2.11. Abstraction layer

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

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

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

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

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

2.12. Actions performed in conjunction with enforcement

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

3. Models (non-normative)

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

3.1. Data-flow model

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

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

The model operates by the following steps.

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

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

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

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

5. The PIP obtains the requested attributes.

6. The PIP returns the requested attributes to the context handler.
7. Optionally, the **context handler** includes the **resource** in the **context**.

8. The **context handler** sends a **decision request**, including the **target**, to the **PDP**. The **PDP** identifies the **applicable policy** and retrieves the required **attributes** and (optionally) the **resource** from the **context handler**. The **PDP** evaluates the **policy**.

9. The **PDP** returns the response **context** (including the **authorization decision**) to the **context handler**.

10. The **context handler** translates the response **context** to the native response format of the **PEP**. The **context handler** returns the response to the **PEP**.

11. The **PEP** fulfills the **obligations**.

12. (Not shown) If **access** is permitted, then the **PEP** permits **access** to the **resource**; otherwise, it denies **access**.

### 3.2. XACML context

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

![Figure 2 - XACML context](image)

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

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

### 3.3. Policy language model

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

- **Rule**;
- **Policy**; and
• **Policy set.**

These are described in the following sub-sections.

![Policy language model diagram]

**Figure 3 - Policy language model**

### 3.3.1 Rule

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

These are discussed in the following sub-sections.

3.3.1.1. Rule target

The target defines the set of:

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

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

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

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

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

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

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

1. the contents of the identified node only,
2. the contents of the identified node and the contents of its immediate child nodes or
3. the contents of the identified node and all its descendant nodes.
All three options are supported in XACML.

### 3.3.1.2. Effect

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

### 3.3.1.3. Condition

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

### 3.3.2 Policy

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

- a **target**;
- a **rule-combining algorithm**-identifier;
- a set of **rules**; and
- **obligations**.

**Rules** are described above. The remaining components are described in the following subsections.

#### 3.3.2.1. Policy target

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

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

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

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

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

3.3.2.3. Obligations

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

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

3.3.3 Policy set

A policy set comprises four main components:

- a target;
- a policy-combining algorithm-identifier
- a set of policies; and
- obligations.

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

3.3.3.1. Policy-combining algorithm

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

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

3.3.3.2. Obligations

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

When a PDP evaluates a policy set containing obligations, it returns certain of those obligations to the PEP in its response context. Section 7.11 explains which obligations are to be returned.
4. Examples (non-normative)

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

4.1. Example one

4.1.1 Example policy

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

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

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

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> RuleCombiningAlgorithm="Identifier:rule-combining-algorithm:deny-overrides">
```

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

[p02] introduces the XACML Policy itself.

[p03-p05] are XML namespace declarations.

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

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

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

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

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

```
<p11> <Target>
<p12> <Subjects>
<p13> <AnySubject/>
<p14> </Subjects>
<p15> <Resources>
```

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

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

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

[p23] specifies the identifier for this rule.

[p24] says what effect this rule has if the rule evaluates to “True”. Rules can have an effect of
either “Permit” or “Deny”. In this case, the rule will evaluate to “Permit”, meaning that, as far as this
one rule is concerned, the requested access should be permitted. If a rule evaluates to “False”,
then it returns a result of “NotApplicable”. If an error occurs when evaluating the rule, the rule
returns a result of “Indeterminate”. As mentioned above, the rule-combining algorithm for the
policy tells how various rule values are combined into a single policy value.

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

[p29] <Target>

[p29] introduces the target of the rule. As described above for the target of a policy, the target of
a rule describes the decision requests to which this rule applies. If the subject, resource and
action in a decision request do not match the values specified in the rule target, then the
remainder of the rule does not need to be evaluated, and a value of “NotApplicable” is returned to
the policy evaluation.
The rule target is similar to the target of the policy itself, but with one important difference. [p32-p41] do not say <AnySubject/>., but instead spell out a specific value that the subject in the decision request must match. The <SubjectMatch> element specifies a matching function in the MatchId attribute, a pointer to a specific subject attribute in the request context by means of the <SubjectAttributeDesignator> element, and a literal value of "medico.com". The matching function will be used to compare the value of the subject attribute with the literal value. Only if the match returns "True" will this rule apply to a particular decision request. If the match returns "False", then this rule will return a value of "NotApplicable".

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

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

### 4.1.2 Example request context

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

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

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

```
[co1] <xml version="1.0" encoding="UTF-8"?>
[co2] <Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
[co3] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
[co4] xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:context"
[co5] http://www.oasis-open.org/tc/xacml/1.0/cs-xacml-schema-context-01.xsd">
```

[co1-05] are the header for the request context, and are used the same way as the header for the policy explained above.

The <Subject> element contains one or more attributes of the entity making the access request. There can be multiple subjects, and each subject can have multiple attributes. In this case, in [co6-c11], there is only one subject, and the subject has only one attribute: the subject's identity, expressed as an e-mail name, is "bs@simpsons.com".

The <Resource> element contains one or more attributes of the resource to which the subject (or subjects) has requested access. There can be only one <Resource>
per decision request. Lines [c13-c16] contain the one attribute of the resource
to which Bart Simpson has requested 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>

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

[c24] </Request>

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

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

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

4.1.3 Example response context

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

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

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

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

[r05-r07] contains the result of evaluating the decision request
against the policy. In this case, the result is "NotApplicable". A policy can return "Permit", "Deny",
"NotApplicable" or "Indeterminate".

[r08] </Response>

[r08] closes the response context.

4.2. Example two

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

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <patient>
    <patientName>
      <first>Bartholomew</first>
      <last>Simpson</last>
    </patientName>
    <patientContact>
      <street>27 Shelbyville Road</street>
      <city>Springfield</city>
      <state>MA</state>
      <zip>12345</zip>
      <phone>555.123.4567</phone>
      <fax/>
      <email/>
    </patientContact>
    <patientDoB>1992-03-21</patientDoB>
    <patientGender>male</patientGender>
    <patient-number>555555</patient-number>
  </patient>
  <parentGuardian>
    <parentGuardianId>HS001</parentGuardianId>
    <parentGuardianName>
      <first>Homer</first>
      <last>Simpson</last>
    </parentGuardianName>
    <parentGuardianContact>
      <street>27 Shelbyville Road</street>
      <city>Springfield</city>
      <state>MA</state>
      <zip>12345</zip>
      <phone>555.123.4567</phone>
      <fax/>
      <email>homer@aol.com</email>
    </parentGuardianContact>
  </parentGuardian>
  <primaryCarePhysician>
    <physicianName>
      <first>Julius</first>
      <last>Hibbert</last>
    </physicianName>
    <physicianContact>
      <street>1 First St</street>
      <city>Springfield</city>
      <state>MA</state>
      <zip>12345</zip>
      <phone>555.123.9012</phone>
      <fax>555.123.9013</fax>
      <email/>
    </physicianContact>
    <registrationID>ABC123</registrationID>
  </primaryCarePhysician>
  <insurer>
    <name>Blue Cross</name>
    <street>1234 Main St</street>
    <city>Springfield</city>
  </insurer>
</record>
```
4.2.2 Example request context

The following example illustrates a request context to which the example rules may be applicable. It represents a request by the physician Julius Hibbert to read the patient date of birth in the record of Bartholomew Simpson.
[32]  <md:patient>
[34]  </md:patient>
[35]  <!-- other fields -->
[36]  </md:record>
[37]  </ResourceContent>
[38]  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#string">
[39]    <AttributeValue>
[40]      /medico.com/records/bart-simpson.xml#
[41]      xmlns(md=http://www.medico.com/schemas/record.xsd)
[43]    </AttributeValue>
[44]  </Attribute>
[45]  </Resource>
[46]  <Action>
[47]    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath" DataType="http://www.w3.org/2001/XMLSchema#string">
[48]      <AttributeValue>
[50]      </AttributeValue>
[51]    </Attribute>
[52]  </Action>
[53]  </Request>

[02]-[03] Standard namespace declarations.

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

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

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

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

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

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

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

[38]-[46] **Resource** identifier.
The **Resource** is identified with an Xpointer expression that names the URI of the file that
is accessed, the target namespace of the document, and the XPath location path to the specific
element.

The XPath location path in the “resource-id” attribute is extracted and placed in the
``xpath`` attribute.

*Resource* target-namespace *attribute*.

**Action attributes** are placed in the *Action* section of the *Request*.

**Action** identifier.

### 4.2.3 Example plain-language rules

The following plain-language rules are to be enforced:

1. **Rule 1:** A person, identified by his or her patient number, may read any record for which he
or she is the designated patient.
2. **Rule 2:** A person may read any record for which he or she is the designated parent or
   guardian, and for which the patient is under 16 years of age.
3. **Rule 3:** A physician may write to any medical element for which he or she is the designated
   primary care physician, provided an email is sent to the patient.
4. **Rule 4:** An administrator shall not be permitted to read or write to medical elements of a
   patient record.

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

### 4.2.4 Example XACML rule instances

#### 4.2.4.1. Rule 1

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Rule>
  <Condition>
    <AnySubject/>
    <Resource/>
    <Effect>Permit</Effect>
    <Description>A person may read any medical record in the
                  http://www.medico.com/schemas/record.xsd namespace
                  for which he or she is a designated patient</Description>
  </Condition>
</Rule>
```

[22] <ResourceMatch
[23]  MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[24]  <AttributeValue
[26]     <![CDATA[http://www.w3.org/2001/XMLSchema#string]]></DATA
[28]  </AttributeValue>
[29]  <ResourceAttributeDesignator
[31]    <![CDATA[http://www.w3.org/2001/XMLSchema#string]]></DATA
[32]  </ResourceAttributeDesignator>
[33] </ResourceMatch>
[34] </Resources>
[35] </Actions>
[36] </Action>
[37] </Target>
[38] </Resource>
[39] </Actions>
[40] </ResourceMatch>
[41] <Condition
[42]  FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[43]   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
[44]         and-only">
[45]    <SubAttributeDesignator
[47]       <![CDATA[http://www.w3.org/2001/XMLSchema#string]]></DATA
[48]    </SubAttributeDesignator>
[49]   </Apply>
[50] </Condition>
[51] </Actions>
[52] </ResourceMatch>
[53] <Policy
[55]  <Assertion
[56]   FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
[57]         and-only">
[58]    <AttributeSelector
[59]      RequestContextPath="/md:record/md:patien"/text()"
[60]      DataType="urn:oasis:names:tc:xacml:1.0:datatype:xpath":node-
[61]         match">
[62]     <![CDATA[http://www.w3.org/2001/XMLSchema#string]]></DATA
[63]    </AttributeSelector>
[64]   </Apply>
[65] </Assertion>
[66] </Policy>

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

[07] **Rule** identifier.

[08]. When a *rule* evaluates to ‘True’ it emits the value of the *Effect* attribute. This value is combined with the *Effect* values of other rules according to the *rule-combining algorithm*. 
Free form description of the rule.

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

The Subjects element may contain either a disjunctive sequence of Subject elements or AnySubject element.

The AnySubject element is a special element that matches any subject in the request context.

The Resources element may contain either a disjunctive sequence of Resource elements or AnyResource element.

The Resource element encloses the conjunctive sequence of ResourceMatch elements.

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

The MatchId attribute names the matching function.

Literal attribute value to match.

The ResourceAttributeDesignator element selects the resource attribute values from the request context. The attribute name is specified by the AttributeId. The selection result is a bag of values.

The ResourceMatch. This match compares the results of two XPath expressions. The first XPath expression is /md:record and the second XPath expression is the location path to the requested xml element. The "xpath-node-match" function evaluates to “True” if the requested XML element is below the /md:record element.

MatchId attribute names the matching function.

The literal XPath expression to match. The md prefix is resolved using a standard namespace declaration.

The ResourceAttributeDesignator selects the bag of values for the "urn:oasis:names:tc:xacml:1.0:xpath" resource attribute. Here, there is just one element in the bag, which is the location path for the requested XML element.

The Actions element may contain either a disjunctive sequence of Action elements or an AnyAction element.

The Action element contains a conjunctive sequence of ActionMatch elements.

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

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

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

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

[49] The FunctionId attribute of the <Condition> element names the function to be used for comparison. In this case, comparison is done with

urn:oasis:names:tc:xacml:1.0:function:contains-equal; this function takes two arguments of the "http://www.w3.org/2001/XMLSchema#string" data-type.

[50] The first argument to the "urn:oasis:names:tc:xacml:1.0:function:contains-equal" in the Condition. Functions can take other functions as arguments. The Apply element encodes the function call with the FunctionId attribute naming the function. Since

urn:oasis:names:tc:xacml:1.0:function:contains-equal takes arguments of the "http://www.w3.org/2001/XMLSchema#string" data-type and

SubjectAttributeDesignator selects a bag of

"http://www.w3.org/2001/XMLSchema#string" values,

"urn:oasis:names:tc:xacml:1.0:function:contains-one-and-only" is used. This function guarantees that its argument evaluates to a bag containing one and only one

"http://www.w3.org/2001/XMLSchema#string" element.

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

[55] The second argument to the "urn:oasis:names:tc:xacml:1.0:function:contains-equal" in the Condition. Functions can take other functions as arguments. The Apply element encodes function call with the FunctionId attribute naming the function. Since

"urn:oasis:names:tc:xacml:1.0:function:contains-equal takes arguments of the "http://www.w3.org/2001/XMLSchema#string" data-type and the AttributeSelector selects a bag of "http://www.w3.org/2001/XMLSchema#string" values,

"urn:oasis:names:tc:xacml:1.0:function:contains-one-and-only" is used. This function guarantees that its argument evaluates to a bag containing one and only one

"http://www.w3.org/2001/XMLSchema#string" element.

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

### 4.2.4.2. Rule 2

Rule 2 illustrates the use of a mathematical function, i.e. the <Apply> element with functionId "urn:oasis:names:tc:xacml:1.0:function:dateTimeAddYearMonthDuration" to calculate date. It also illustrates the use of predicate expressions, with the functionId "urn:oasis:names:tc:xacml:1.0:function:and".

```xml
<xml version="1.0" encoding="UTF-8"/>
```

oasis-###-xacml-1.1.pdf 35
<Rule>
    <xmlns:urn:oasis:names:tc:xacml:1.0:policy>
        <xmlns:urn:oasis:names:tc:xacml:1.0:context>
            <xmlns:urn:oasis:names:tc:xacml:1.0:md>
                <RuleId>urn:oasis:names:tc:xacml:examples:ruleid:2</RuleId>
                <Effect>Permit</Effect>
                <Description>
                    A person may read any medical record in the
                    http://www.medico.com/records.xsd namespace
                    for which he or she is the designated parent or guardian,
                    and for which the patient is under 16 years of age
                </Description>
                <Target>
                    <Subjects>
                        <AnySubject/>
                    </Subjects>
                    <Resources>
                        <Resource>
                            <!-- match document target namespace -->
                            <ResourceMatch>
                                <MatchId>
                                    urn:oasis:names:tc:xacml:1.0:function:string-equal
                                </MatchId>
                                <DataType>
                                    http://www.w3.org/2001/XMLSchema#string
                                </DataType>
                                <AttributeValue>
                                    http://www.medico.com/schemas/record.xsd
                                </AttributeValue>
                                <ResourceAttributeDesignator AttributeId=
                                    "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
                                >
                                    <DataType>
                                        http://www.w3.org/2001/XMLSchema#string
                                    </DataType>
                                    <AttributeValue>
                                        /md:record</AttributeValue>
                                    </ResourceAttributeDesignator>
                                </ResourceMatch>
                            </Resource>
                            <!-- match requested xml element -->
                            <ResourceMatch>
                                <MatchId>
                                    urn:oasis:names:tc:xacml:1.0:function:xpath-node-match
                                </MatchId>
                                <DataType>
                                    http://www.w3.org/2001/XMLSchema#string
                                </DataType>
                                <AttributeValue>
                                    /md:record</AttributeValue>
                                </ResourceMatch>
                            </Resource>
                            <!-- match 'read' action -->
                            <ActionMatch>
                                <MatchId>
                                    urn:oasis:names:tc:xacml:1.0:function:string-equal
                                </MatchId>
                                <DataType>
                                    http://www.w3.org/2001/XMLSchema#string
                                </DataType>
                                <AttributeValue>
                                    read</AttributeValue>
                                </ResourceAttributeDesignator>
                            </ActionMatch>
                        </Resources>
                        <Actions>
                            <!-- compare parent-guardian-id subject attribute with
                                 the value in the document -->
                            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                                <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one
                                 and-only">
                            </Apply>
                        </Actions>
                    </Target>
                </Target>
            </xmlns:urn:oasis:names:tc:xacml:1.0:md>
        </xmlns:urn:oasis:names:tc:xacml:1.0:context>
    </xmlns:urn:oasis:names:tc:xacml:1.0:policy>
[53] <!-- parent-guardian-id subject attribute -->
[54] <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id"
[55]   DataType="http://www.w3.org/2001/XMLSchema#string"/>
[56] </Apply>
[57] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
[58] <!-- parent-guardian-id element in the document -->
[60]   DataType="http://www.w3.org/2001/XMLSchema#string">
[61]   </AttributeSelector>
[62] </Apply>
[63] </Apply>
[64] </Apply>
[65] </Apply>
[66] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal">
[67]   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-only">
[68]     <EnvironmentAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date"
[69]       DataType="http://www.w3.org/2001/XMLSchema#date"/>
[70]   </Apply>
[71] </Apply>
[72] </Apply>
[73] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration">
[75]     DataType="http://www.w3.org/2001/XMLSchema#date">
[76]     </AttributeSelector>
[77]   </Apply>
[78]   <Apply>
[79]     <AttributeValue DataType="http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration">P16Y</AttributeValue>
[80]   </Apply>
[81] </Apply>
[82] </Apply>
[83] </Condition>
[84] </Rule>

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

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

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

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

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

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

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

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

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

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

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

[68]-[69] Current date is evaluated by selecting the

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

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

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

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

4.2.4.3. Rule 3

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

[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <Policy
[03] xmlns="urn:oasis:names:tc:xacml:1.0:policy"
[04] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
[05] xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
[06] xmlns:md="http://www.medico.com/schemas/record.xsd"
[07] PolicyId="urn:oasis:names:tc:xacml:1.0:examples:policyid:3"
[08] RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
[09] rule-combining-algorithm:deny-overrides">
[10] <Description>
[11] Policy for any medical record in the
[13] </Description>
[14] <Target>
[15] <Subjects>
[16] <AnySubject/>
[17] </Subjects>
[18] <Resources>
[19] <Resource>
[20] <![-- match document target namespace -->]
[21] <ResourceMatch
[22] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[23] <AttributeValue
[24] DataType="http://www.w3.org/2001/XMLSchema#string">
[26] </AttributeValue>
[27] <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:target-name-space"
[28] DataType="http://www.w3.org/2001/XMLSchema#string"/>
[29] </ResourceMatch>
[31] </Resources>
[32] <Actions>
[33] <AnyAction/>
[34] </Actions>
[35] </Target>
[37] Effect="Permit">
[38] <Description>
[39] A physician may write any medical element in a record
[40] for which he or she is the designated primary care
[41] physician, provided an email is sent to the patient
[42] </Description>
[43] </Rule>
[44] <Target>
[45] <Subjects>
[46] <Subject>
[47] <![-- match subject group attribute -->]
[48] <SubjectMatch
[50] <AttributeValue
[51] DataType="http://www.w3.org/2001/XMLSchema#string">physician</AttributeValue>
[52] <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:example:attribute:role"
[53] DataType="http://www.w3.org/2001/XMLSchema#string"/>
[54] </SubjectMatch>
[55] </Subject>
[56] </Subjects>
[57] <Resources>
[58] <Resource>
[59] <![-- match requested xml element -->]
[60] <ResourceMatch
[61] MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
[56] <AttributeValue
[57]  DataType="http://www.w3.org/2001/XMLSchema#string">
[58]   /md:record/md:medical
[59] </AttributeValue>
[60] <ResourceAttributeDesignator AttributeId="
[61]   urn:oasis:names:tc:xacml:1.0:resource:xpath">
[62]  DataType="http://www.w3.org/2001/XMLSchema#string"/>
[63] </Resource>
[64] </Actions>
[65] </Action>
[66] </match action -->
[67] <ActionMatch
[68]  MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[69] <AttributeValue
[70]  DataType="http://www.w3.org/2001/XMLSchema#string">
[71]   write</AttributeValue>
[72] </ActionMatch>
[73] </Action>
[74] </Actions>
[75] </Target>
[76] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
[77]   one-and-only">
[78]  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
[79]   and-only">
[80]   <!-- physician-id subject attribute -->
[81]   <SubjectAttributeDesignator AttributeId="
[82]     urn:oasis:names:tc:xacml:1.0:example:
[83]       attribute:physician-id">
[84]  DataType="http://www.w3.org/2001/XMLSchema#string"/>
[85] </Apply>
[86] </Apply>
[87] </Condition>
[88] </Rule>
[89] </Obligations>
[90] </Obligation>
[91] <!-- send e-mail message to the document owner -->
[92] <Obligation ObligationId="
[94]   FulfillOn="Permit">
[95] <AttributeAssignment AttributeId="
[96]   urn:oasis:names:tc:xacml:1.0:example:attribute:mailto">
[97]  DataType="http://www.w3.org/2001/XMLSchema#string">
[98] <AttributeSelector RequestContextPath="
[99]   "/md:record/md:primaryCarePhysician/md:registrationID/text()"
[100]  DataType="http://www.w3.org/2001/XMLSchema#string"/>
[101] </AttributeAssignment>
[102] <AttributeAssignment AttributeId="
[103]   urn:oasis:names:tc:xacml:1.0:example:attribute:text">
[104]  DataType="http://www.w3.org/2001/XMLSchema#string">
[105] <AttributeValue
[106]  DataType="http://www.w3.org/2001/XMLSchema#string">
[107]   Your medical record has been accessed by:
[108] </AttributeValue>
[109] <AttributeAssignment>
[110] <AttributeAssignment AttributeId="
[111]  DataTyype="http://www.w3.org/2001/XMLSchema#string">
[112]  <SubjectAttributeDesignator AttributeId="
[113]  "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
[114]  DataTyype="http://www.w3.org/2001/XMLSchema#string">
[114]  </AttributeAssignment>
[115]  </Obligation>
[116]  </Obligations>
[117]  </Policy>

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

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

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


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

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

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

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

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

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

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

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

[94] FulfillOn attribute defines an authorization decision value for which this obligation must
be fulfilled.
[95]-[101] **Obligation** may have one or more parameters. The **obligation** parameter
from the content of the XML document.

[95-96] **AttributeId** declares


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

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

[102]-[108] The **obligation** parameter

"urn:oasis:names:tc:xacml:1.0:examples:attribute:text" of data-type

"http://www.w3.org/2001/XMLSchema#string" is assigned the literal value "Your
medical record has been accessed by:"

[109]-[114] The **obligation** parameter

"urn:oasis:names:tc:xacml:1.0:examples:attribute:text" of the data-type is assigned the value of the
"urn:oasis:names:tc:xacml:1.0:subject:subject-id" **subject attribute**.

### 4.2.4.4. Rule 4

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

```xml
<Rule>
  <Description>
    An Administrator shall not be permitted to read or write
    medical elements of a patient record in the
  </Description>
  <Target>
    <Subjects>
      <Subject>
        <!-- match role subject attribute -->
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <!-- match document target namespace -->
      </Resource>
    </Resources>
  </Target>
  <Condition>
    <AttributeMatch>
      <MatchId>
        "urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto"
      </MatchId>
    </AttributeMatch>
  </Condition>
</Rule>
```

```xml
<Rule>
  <Description>
    An Administrator shall not be permitted to read or write
    medical elements of a patient record in the
  </Description>
  <Target>
    <Subjects>
      <Subject>
        <!-- match role subject attribute -->
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <!-- match document target namespace -->
      </Resource>
    </Resources>
  </Target>
  <Condition>
    <AttributeMatch>
      <MatchId>
        "urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto"
      </MatchId>
    </AttributeMatch>
  </Condition>
</Rule>
```
[31]  <AttributeValue>
[32]  <ResourceAttributeDesignator AttributeId=
[33]  "urn:oasis:names:tc:xacml:1.0:resource:target-name" DataTy
[34]  e="http://www.w3.org/2001/XMLSchema#string"/>
[35]  </ResourceMatch>
[36]  <!-- match requested xml element -->
[37]  <ResourceMatch>
[38]  <AttributeValue>
[39]  <ResourceAttributeDesignator AttributeId=
[40]  "urn:oasis:names:tc:xacml:1.0:resource:xpath" DataTy
[41]  e="http://www.w3.org/2001/XMLSchema#string"/>
[42]  </ResourceMatch>
[43]  </Resource>
[44]  </Resources>
[45]  <Actions>
[46]  <Action>
[47]  <!-- match 'read' action -->
[48]  <ActionMatch>
[49]  <AttributeValue>
[50]  <ResourceAttributeDesignator AttributeId=
[51]  "urn:oasis:names:tc:xacml:1.0:action:action-id" DataTy
[52]  e="http://www.w3.org/2001/XMLSchema#string"/>
[53]  </ActionMatch>
[54]  </Action>
[55]  <!-- match 'write' action -->
[56]  <ActionMatch>
[57]  <AttributeValue>
[58]  <ResourceAttributeDesignator AttributeId=
[59]  "urn:oasis:names:tc:xacml:1.0:action:action-id" DataTy
[60]  e="http://www.w3.org/2001/XMLSchema#string"/>
[61]  </ActionMatch>
[62]  </Actions>
[63]  </Target>
[64]  </Rule>

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

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


[14]-[63] Rule target. The Rule target defines a set of decision requests that are applicable to the rule. This rule is matched by:
• a decision request with subject attribute
  “urn:oasis:names:tc:xacml:1.0:examples:attribute:role” equal to
  “administrator”;

• the value of resource attribute
  “urn:oasis:names:tc:xacml:1.0:resource:target-namespace” is equal to
  “http://www.medico.com/schemas/record.xsd”

• the value of the requested XML element matches the XPath expression
  “/md:record/md:medical”;

• the value of action attribute “urn:oasis:names:tc:xacml:1.0:action:action-id” is equal to
  “read”

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

This rule does not have a Condition element.

### 4.2.4.5. Example PolicySet

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

• Either the requestor is the patient; or

• the requestor is the parent or guardian and the patient is under 16; or

• the requestor is the primary care physician and a notification is sent to the patient; and

• the requestor is not an administrator.

The following XACML <PolicySet> illustrates the combined policies. Policy 3 is included by
reference and policy 2 is explicitly included.
PolicySet declaration. Standard XML namespace declarations are included as well as PolicySetId, and policy combining algorithm identifier.

PolicySetId is used for identifying this policy set and for possible inclusion of this policy set into another policy set.

Policy combining algorithm identifier. Policies in the policy set are combined according to the specified policy combining algorithm identifier when the authorization decision is computed.

Free form description of the policy set.

PolicySet Target element defines a set of decision requests that are applicable to this PolicySet.

PolicyIdReference includes policy by id.

Policy 2 is explicitly included in this policy set.
5. Policy syntax (normative, with the exception of the schema fragments)

5.1. Element <PolicySet>

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

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

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

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

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.

```xml
<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="xs:anyURI" use="required"/>
  <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI" use="required"/>
</xs:complexType>
```

The <PolicySet> element is of PolicySetType complex type.

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

PolicySetId [Required]

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.
PolicyCombiningAlgId [Required]

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

<Description> [Optional]

A free-form description of the <PolicySet>.

PolicySetDefaults> [Optional]

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

<Target> [Required]

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

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

PolicySet> [Any Number]

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

Policy> [Any Number]

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

PolicySetIdReference> [Any Number]

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

PolicyIdReference> [Any Number]

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

<Obligations> [Optional]

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

5.2. Element <Description>

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

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

5.3. Element <PolicySetDefaults>

The <PolicySetDefaults> element SHALL specify default values that apply to the <PolicySet> element.
<xs:element name="PolicySetDefaults" type="xacml:DefaultsType"/>
<xs:complexType name="DefaultsType">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="xacml:XPathVersion" minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<PolicySetDefaults> element is of <b>DefaultsType</b> complex type.
The <PolicySetDefaults> element contains the following elements:

  <XPathVersion> [Optional]
    Default XPath version.

5.4. Element <b>XPathVersion</b>

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

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

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

5.5. Element <b>Target</b>

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

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

<xs:element name="Target" type="xacml:TargetType"/>
<xs:complexType name="TargetType">
  <xs:sequence>
    <xs:element ref="xacml:Subjects"/>
    <xs:element ref="xacml:Resources"/>
    <xs:element ref="xacml:Actions"/>
  </xs:sequence>
</xs:complexType>

The <Target> element is of <b>TargetType</b> complex type.
The <Target> element contains the following elements:

  <Subjects> [Required]
    Matching specification for the subject attributes in the context.
  <Resources> [Required]
    Matching specification for the resource attributes in the context.
5.6. Element <Subjects>

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

```
<xs:element name="Subjects" type="xacml:SubjectsType"/>
<xs:complexType name="SubjectsType"/>
  <xs:choice>
    <xs:element ref="xacml:Subject" maxOccurs="unbounded"/>
    <xs:element ref="xacml:AnySubject"/>
  </xs:choice>
</xs:complexType>
```

The <Subjects> element is of SubjectsType complex type.

5.7. Element <Subject>

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

```
<xs:element name="Subject" type="xacml:SubjectType"/>
<xs:complexType name="SubjectType"/>
  <xs:sequence>
    <xs:element ref="xacml:SubjectMatch" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

The <Subject> element is of SubjectType complex type.

5.8. Element <AnySubject>

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

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

5.9. Element <SubjectMatch>

The <SubjectMatch> element SHALL identify a set of subject-related entities by matching attribute values in a <xacml-context:Subject> element of the context with the embedded attribute value.
The `<SubjectMatch>` element is of `<SubjectMatchType>` complex type.

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

**MatchId** [Required]

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

**AttributeValue** [Required]

Embedded attribute value.

**SubjectAttributeDesignator** [Required choice]

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

**AttributeSelector** [Required choice]

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

### 5.10. Element `<Resources>`

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

The `<Resources>` element is of `<ResourcesType>` complex type.

The `<Resources>` element contains the following elements:

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

See Section 5.11.

**<AnyResource>** [Required Choice]

See Section 5.12.

### 5.11. Element `<Resource>`

The `<Resource>` element SHALL contain a *conjunctive sequence* of `<ResourceMatch>` elements.
The <Resource> element is of ResourceType complex type.

The <Resource> element contains the following elements:

<ResourceMatch> [One to Many]

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

5.12. Element <AnyResource>

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

5.13. Element <ResourceMatch>

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

The <ResourceMatch> element is of ResourceMatchType complex type.

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

MatchId [Required]

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

<AttributeValue> [Required]

Embedded attribute value.

<ResourceAttributeDesignator> [Required Choice]

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

<AttributeSelector> [Required Choice]

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

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

```xml
<x:s:element name="Actions" type="xacml:ActionsType"/>
<x:s:complexType name="ActionsType">
  <x:s:choice>
    <x:s:element ref="xacml:Action" maxOccurs="unbounded"/>
    <x:s:element ref="xacml:AnyAction"/>
  </x:s:choice>
</x:s:complexType>
```

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

The <Actions> element contains the following elements:

- <Action> [One To Many, Required Choice]
  
  See Section 5.15.

- <AnyAction> [Required Choice]
  
  See Section 5.16.

5.15. Element <Action>

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

```xml
<x:s:element name="Action" type="xacml:ActionType"/>
<x:s:complexType name="ActionType">
  <x:s:sequence>
    <x:s:element ref="xacml:ActionMatch" maxOccurs="unbounded"/>
  </x:s:sequence>
</x:s:complexType>
```

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

The <Action> element contains the following elements:

- <ActionMatch> [One to Many]

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

5.16. Element <AnyAction>

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

```xml
<x:s:element name="AnyAction"/>
```

5.17. Element <ActionMatch>

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

```xml
<x:s:element name="ActionMatch" type="xacml:ActionMatchType"/>
<x:s:complexType name="ActionMatchType">
  <x:s:sequence>
    <x:s:element ref="xacml:AttributeValue"/>
  </x:s:sequence>
</x:s:complexType>
```
The `<ActionMatch>` element is of `ActionMatchType` complex type.

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

- **MatchId [Required]**
  Specifies a matching function. The value of this attribute MUST be of type `xs:anyURI`, with legal values documented in Section A.12.

- **AttributeValue [Required]**
  Embedded attribute value.

- **ActionAttributeDesignator [Required Choice]**
  Identifies one or more attribute values in the `<Action>` element of the context.

- **AttributeSelector [Required Choice]**
  MAY be used to identify one or more attribute values in the request context. The XPath expression SHOULD resolve to an attribute in the `<Action>` element of the context.

### 5.18. Element `<PolicySetIdReference>`

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

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

### 5.19. Element `<PolicyIdReference>`

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

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

### 5.20. Element `<Policy>`

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

The main components of this element are the `<Target>`, `<Rule>` and `<Obligations>` elements and the `RuleCombiningAlgId` attribute.
The <Target> element SHALL define the applicability of the <Policy> to a set of decision requests.

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

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

```xml
<xs:element name="Policy" type="xacml:PolicyType"/>
<xs:complexType name="PolicyType">
  <xs:sequence>
    <xs:element ref="xacml:Description" minOccurs="0"/>
    <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>
    <xs:element ref="xacml:Target"/>
    <xs:element ref="xacml:Rule" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:Obligations" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
  <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>
</xs:complexType>
```

The <Policy> element is of PolicyType complex type.

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

PolicyId [Required]

Policy 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 identifier is in the form of a URL, then it MAY be resolvable.

RuleCombiningAlgId [Required]

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

Description [Optional]

A free-form description of the policy. See Section 5.2 Element Description.

PolicyDefaults [Optional]

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

Target [Required]

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

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

Rule [Any Number]

A sequence of authorizations that MUST be combined according to the RuleCombiningAlgId attribute. Rules whose <Target> elements match the decision request MUST be considered. Rules whose <Target> elements do not match the decision request SHALL be ignored.
5.21. Element <PolicyDefaults>

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

```
<xsd:element name="PolicyDefaults" type="xacml:DefaultsType"/>
<xsd:complexType name="DefaultsType">  
<xsd:sequence>
    <xsd:choice>
        <xsd:element ref="xacml:XPathVersion" minOccurs="0"/>
    </xsd:choice>
</xsd:sequence>
</xsd:complexType>
```

The <PolicyDefaults> element is of DefaultsType complex type.

The <PolicyDefaults> element contains the following elements:

- XPathVersion [Optional]
  Default XPath version.

5.22. Element <Rule>

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

```
<xsd:element name="Rule" type="xacml:RuleType"/>
<xsd:complexType name="RuleType">  
<xsd:sequence>
    <xsd:element ref="xacml:Description" minOccurs="0"/>
    <xsd:element ref="xacml:Target" minOccurs="0"/>
    <xsd:element ref="xacml:Condition" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
```

The <Rule> element is of RuleType complex type.

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

- RuleId [Required]
  A URN identifying this rule.
- Effect [Required]
  Rule effect. Values of this attribute are either “Permit” or “Deny”.
- Description [Optional]
  A free-form description of the rule.
Identifies the set of decision requests that the <Rule> element is intended to evaluate. If this element is omitted, then the target for the <Rule> SHALL be defined by the <Target> element of the enclosing <Policy> element. See Section 5.5 for details.

5.23. Simple type EffectType

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

```xml
<xsd:simpleType name="EffectType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Permit"/>
    <xsd:enumeration value="Deny"/>
  </xsd:restriction>
</xsd:simpleType>
```

5.24. Element <Condition>

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

```xml
<xsd:element name="Condition" type="xacml:ApplyType"/>
```

5.25. Element <Apply>

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

```xml
<xsd:complexType name="ApplyType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element ref="xacml:Function"/>
    <xsd:element ref="xacml:Apply"/>
    <xsd:element ref="xacml:SubjectAttributeDesignator"/>
    <xsd:element ref="xacml:ResourceAttributeDesignator"/>
    <xsd:element ref="xacml:ActionAttributeDesignator"/>
    <xsd:element ref="xacml:EnvironmentAttributeDesignator"/>
    <xsd:element ref="xacml:AttributeSelector"/>
  </xsd:choice>
  <xsd:attribute name="FunctionId" type="xsd:anyURI" use="required"/>
</xsd:complexType>
```

The <Apply> element is of ApplyType complex type.

The <Apply> element contains the following attributes and elements:
FunctionId [Required]
The URN of a function. XACML-defined functions are described in Appendix A.

<Function> [Optional]
The name of a function that is applied to the elements of a bag. See Section A14.11.

<Apply> [Optional]
A nested function-call argument.

<AttributeValue> [Optional]
A literal value argument.

<SubjectAttributeDesignator> [Optional]
A subject attribute argument.

<ResourceAttributeDesignator> [Optional]
A resource attribute argument.

>ActionAttributeDesignator> [Optional]
An action attribute argument.

<EnvironmentAttributeDesignator> [Optional]
An environment attribute argument.

<AttributeSelector> [Optional]
An attribute selector argument.

5.26. Element <Function>
The Function element SHALL be used to name a function that is applied by the higher-order bag functions to every element of a bag. The higher-order bag functions are described in Section A14.11.

```
<x:s:element name="Function" type="xacml:FunctionType"/>
<x:s:complexType name="FunctionType">
  <x:s:attribute name="FunctionId" type="xs:anyURI" use="required"/>
</x:s:complexType>
```

The Function element is of FunctionType complex type.

The Function element contains the following attributes:

FunctionId [Required]
The identifier for the function that is applied to the elements of a bag by the higher-order bag functions.

5.27. Complex type AttributeDesignatorType
The AttributeDesignatorType complex type is the type for elements and extensions that identify attributes. An element of this type contains properties by which it MAY be matched to attributes in the request context.
In addition, elements of this type MAY control behaviour in the event that no matching attribute is present in the context.

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

```
<x:s:complexType name="AttributeDesignatorType">
  <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
  <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
  <xs:attribute name="Issuer" type="xs:string" use="optional"/>
  <xs:attribute name="MustBePresent" type="xs:boolean" use="optional" default="false"/>
</x:s:complexType>
```

A named attribute SHALL match an attribute if the values of their respective AttributeId, DataType and Issuer attributes match. The attribute designator’s AttributeId MUST match, by URI equality, the AttributeId of the attribute. The attribute designator’s DataType MUST match, by URI equality, the DataType of the same attribute.

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

The <AttributeDesignatorType> contains the following attributes:

- **AttributeId [Required]**
  - This attribute SHALL specify the AttributeId with which to match the attribute.

- **DataType [Required]**
  - This attribute SHALL specify the data-type with which to match the attribute.

- **Issuer [Optional]**
  - This attribute, if supplied, SHALL specify the Issuer with which to match the attribute.

- **MustBePresent [Optional]**
  - This attribute governs whether the element returns “Indeterminate” in the case where the named attribute is absent. If the named attribute is absent and MustBePresent is “True”, then this element SHALL result in “Indeterminate”. The default value SHALL be “False”.

5.28. Element <SubjectAttributeDesignator>

The <SubjectAttributeDesignator> element is of the SubjectAttributeDesignatorType.

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

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

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

A subject attribute is an attribute of a particular subject, i.e. contained within a <Subject> element.
A named subject attribute is a named attribute for a subject.

A named categorized subject attribute is a named subject attribute for a particular categorized subject.

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

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

Elements and extensions of the SubjectAttributeDesignatorType complex type determine the presence of select attribute values associated with named categorized subject attributes.

Elements and extensions of the SubjectAttributeDesignatorType SHALL NOT alter the match semantics of named categorized subject attributes, but MAY narrow the search space.

```
<xs:complexType name="SubjectAttributeDesignatorType">
  <xs:complexContent>
    <xs:extension base="xacml:AttributeDesignatorType">
      <xs:attribute name="SubjectCategory"
        type="xs:anyURI"
        use="optional"
        default="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

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

SubjectCategory [Optional]

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

5.29. Element <ResourceAttributeDesignator>

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

The <ResourceAttributeDesignator> element SHALL return a bag containing all the resource attribute values that are matched by the named resource attribute. The MustBePresent attribute governs whether this element returns an empty bag or "Indeterminate" in the case that the named resource attribute is absent. If the named resource attribute is not present and the MustBePresent attribute is "False" (its default value), then this element SHALL evaluate to an empty bag. If the named resource attribute is not present and the MustBePresent attribute is "True", then this element SHALL evaluate to "Indeterminate".

Regardless of the MustBePresent attribute, if it cannot be determined whether the named
resource attribute is present or not in the request context, or the value of the named resource attribute is unavailable, then the expression SHALL evaluate to “Indeterminate”. A named resource attribute SHALL match a resource attribute as per the match semantics specified in the AttributeDesignatorType complex type [Section 5.27].

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

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

5.30. Element <ActionAttributeDesignator>

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

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

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

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

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

5.31. Element <EnvironmentAttributeDesignator>

The <EnvironmentAttributeDesignator> element retrieves a bag of values for a named environment attribute. An environment attribute is an attribute contained within the <Environment> element of the <xacml-context:Request> element. A named environment attribute has specific criteria (described below) with which to match an environment attribute. A named environment attribute SHALL be considered present, if there is at least one environment attribute that matches the criteria. An environment attribute value is an attribute value that is contained within an environment attribute.
The `<EnvironmentAttributeDesignator>` element SHALL evaluate to a bag of all the environment attribute values that are matched by the named environment attribute. The MustBePresent attribute governs whether this element returns an empty bag or “Indeterminate” in the case that the named environment attribute is absent. If the named environment attribute is not present and the MustBePresent attribute is “False” (its default value), then this element SHALL evaluate to an empty bag. If the named environment attribute is not present and the MustBePresent attribute is “True”, then this element SHALL evaluate to “Indeterminate”.

Regardless of the MustBePresent attribute, if it cannot be determined whether the named environment attribute is present or not present in the request context, or the value of the named environment attribute is unavailable, then the expression SHALL evaluate to “Indeterminate”.

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

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

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

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

### 5.32. Element `<AttributeSelector>`

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

```xml
  xs:string()
  xs:boolean()
  xs:integer()
  xs:double()
  xs:dateTime()
  xs:date()
  xs:time()
  xs:hexBinary()
  xs:base64Binary()
  xs:anyURI()
  xf:yearMonthDuration()
  xf:dayTimeDuration()
```

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

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

If the selected node is different from the node types listed above (a text node, an attribute node, a processing instruction node, or a comment node), then the result of that policy SHALL be "Indeterminate" with a StatusCode value of "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

Support for the <AttributeSelector> element is OPTIONAL.

The <AttributeSelector> element is of AttributeSelectorType complex type.

The <AttributeSelector> element has the following attributes:

- RequestContextPath [Required]
  - An XPath expression whose context node is the <xacml-context:Request> element.
  - There SHALL be no restriction on the XPath syntax.

- DataType [Required]
  - The bag of values returned by the AttributeSelector SHALL be of this data type.

- MustBePresent [Optional]
  - Whether or not the designated attribute must be present in the context. If the XPath expression selects no node, and the MustBePresent attribute is TRUE, then the result SHALL be "Indeterminate" and the status code SHALL be "urn:oasis:names:tc:xacml:1.0:status:missing-attribute". If the XPath expression selects no node, and the MustBePresent attribute is missing or FALSE, then the result SHALL be an empty bag. If the XPath expression selects at least one node and the selected node(s) could be successfully converted to a bag of values of the specified data-type, then the result SHALL be the bag, regardless of the value of the MustBePresent attribute. If the XPath expression selects at least one node, but there is an error in converting one or more of the nodes to values of the specified data-type, then the result SHALL be "Indeterminate" and the status code SHALL be "urn:oasis:names:tc:xacml:1.0:status:processing-error", regardless of the value of the MustBePresent attribute.

### 5.33. Element <AttributeValue>

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

```xml
<x:s:element name="AttributeValue" type="xacml:AttributeValueType"/>
<x:s:complexType name="AttributeValueType" mixed="true">
  <x:s:sequence>
    <x:s:element minOccurs="0" maxOccurs="unbounded">
      <x:s:complexType name="AttributeValueType">" lax" minOccurs="0">
    </x:s:sequence>
  </x:s:complexType>
</x:s:complexType>
```
5.34. Element <Obligations>

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

Support for the <Obligations> element is OPTIONAL.

```
<xs:element name="Obligations" type="xacml:ObligationsType"/>
<xs:complexType name="ObligationsType">
  <xs:sequence>
    <xs:element ref="xacml:Obligation" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

The <Obligations> element is of ObligationsType complexType.

The <Obligations> element contains the following element:

- A sequence of obligations

5.35. Element <Obligation>

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

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

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

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

- ObligationId [Required]
- Obligation identifier. The value of the obligation identifier SHALL be interpreted by the
  PEP.
- FulfillOn [Required]
- The effect for which this obligation applies.
- <AttributeAssignment> [One To Many]
- Obligation arguments assignment. The values of the obligation arguments SHALL be
  interpreted by the PEP.
5.36. Element <AttributeAssignment>

The <AttributeAssignment> element SHALL contain an AttributeId and the corresponding attribute value. The AttributeId is part of attribute meta-data, and is used when the attribute cannot be referenced by its location in the <xacml-context:Request>. This situation may arise in an <Obligation> element if the obligation includes parameters. The <AttributeAssignment> element MAY be used in any way consistent with the schema syntax, which is a sequence of “any”. The value specified SHALL be understood by the PEP, but it is not further specified by XACML. See section 7.11 “Obligations”.

```xml
<x:s:element name="AttributeAssignment"
    type="xacml:AttributeAssignmentType"/>
<x:s:complexType name="AttributeAssignmentType" mixed="true">
    <x:s:extension base="xacml:AttributeValueType">
        <x:s:attribute name="AttributeId" type="xs:anyURI" use="required"/>
    </x:s:extension>
</x:s:complexType>
```

The <AttributeAssignment> element is of AttributeAssignmentType complex type.

The <AttributeAssignment> element contains the following attributes:

- **AttributeId** [Required]

The attribute Identifier

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

6.1. Element <Request>

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 decision request into the form of an XACML context <Request>.

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

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

The <Request> element is of RequestType complex type.

The <Request> element contains the following elements:
<Subject> [One to Many]

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

<Resource> [Required]

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

<Action> [Required]

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

<Environment> [Optional]

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

6.2. Element <Subject>

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

```xml
<xs:element name="Subject" type="xacml-context:SubjectType"/>
<xs:complexType name="SubjectType">
  <xs:sequence>
    <xs:element ref="xacml-context:Attribute" minOccurs="0"
               maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

The <Subject> element is of SubjectType complex type.

The <Subject> element contains the following elements:

SubjectCategory [Optional]

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

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

<Attribute> [Any Number]
A sequence of attributes that apply to the subject.

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

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

### 6.3. Element <Resource>

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

```xml
<x:s:element name="Resource" type="xacml-context:ResourceType"/>
<x:s:complexType name="ResourceType">
  <x:s:sequence>
    <x:s:element ref="xacml-context:ResourceContent" minOccurs="0" maxOccurs="unbounded"/>
  </x:s:sequence>
</x:s:complexType>
```

The <Resource> element is of ResourceType complex type.

The <Resource> element contains the following elements:

- <ResourceContent> [Optional]
  
  The resource content.

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

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

### 6.4. Element <ResourceContent>

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

```xml
<x:s:complexType name="ResourceContentType" mixed="true">
  <x:s:sequence>
    <x:s:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </x:s:sequence>
</x:s:complexType>
```

The <ResourceContent> element is of ResourceContentType complex type.

The <ResourceContent> element allows arbitrary elements and attributes.
6.5. Element <Action>

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

```
<x:s:element name="Action" type="xacml-context:ActionType"/>
<x:s:complexType name="ActionType">
  <x:s:sequence>
    <x:s:element ref="xacml-context:Attribute" minOccurs="0" maxOccurs="unbounded"/>
  </x:s:sequence>
</x:s:complexType>
```

The <Action> element is of ActionType complex type.

The <Action> element contains the following elements:

- <Attribute> [Any Number]
  - List of attributes of the action to be performed on the resource.

6.6. Element <Environment>

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

```
<x:s:element name="Environment" type="xacml-context:EnvironmentType"/>
<x:s:complexType name="EnvironmentType">
  <x:s:sequence>
    <x:s:element ref="xacml-context:Attribute" minOccurs="0" maxOccurs="unbounded"/>
  </x:s:sequence>
</x:s:complexType>
```

The <Environment> element is of EnvironmentType complex type.

The <Environment> element contains the following elements:

- <Attribute> [Any Number]
  - A list of environment attributes. Environment attributes are attributes that are not associated with either the resource, the action or any of the subjects of the access request.

6.7. Element <Attribute>

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

```
<x:s:element name="Attribute" type="xacml-context:AttributeType"/>
<x:s:complexType name="AttributeType">
  <x:s:sequence>
    <x:s:element ref="xacml-context:AttributeValue"/>
  </x:s:sequence>
</x:s:complexType>
```

```
<x:s:element name="AttributeId" type="xs:anyURI" use="required"/>
<x:s:element name="DataType" type="xs:anyURI" use="required"/>
<x:s:element name="Issuer" type="xs:string" use="optional"/>
```
The `<Attribute>` element is of `AttributeType` complex type.

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

- **AttributeId** [Required]
  - `Attribute` identifier. A number of identifiers are reserved by XACML to denote commonly used `attributes`.

- **DataType** [Required]
  - The data-type of the contents of the `<AttributeValue>` element. This SHALL be either a primitive type defined by the XACML 1.0 specification or a type defined in a namespace declared in the `<xacml-context>` element.

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

- **IssueInstant** [Optional]
  - The date and time at which the `attribute` was issued.

- **<AttributeValue>** [Required]
  - Exactly one `attribute` value. The mandatory `attribute` value MAY have contents that are empty, occur once, or occur multiple times.

### 6.8. Element `<AttributeValue>`

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

The `<AttributeValue>` element is of `AttributeValueType` type.

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

### 6.9. Element `<Response>`

The `<Response>` element is a top-level element in the XACML `context` schema. The `<Response>` element is an abstraction layer used by the `policy` language. Any proprietary system using the XACML specification MUST transform an XACML `context` `<Response>` into the form of its `authorization decision`. 
The <Response> element encapsulates the **authorization decision** produced by the PDP. It includes a sequence of one or more results, with one <Result> element per requested resource. Multiple results MAY be returned when the value of the “urn:oasis:xacml:1.0:resource:scope” resource attribute in the request context is “Descendants” or “Children”. Support for multiple results is OPTIONAL.

```
<x:s:element name="Response" type="xacml-context:ResponseType"/>
<x:s:complexType name="ResponseType">
    <x:s:element ref="xacml-context:Result" maxOccurs="unbounded"/>
</x:s:complexType>
```

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

The <Response> element contains the following elements:

- <Result> [One to Many]

  An authorization decision result.

### 6.10. Element <Result>

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

```
<x:s:element name="Result" type="xacml-context:ResponseType"/>
<x:s:complexType name="ResultType">
    <x:s:sequence>
        <x:s:element ref="xacml-context:Decision"/>
        <x:s:element ref="xacml-context:Status"/>
        <x:s:element ref="xacml:Obligations" minOccurs="0"/>
    </x:s:sequence>
    <x:s:attribute name="ResourceId" type="xs:string" use="optional"/>
</x:s:complexType>
```

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

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

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

- **<Decision>** [Required]
  - The **authorization decision**: “Permit”, “Deny”, “Indeterminate” or “NotApplicable”.

- **<Status>** [Required]
  - Indicates whether errors occurred during evaluation of the decision request, and optionally, information about those errors.

- **<xacml:Obligations>** [Optional]
A list of obligations that MUST be fulfilled by the PEP. If the PEP does not understand an obligation, then it MUST act as if the PDP had denied access to the requested resource. See Section 7.11 for a description of how the set of obligations to be returned by the PDP is determined.

6.11. Element <Decision>

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

```xml
<x:s:element name="Decision" type="xacml-context:DecisionType"/>
<x:s:complexType name="DecisionType">
  <x:s:restriction base="xs:string">
    <x:s:enumeration value="Permit"/>
    <x:s:enumeration value="Deny"/>
    <x:s:enumeration value="Indeterminate"/>
    <x:s:enumeration value="NotApplicable"/>
  </x:s:restriction>
</x:s:complexType>
```

The <Decision> element is of DecisionType simple type.

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

- "Permit": the requested access is permitted.
- "Deny": the requested access is denied.
- "Indeterminate": the PDP is unable to evaluate the requested access. Reasons for such inability include: missing attributes, network errors while retrieving policies, division by zero during policy evaluation, syntax errors in the decision request or in the policy, etc..
- "NotApplicable": the PDP does not have any policy that applies to this decision request.

6.12. Element <Status>

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

```xml
<x:s:element name="Status" type="xacml-context:StatusType"/>
<x:s:complexType name="StatusType">
  <x:s:sequence>
    <x:s:element ref="xacml-context:StatusCode"/>
    <x:s:element ref="xacml-context:StatusMessage" minOccurs="0"/>
    <x:s:element ref="xacml-context:StatusDetail" minOccurs="0"/>
  </x:s:sequence>
</x:s:complexType>
```

The <Status> element is of StatusType complex type.

The <Status> element contains the following elements:

- <StatusCode> [Required]
  - Status code.
- <StatusMessage> [Optional]
  - A status message describing the status code.
- <StatusDetail> [Optional]
  - Additional status information.
6.13. Element <StatusCode>

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

```xml
<xs:element name="StatusCode" type="xacml-context:StatusCodeType"/>
<xs:complexType name="StatusCodeType">
  <xs:sequence>
    <xs:element ref="xacml-context:StatusCode" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="Value" type="xs:anyURI" use="required"/>
</xs:complexType>
```

The <StatusCode> element is of StatusCodeType complex type.

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

Value [Required]

See Section B.9 for a list of values.

<StatusCode> [Any Number]

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


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

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

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

6.15. Element <StatusDetail>

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

```xml
<xs:element name="StatusDetail" type="xacml-context:StatusDetailType"/>
<xs:complexType name="StatusDetailType">
  <xs:sequence>
    <xs:element any namespace="#any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

The <StatusDetail> element is of StatusDetailType complex type.

The <StatusDetail> element allows arbitrary XML content.

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

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

A PDP MUST NOT return a <StatusDetail> element in conjunction with the "ok" status value.

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

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

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

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

7. Functional requirements (normative)

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

7.1. Policy enforcement point

This section describes the requirements for the PEP.

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

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

7.2. Base policy

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

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

The target value SHALL be "Match" if the subject, resource and action specified in the target all match values in the request context. The target value SHALL be "No-match" if one or more of the subject, resource and action specified in the target do not match values in the request context.

The value of a <SubjectMatch>, <ResourceMatch> or <ActionMatch> element, in which a referenced attribute value cannot be obtained, depends on the value of the MustBePresent attribute of the <AttributeDesignator> or <AttributeSelector> element. If the MustBePresent attribute is "True", then the result of the <SubjectMatch>, <ResourceMatch> or <ActionMatch> element SHALL be "Indeterminate" in this case. If the MustBePresent attribute is "False" or missing, then the result of the <SubjectMatch>, <ResourceMatch> or <ActionMatch> element SHALL be "No-match".

7.4. Condition evaluation

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

7.5. Rule evaluation

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

<table>
<thead>
<tr>
<th>Target</th>
<th>Condition</th>
<th>Rule Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Match&quot;</td>
<td>&quot;True&quot;</td>
<td>Effect</td>
</tr>
<tr>
<td>&quot;Match&quot;</td>
<td>&quot;False&quot;</td>
<td>&quot;NotApplicable&quot;</td>
</tr>
<tr>
<td>&quot;Match&quot;</td>
<td>&quot;Indeterminate&quot;</td>
<td>&quot;Indeterminate&quot;</td>
</tr>
<tr>
<td>&quot;No-match&quot;</td>
<td>Don’t care</td>
<td>&quot;NotApplicable&quot;</td>
</tr>
<tr>
<td>&quot;Indeterminate&quot;</td>
<td>Don’t care</td>
<td>&quot;Indeterminate&quot;</td>
</tr>
</tbody>
</table>

Table 1 - Rule truth table

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

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

7.6. Policy evaluation

The value of a policy SHALL be determined only by its contents, considered in relation to the contents of the request context. A policy's value SHALL be determined by evaluation of the policy's target and rules, according to the specified rule-combining algorithm.
The policy's target SHALL be evaluated to determine the applicability of the policy. If the target evaluates to "Match", then the value of the policy SHALL be determined by evaluation of the policy's rules, according to the specified rule-combining algorithm. If the target evaluates to "No-match", then the value of the policy SHALL be "NotApplicable". If the target evaluates to "Indeterminate", then the value of the policy SHALL be "Indeterminate".

The policy truth table is shown in Table 2.

<table>
<thead>
<tr>
<th>Target</th>
<th>Rule values</th>
<th>Policy Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Match&quot;</td>
<td>At least one rule value is its Effect</td>
<td>Specified by the rule-combining algorithm</td>
</tr>
<tr>
<td>&quot;Match&quot;</td>
<td>All rule values are &quot;NotApplicable&quot;</td>
<td>&quot;NotApplicable&quot;</td>
</tr>
<tr>
<td>&quot;Match&quot;</td>
<td>At least one rule value is &quot;Indeterminate&quot;</td>
<td>Specified by the rule-combining algorithm</td>
</tr>
<tr>
<td>&quot;No-match&quot;</td>
<td>Don’t-care</td>
<td>&quot;NotApplicable&quot;</td>
</tr>
<tr>
<td>&quot;Indeterminate&quot;</td>
<td>Don’t-care</td>
<td>&quot;Indeterminate&quot;</td>
</tr>
</tbody>
</table>

Table 2 - Policy truth table

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

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

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

7.7. Policy Set evaluation

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

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

The policy set truth table is shown in Table 3.
Table 3 – Policy set truth table

<table>
<thead>
<tr>
<th>Target</th>
<th>Policy values</th>
<th>Policy Set Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match</td>
<td>At least one policy value is its <strong>Decision</strong></td>
<td>Specified by the <strong>policy-combining algorithm</strong></td>
</tr>
<tr>
<td>Match</td>
<td>All policy values are “NotApplicable”</td>
<td>“NotApplicable”</td>
</tr>
<tr>
<td>Match</td>
<td>At least one policy value is “Indeterminate”</td>
<td>Specified by the <strong>policy-combining algorithm</strong></td>
</tr>
<tr>
<td>“No-match”</td>
<td>Don’t-care</td>
<td>“NotApplicable”</td>
</tr>
<tr>
<td>Indeterminate</td>
<td>Don’t-care</td>
<td>“Indeterminate”</td>
</tr>
</tbody>
</table>

2939

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

2949 If the **target** value is "No-match" or "Indeterminate" then the **policy set** value SHALL be
2950 "NotApplicable" or “Indeterminate”, respectively, regardless of the value of the **policies**. For these
2951 cases, therefore, the **policies** need not be evaluated in order to determine the **policy set** value.
2952 If the **target** value is “Match” and the **policies** value is “At least one policy value is it’s **Decision**” or
2953 “At least one policy value is ‘Indeterminate’”, then the **policy-combining algorithm** specified in the
2954 **policy set** SHALL determine the **policy set** value.

2956 7.8. Hierarchical resources

2957 It is often the case that a **resource** is organized as a hierarchy (e.g. file system, XML document).
2958 Some access requesters may request **access** to an entire subtree of a **resource** specified by a
2959 node. XACML allows the **PEP** (or **context handler**) to specify whether the **decision request** is
2960 just for a single **resource** or for a subtree below the specified **resource**. The latter is equivalent to
2961 repeating a single request for each node in the entire subtree. When a request **context** contains a
2962 resource attribute of type
2963
2964 "urn:oasis:names:tc:xacml:1.0:resource:scope"
2965
2966 with a value of "Immediate", or if it does not contain that **attribute**, then the **decision request**
2967 SHALL be interpreted to apply to just the single **resource** specified by the
2968 “urn:oasis:names:tc:xacml:1.0:resource:resource-id” **attribute**.
2969
2970 When the
2971
2972 "urn:oasis:names:tc:xacml:1.0:resource:scope"
attribute has the value "Children", the decision request SHALL be interpreted to apply to the
specified resource and its immediate children resources.

When the
"urn:oasis:names:tc:xacml:1.0:resource:scope"
attribute has the value "Descendants", the decision request SHALL be interpreted to apply to
both the specified resource and all its descendant resources.

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

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

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

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

7.9. Attributes

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

7.9.1. Attribute Matching

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

7.9.2. Attribute Retrieval

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

7.9.3. Environment Attributes

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

7.10. Authorization decision

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

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

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

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

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

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

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

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

7.11. Obligations

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

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

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

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

7.12. Unsupported functionality

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

7.13. Syntax and type errors

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

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

8. XACML extensibility points (non-normative)

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

8.1. Extensible XML attribute types

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

AttributeId,
AttributeValue,
DataType,
FunctionId,
MatchId,
ObligationId,
PolicyCombiningAlgId,
RuleCombiningAlgId,
StatusCode,
SubjectCategory.
See Section 5 for definitions of these attribute types.

8.2. Structured attributes

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

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.

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.

9. Security and privacy considerations (non-normative)

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.

9.1. Threat model

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.

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.

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.

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

9.1.1. Unauthorized disclosure

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

Unauthorized disclosure is addressed by confidentiality mechanisms.

9.1.2. Message replay

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

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

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

9.1.3. Message insertion

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

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

9.1.4. Message deletion

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

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

9.1.5. Message modification

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

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

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

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

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

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

9.1.7. Negative rules

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

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

If not used with care, negative rules can lead to policy violation in two common cases. They are: when attributes are suppressed and when the base group changes. An example of suppressed attributes would be if we have a policy that access should be permitted, unless the subject is a credit risk. If it is possible that the attribute of being a credit risk may be unknown to the PDP for some reason, then unauthorized access may be permitted. In some environments, the subject may be able to suppress the publication of attributes by the application of privacy controls, or the server or repository that contains the information may be unavailable for accidental or intentional reasons.
An example of a changing base group would be if there is a policy that everyone in the engineering department may change software source code, except for secretaries. Suppose now that the department was to merge with another engineering department and the intent is to maintain the same policy. However, the new department also includes individuals identified as administrative assistants, who ought to be treated in the same way as secretaries. Unless the policy is altered, they will unintentionally be permitted to change software source code. Problems of this type are easy to avoid when one individual administers all policies, but when administration is distributed, as XACML allows, this type of situation must be explicitly guarded against.

9.2. Safeguards

9.2.1. Authentication

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

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

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

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

9.2.2. Policy administration

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

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

9.2.3. Confidentiality

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

9.2.3.1. Communication confidentiality

In some environments it is deemed good practice to treat all data within an access control system as confidential. In other environments, policies may be made freely available for distribution, inspection and audit. The idea behind keeping policy information secret is to make it more difficult for an adversary to know what steps might be sufficient to obtain unauthorized access. Regardless of the approach chosen, the security of the access control system should not depend on the secrecy of the policy.
Any security concerns or requirements related to transmitting or exchanging XACML `<Policy>` elements are outside the scope of the XACML standard. While it is often important to ensure that the integrity and confidentiality of `<Policy>` elements is maintained when they are exchanged between two parties, it is left to the implementers to determine the appropriate mechanisms for their environment.

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

### 9.2.3.2. Statement level confidentiality

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

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

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

### 9.2.4. Policy integrity

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

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

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

### 9.2.5. Policy identifiers

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

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

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

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

9.2.7. Privacy

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

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

10. Conformance (normative)

10.1. Introduction

The XACML specification addresses the following aspect of conformance:

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

10.2. Conformance tables

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


### 10.2.1. Schema elements

The implementation MUST support those schema elements that are marked “M”.

<table>
<thead>
<tr>
<th>Element name</th>
<th>M/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>xacml-context:Action</td>
<td>M</td>
</tr>
<tr>
<td>xacml-context:Attribute</td>
<td>M</td>
</tr>
<tr>
<td>xacml-context:AttributeValue</td>
<td>M</td>
</tr>
<tr>
<td>xacml-context:Decision</td>
<td>M</td>
</tr>
<tr>
<td>xacml-context:Environment</td>
<td>M</td>
</tr>
<tr>
<td>xacml-context:Obligations</td>
<td>O</td>
</tr>
<tr>
<td>xacml-context:Request</td>
<td>M</td>
</tr>
<tr>
<td>xacml-context:Resource</td>
<td>M</td>
</tr>
<tr>
<td>xacml-context:ResourceContent</td>
<td>O</td>
</tr>
<tr>
<td>xacml-context:Response</td>
<td>M</td>
</tr>
<tr>
<td>xacml-context:Result</td>
<td>M</td>
</tr>
<tr>
<td>xacml-context:Status</td>
<td>M</td>
</tr>
<tr>
<td>xacml-context:StatusCode</td>
<td>M</td>
</tr>
<tr>
<td>xacml-context:StatusDetail</td>
<td>O</td>
</tr>
<tr>
<td>xacml-context:StatusMessage</td>
<td>O</td>
</tr>
<tr>
<td>xacml-context:Subject</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Action</td>
<td>M</td>
</tr>
<tr>
<td>xacml:ActionAttributeDesignator</td>
<td>M</td>
</tr>
<tr>
<td>xacml:ActionMatch</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Actions</td>
<td>M</td>
</tr>
<tr>
<td>xacml:AnyAction</td>
<td>M</td>
</tr>
<tr>
<td>xacml:AnyResource</td>
<td>M</td>
</tr>
<tr>
<td>xacml:AnySubject</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Apply</td>
<td>M</td>
</tr>
<tr>
<td>xacml:AttributeAssignment</td>
<td>O</td>
</tr>
<tr>
<td>xacml:AttributeSelector</td>
<td>O</td>
</tr>
<tr>
<td>xacml:AttributeValue</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Condition</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Description</td>
<td>M</td>
</tr>
<tr>
<td>xacml:EnvironmentAttributeDesignator</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Function</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Obligation</td>
<td>O</td>
</tr>
<tr>
<td>xacml:Obligations</td>
<td>O</td>
</tr>
<tr>
<td>xacml:Policy</td>
<td>M</td>
</tr>
<tr>
<td>xacml:PolicyDefaults</td>
<td>O</td>
</tr>
<tr>
<td>xacml:PolicyIdReference</td>
<td>M</td>
</tr>
<tr>
<td>xacml:PolicySet</td>
<td>M</td>
</tr>
<tr>
<td>xacml:PolicySetDefaults</td>
<td>O</td>
</tr>
<tr>
<td>xacml:PolicySetIdReference</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Resource</td>
<td>M</td>
</tr>
<tr>
<td>xacml:ResourceAttributeDesignator</td>
<td>M</td>
</tr>
<tr>
<td>xacml:ResourceMatch</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Resources</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Rule</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Subject</td>
<td>M</td>
</tr>
<tr>
<td>xacml:SubjectMatch</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Subjects</td>
<td>M</td>
</tr>
</tbody>
</table>


10.2.2. Identifier Prefixes

The following identifier prefixes are reserved by XACML.

<table>
<thead>
<tr>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn:oasis:names:tc:xacml:1.0</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:conformance-test</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:context</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:example</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:policy</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:resource</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:action</td>
</tr>
</tbody>
</table>

10.2.3. Algorithms

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>M/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:rule-combining</td>
<td></td>
</tr>
<tr>
<td>algorithm:deny-overrides</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:policy-combining</td>
<td></td>
</tr>
<tr>
<td>algorithm:deny-overrides</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:rule-combining</td>
<td></td>
</tr>
<tr>
<td>algorithm:permit-overrides</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:policy-combining</td>
<td></td>
</tr>
<tr>
<td>algorithm:permit-overrides</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:rule-combining</td>
<td></td>
</tr>
<tr>
<td>algorithm:first-applicable</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:policy-combining</td>
<td></td>
</tr>
<tr>
<td>algorithm:first-applicable</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:policy-combining</td>
<td></td>
</tr>
<tr>
<td>algorithm:only-one-applicable</td>
<td></td>
</tr>
</tbody>
</table>

10.2.4. Status Codes

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>M/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:status:missing-attribute</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:status:ok</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:status:processing-error</td>
<td></td>
</tr>
</tbody>
</table>

oasis-#####-xacml-1.1.pdf 86
10.2.5. Attributes

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>M/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:environment:current-time</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:environment:current-date</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:environment:current-dateTime</td>
<td>M</td>
</tr>
</tbody>
</table>

10.2.6. Identifiers

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>M/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:authentication-method</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:authentication-time</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:authentication-id</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:session-start-time</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:subject-id</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:category:access-subject</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:category:codebase</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:category:intermediary-subject</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:category:recipient-subject</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:category:requesting-subject</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:resource:resource-location</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:resource:resource-id</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:resource:scope</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:resource:simple-file-name</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:action:action-id</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:action:implied-action</td>
<td>M</td>
</tr>
</tbody>
</table>

10.2.7. Data-types

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

<table>
<thead>
<tr>
<th>Data-type</th>
<th>M/O</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.w3.org/2001/XMLSchema#string">http://www.w3.org/2001/XMLSchema#string</a></td>
<td>M</td>
</tr>
<tr>
<td><a href="http://www.w3.org/2001/XMLSchema#boolean">http://www.w3.org/2001/XMLSchema#boolean</a></td>
<td>M</td>
</tr>
<tr>
<td><a href="http://www.w3.org/2001/XMLSchema#integer">http://www.w3.org/2001/XMLSchema#integer</a></td>
<td>M</td>
</tr>
<tr>
<td><a href="http://www.w3.org/2001/XMLSchema#double">http://www.w3.org/2001/XMLSchema#double</a></td>
<td>M</td>
</tr>
<tr>
<td><a href="http://www.w3.org/2001/XMLSchema#time">http://www.w3.org/2001/XMLSchema#time</a></td>
<td>M</td>
</tr>
<tr>
<td><a href="http://www.w3.org/2001/XMLSchema#date">http://www.w3.org/2001/XMLSchema#date</a></td>
<td>M</td>
</tr>
<tr>
<td><a href="http://www.w3.org/2001/XMLSchema#dateTime">http://www.w3.org/2001/XMLSchema#dateTime</a></td>
<td>M</td>
</tr>
<tr>
<td><a href="http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration">http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration</a></td>
<td>M</td>
</tr>
</tbody>
</table>
### 10.2.8. Functions

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

<table>
<thead>
<tr>
<th>Function</th>
<th>M/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:string-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:boolean-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:integer-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:double-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:date-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:time-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dateTime-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:anyURI-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:x500Name-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:integer-add</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:double-add</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:integer-subtract</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:double-subtract</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:integer-multiply</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:double-multiply</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:integer-divide</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:double-divide</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:integer-mod</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:integer-abs</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:double-nums</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:round</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:floor</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:string-normalize-space</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:double-to-integer</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:integer-to-double</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:or</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:and</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:n-of</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:not</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:present</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:integer-greater-than</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:integer-less-than</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:double-greater-than</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:double-less-than</td>
<td>M</td>
</tr>
</tbody>
</table>
urn:oasis:names:tc:xacml:1.0:function:hexBinary-one-and-only
urn:oasis:names:tc:xacml:1.0:function:hexBinary-bag-size
urn:oasis:names:tc:xacml:1.0:function:hexBinary-is-in
urn:oasis:names:tc:xacml:1.0:function:hexBinary-bag
urn:oasis:names:tc:xacml:1.0:function:base64Binary-one-and-only
urn:oasis:names:tc:xacml:1.0:function:base64Binary-bag-size
urn:oasis:names:tc:xacml:1.0:function:base64Binary-is-in
urn:oasis:names:tc:xacml:1.0:function:base64Binary-bag
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-one-and-only
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-bag-size
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-is-in
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-one-and-only
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-bag-size
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-is-in
urn:oasis:names:tc:xacml:1.0:function:x500Name-one-and-only
urn:oasis:names:tc:xacml:1.0:function:x500Name-bag-size
urn:oasis:names:tc:xacml:1.0:function:x500Name-is-in
urn:oasis:names:tc:xacml:1.0:function:x500Name-bag
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-one-and-only
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-bag-size
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-is-in
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-bag
urn:oasis:names:tc:xacml:1.0:function:any-of
urn:oasis:names:tc:xacml:1.0:function:any-of-any
urn:oasis:names:tc:xacml:1.0:function:any-of-all
urn:oasis:names:tc:xacml:1.0:function:all-of
urn:oasis:names:tc:xacml:1.0:function:all-of-any
urn:oasis:names:tc:xacml:1.0:function:all-of-all
urn:oasis:names:tc:xacml:1.0:function:map
urn:oasis:names:tc:xacml:1.0:function:x500Name-match
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match
urn:oasis:names:tc:xacml:1.0:function:regexp-string-match
urn:oasis:names:tc:xacml:1.0:function:xpath-node-count
urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal
urn:oasis:names:tc:xacml:1.0:function:string-intersection
urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of
urn:oasis:names:tc:xacml:1.0:function:string-union
urn:oasis:names:tc:xacml:1.0:function:string-subset
urn:oasis:names:tc:xacml:1.0:function:string-set-equals
urn:oasis:names:tc:xacml:1.0:function:boolean-intersection
urn:oasis:names:tc:xacml:1.0:function:boolean-at-least-one-member-of
urn:oasis:names:tc:xacml:1.0:function:boolean-union
urn:oasis:names:tc:xacml:1.0:function:boolean-subset
urn:oasis:names:tc:xacml:1.0:function:boolean-set-equals
urn:oasis:names:tc:xacml:1.0:function:integer-intersection
urn:oasis:names:tc:xacml:1.0:function:integer-at-least-one-member-of
urn:oasis:names:tc:xacml:1.0:function:integer-union
urn:oasis:names:tc:xacml:1.0:function:integer-subset
urn:oasis:names:tc:xacml:1.0:function:integer-set-equals
urn:oasis:names:tc:xacml:1.0:function:double-intersection
urn:oasis:names:tc:xacml:1.0:function:double-at-least-one-member-of
urn:oasis:names:tc:xacml:1.0:function:double-union
urn:oasis:names:tc:xacml:1.0:function:double-subset
urn:oasis:names:tc:xacml:1.0:function:double-set-equals
<table>
<thead>
<tr>
<th>xacml 1.0 function: time-intersection</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>xacml 1.0 function: time-at-least-one-member-of</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: time-union</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: time-subset</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: time-set-equals</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: date-intersection</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: date-at-least-one-member-of</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: date-union</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: date-subset</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: date-set-equals</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: dateTime-intersection</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: dateTime-at-least-one-member-of</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: dateTime-union</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: dateTime-subset</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: dateTime-set-equals</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: anyURI-intersection</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: anyURI-at-least-one-member-of</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: anyURI-union</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: anyURI-subset</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: anyURI-set-equals</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: hexBinary-intersection</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: hexBinary-at-least-one-member-of</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: hexBinary-union</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: hexBinary-subset</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: hexBinary-set-equals</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: base64Binary-intersection</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: base64Binary-at-least-one-member-of</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: base64Binary-union</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: base64Binary-subset</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: base64Binary-set-equals</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: dayTimeDuration-intersection</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: dayTimeDuration-at-least-one-member-of</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: dayTimeDuration-union</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: dayTimeDuration-subset</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: dayTimeDuration-set-equals</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: yearMonthDuration-intersection</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: yearMonthDuration-at-least-one-member-of</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: yearMonthDuration-union</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: yearMonthDuration-subset</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: yearMonthDuration-set-equals</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: x500Name-intersection</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: x500Name-at-least-one-member-of</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: x500Name-union</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: x500Name-subset</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: x500Name-set-equals</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: rfc822Name-intersection</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: rfc822Name-at-least-one-member-of</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: rfc822Name-union</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: rfc822Name-subset</td>
<td>M</td>
</tr>
<tr>
<td>xacml 1.0 function: rfc822Name-set-equals</td>
<td>M</td>
</tr>
</tbody>
</table>
11. References

[DS] D. Eastlake et al., *XML-Signature Syntax and Processing*,

[Hancock] Hancock, "Polymorphic Type Checking", in Simon L. Peyton Jones,
"Implementation of Functional Programming Languages", Section 8,
Prentice-Hall International, 1987

[Haskell] Haskell, a purely functional language. Available at
http://www.haskell.org/

ACM Conference on Computer and Communications Security, Nov 1994,
Fairfax, Virginia, USA.

7653-8, IEEE Product No. SH10116-TBR

[Kudo00] Kudo M and Hada S, XML document security based on provisional
authorization, Proceedings of the Seventh ACM Conference on Computer

[LDAP-1] RFC2256, A summary of the X500(96) User Schema for use with LDAPv3,
Section 5, M Wahl, December 1997 http://www.ietf.org/rfc/rfc2256.txt

[LDAP-2] RFC2798, Definition of the inetOrgPerson, M. Smith, April 2000
http://www.ietf.org/rfc/rfc2798.txt

Recommendation, 21 February 2001. Available at:
http://www.w3.org/TR/MathML2/

[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.ifia.org/documents/infopol/copyright/perh2.txt

[RBAC] Role-Based Access Controls, David Ferraiolo and Richard Kuhn, 15th
National Computer Security Conference, 1992. Available at:
http://csrc.nist.gov/rbac

[RegEx] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001,
Appendix D. Available at: http://www.w3.org/TR/xmlschema-0/

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,

[SAML] Security Assertion Markup Language available from
http://www.oasis-open.org/committees/security/#documents

1994.

[XF] XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft
16 August 2002. Available at: http://www.w3.org/TR/2002/WD-xquery-
operators-20020816

[XS] XML Schema, parts 1 and 2. Available at:
http://www.w3.org/TR/xmlschema-1/ and
http://www.w3.org/TR/xmlschema-2/

[XPath] XML Path Language (XPath), Version 1.0, W3C Recommendation 16
November 1999. Available at: http://www.w3.org/TR/xpath
XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16
November 1999. Available at: http://www.w3.org/TR/xslt
Appendix A. Standard data-types, functions and their semantics (normative)

A.1. Introduction

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

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

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

A.2. Primitive types

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

- http://www.w3.org/2001/XMLSchema#string
- http://www.w3.org/2001/XMLSchema#boolean
- http://www.w3.org/2001/XMLSchema#integer
- http://www.w3.org/2001/XMLSchema#double
- http://www.w3.org/2001/XMLSchema#time
- http://www.w3.org/2001/XMLSchema#date
- http://www.w3.org/2001/XMLSchema#dateTime
- http://www.w3.org/2001/XMLSchema#anyURI
- http://www.w3.org/2001/XMLSchema#hexBinary
- http://www.w3.org/2001/XMLSchema#base64Binary
- http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration
- http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration
- urn:oasis:names:tc:xacml:1.0:data-type:x500Name
- urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name
A.3. Structured types

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

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

   &lt;AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"&gt;
   &lt;ds:KeyInfo&gt;Key&lt;/ds:KeyInfo&gt;&lt;/AttributeValue&gt;

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

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

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

A.4. Representations

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

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

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

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

The "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" primitive type represents electronic mail addresses, and its string representation is specified by RFC 822.

---

1 An earlier RFC, RFC 1779 "A String Representation of Distinguished Names", is less restrictive, so urn:oasis:names:tc:xacml:1.0:data-type:x500Name uses the syntax in RFC 2253 for better interoperability.
An RFC822 name consists of a local-part followed by "@" followed by a domain-part. The local-part is case-sensitive, while the domain-part (which is usually a DNS host name) is not case-sensitive.²

A.5. Bags

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

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

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

A.6. Expressions

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

XACML defines the following elements to be legal XACML expressions:

- <AttributeValue>
- <SubjectAttributeDesignator>
- <SubjectAttributeSelector>
- <ResourceAttributeDesignator>
- <ActionAttributeDesignator>
- <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.
• <AttributeSelector>
• <Apply>
• <Condition>
• <Function>

A.7. Element <AttributeValue>

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

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

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

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

A.9. Element <Apply>

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

A.10. Element <Condition>

The <Condition> element MAY appear in the <Rule> element as the premise for emitting the corresponding effect of the rule. The <Condition> element has the same structure as the <Apply> element, with the restriction that its result SHALL be of data-type "http://www.w3.org/2001/XMLSchema#boolean". The evaluation of the <Condition> element SHALL follow the same evaluation semantics as those of the <Apply> element.
A.11. Element <Function>

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

A.12. Matching elements

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

<SubjectMatch>
<ResourceMatch>
<ActionMatch>

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

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

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

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

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

The evaluation semantics for a matching element is as follows. If an operational error were to
occur while evaluating the <AttributeDesignator> or <AttributeSelector> element, then
the result of the entire expression SHALL be "Indeterminate". If the <AttributeDesignator> or 
AttributeSelector> element were to evaluate to an empty bag, then the result of the 
expression SHALL be "False". Otherwise, the MatchId function SHALL be applied between the 
explicit attribute value and each element of the bag returned from the <AttributeDesignator> 
or <AttributeSelector> element. If at least one of those function applications were to evaluate 
to "True", then the result of the entire expression SHALL be "True". Otherwise, if at least one of the 
function applications results in "Indeterminate", then the result SHALL be "Indeterminate". Finally, 
only if all function applications evaluate to "False", the result of the entire expression SHALL be 
"False".

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

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

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

```
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
  <Function
    FunctionId="urn:oasis:names:tc:xacml:1.0:function:regexp-string-match"/>
  <AttributeValue
    AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
    DataType="http://www.w3.org/2001/XMLSchema#string"/>
  <SubjectAttributeDesignator
    AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
    DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>
</Apply>
```

This expression of the semantics is NOT normative.

### A.13. Arithmetic evaluation

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

- flags - all set to 0
- trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the “division-by-zero” trap 
enabler, which SHALL be set to 1
- precision - is set to the designated double precision
- rounding - is set to round-half-even (IEEE 854 §4.1)
A.14. XACML standard functions

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

A14.1 Equality predicates

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

• string-equal

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

• boolean-equal

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

• integer-equal

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

• double-equal

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

• date-equal

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

• time-equal

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

• dateTime-equal

This function SHALL take two arguments of data-type
“http://www.w3.org/2001/XMLSchema#dateTime” and SHALL return an
http://www.w3.org/2001/XMLSchema#boolean. It SHALL perform its evaluation according to the "op:dateTime-equal" function [XF Section 8.3.8].

- dayTimeDuration-equal

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

- yearMonthDuration-equal

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

- anyURI-equal

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

- x500Name-equal

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

1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names".

2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute ValuePairs in that RDN in ascending order when compared as octet strings (described in ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").

3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section 4.1.2.4 "Issuer".

- rfc822Name-equal

This function SHALL take two arguments of data-type "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean".

This function SHALL determine whether two "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" arguments are equal. An RFC822 name consists of a local-part followed by "@" followed by a domain-part. The local-part is case-sensitive, while the domain-part (which is usually a DNS host name) is case-insensitive. Perform the following operations:

---

3 ITU-T Rec. X.520 contains rules for matching X500 names, but these are very complex and require knowledge of the syntax of various AttributeTypes. IETF RFC 3280 contains simplified matching rules that the XACML x500Name-equal function uses.
1. Normalize the domain-part of each argument to lower case
2. Compare the expressions by applying the function “urn:oasis:names:tc:xacml:1.0:function:string-equal” to the normalized arguments.

- hexBinary-equal

This function SHALL take two arguments of data-type “http://www.w3.org/2001/XMLSchema#hexBinary” and SHALL return an "True" if the octet sequences represented by the value of both arguments have equal length and are equal in a conjunctive, point-wise, comparison using the "urn:oasis:names:tc:xacml:1.0:function:integer-equal". The conversion from the string representation to an octet sequence SHALL be as specified in [XS Section 8.2.15]

- base64Binary-equal

This function SHALL take two arguments of data-type “http://www.w3.org/2001/XMLSchema#base64Binary” and SHALL return an "True" if the octet sequences represented by the value of both arguments have equal length and are equal in a conjunctive, point-wise, comparison using the "urn:oasis:names:tc:xacml:1.0:function:integer-equal". The conversion from the string representation to an octet sequence SHALL be as specified in [XS Section 8.2.16]

### A14.2 Arithmetic functions

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

- integer-add

This function MAY have two or more arguments.

- double-add

This function MAY have two or more arguments.

- integer-subtract

- double-subtract

- integer-multiply

- double-multiply

- integer-divide

- double-divide

- integer-mod

The following functions SHALL take a single argument of the specified data-type. The round and floor functions SHALL take a single argument of data-type “http://www.w3.org/2001/XMLSchema#double” and return data-type
“http://www.w3.org/2001/XMLSchema#double”. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

- integer-abs
- double-abs
- round
- floor

### A14.3 String conversion functions

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

- string-normalize-space

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

- string-normalize-to-lower-case

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

### A14.4 Numeric data-type conversion functions

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

- double-to-integer

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

- integer-to-double

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

### A14.5 Logical functions

This section contains the specification for logical functions that operate on arguments of the "http://www.w3.org/2001/XMLSchema#boolean" data-type.
• or

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

• and

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

• n-of

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

• not

This function SHALL take one logical argument. If the argument evaluates to "True", then
the result of the expression SHALL be "False". If the argument evaluates to "False", then
the result of the expression SHALL be "True". In an expression that contains any of these
functions, if ANY argument to this function evaluates to "Indeterminate", then the
expression SHALL evaluate to "Indeterminate".

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

A14.6 Arithmetic comparison functions

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

• integer-greater-than

• integer-greater-than-or-equal
• integer-less-than
• integer-less-than-or-equal
• double-greater-than
• double-greater-than-or-equal
• double-less-than
• double-less-than-or-equal

A14.7 Date and time arithmetic functions

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

• date-time-add-dayTimeDuration

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

• date-time-add-yearMonthDuration

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

• date-time-subtract-dayTimeDuration

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

• date-time-subtract-yearMonthDuration

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

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

• date-subtract-yearMonthDuration

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

A14.8 Non-numeric comparison functions

These functions perform comparison operations on two arguments of non-numerical types. In an
expression that contains any of these functions, if any argument is "Indeterminate", then the
expression SHALL evaluate to "Indeterminate".

• string-greater-than

This function SHALL take two arguments of data-type
“http://www.w3.org/2001/XMLSchema#string” and SHALL return an
“http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if and only if the
arguments are compared byte by byte and, after an initial prefix of corresponding bytes
from both arguments that are considered equal by
“urn:oasis:names:tc:xacml:1.0:function:integer-equal", the next byte by byte comparison is
such that the byte from the first argument is greater than the byte from the second
argument by the use of the function "urn:oasis:names:tc:xacml:1.0:function:integer-equal".

• string-greater-than-or-equal

This function SHALL take two arguments of data-type
“http://www.w3.org/2001/XMLSchema#string” and SHALL return an
“http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated
with the logical function "urn:oasis:names:tc:xacml:1.0:function:or" with two arguments
containing the functions "urn:oasis:names:tc:xacml:1.0:function:string-greater-than" and
“urn:oasis:names:tc:xacml:1.0:function:string-equal” containing the original arguments

• string-less-than

This function SHALL take two arguments of data-type
“http://www.w3.org/2001/XMLSchema#string” and SHALL return an
“http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if and only if the
arguments are compared byte by byte and, after an initial prefix of corresponding bytes
from both arguments are considered equal by
“urn:oasis:names:tc:xacml:1.0:function:integer-equal", the next byte by byte comparison is
such that the byte from the first argument is less than the byte from the second argument
by the use of the function "urn:oasis:names:tc:xacml:1.0:function:integer-less-than".
• string-less-than-or-equal

This function SHALL take two arguments of data-type
http://www.w3.org/2001/XMLSchema#string and SHALL return an
with the function "urn:oasis:names:tc:xacml:1.0:function:or" with two arguments containing
the functions "urn:oasis:names:tc:xacml:1.0:function:string-less-than" and
"urn:oasis:names:tc:xacml:1.0:function:string-equal" containing the original arguments.

• time-greater-than

This function SHALL take two arguments of data-type
http://www.w3.org/2001/XMLSchema#time and SHALL return an
"http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
argument is greater than the second argument according to the order relation specified for
"http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

• time-greater-than-or-equal

This function SHALL take two arguments of data-type
http://www.w3.org/2001/XMLSchema#time and SHALL return an
"http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
argument is greater than or equal to the second argument according to the order relation
specified for “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].

• time-less-

This function SHALL take two arguments of data-type
http://www.w3.org/2001/XMLSchema#time and SHALL return an
"http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
argument is less than the second argument according to the order relation specified for
"http://www.w3.org/2001/XMLSchema#time" [XS Section 3.2.8].

• time-less-than-or-equal

This function SHALL take two arguments of data-type
http://www.w3.org/2001/XMLSchema#time and SHALL return an
"http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
argument is less than or equal to the second argument according to the order relation
specified for “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].

• date-Time-greater-than

This function SHALL take two arguments of data-type
http://www.w3.org/2001/XMLSchema#dateTime and SHALL return an
"http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
argument is greater than the second argument according to the order relation specified for
"http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

• date-Time-greater-than-or-equal

This function SHALL take two arguments of data-type
http://www.w3.org/2001/XMLSchema#dateTime and SHALL return an
"http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first
argument is greater than or equal to the second argument according to the order relation
specified for “http://www.w3.org/2001/XMLSchema#dateTime” [XS Section 3.2.7].

• date-Time-less-than

doxygen:datetime:greater-than
This function SHALL take two arguments of data-type "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is less than the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

- date-time-less-than-or-equal

This function SHALL take two arguments of data-type "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is less than or equal to the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

- date-greater-than

This function SHALL take two arguments of data-type "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is greater than the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.9].

- date-greater-than-or-equal

This function SHALL take two arguments of data-type "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is greater than or equal to the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.9].

- date-less-than

This function SHALL take two arguments of data-type "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is less than the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.9].

- date-less-than-or-equal

This function SHALL take two arguments of data-type "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument is less than or equal to the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.9].

**A14.9 Bag functions**

These functions operate on a *bag* of *type* values, where *data-type* is one of the primitive types. In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate". Some additional conditions defined for each function below SHALL cause the expression to evaluate to "Indeterminate".

- type-one-and-only
This function SHALL take an argument of a bag of type values and SHALL return a value of data-type. It SHALL return the only value in the bag. If the bag does not have one and only one value, then the expression SHALL evaluate to “Indeterminate”.

- type-bag-size

This function SHALL take a bag of type values as an argument and SHALL return an “http://www.w3.org/2001/XMLSchema#integer” indicating the number of values in the bag.

- type-is-in

This function SHALL take an argument of data-type type as the first argument and a bag of type values as the second argument. The expression SHALL evaluate to “True” if the first argument matches by the "urn:oasis:names:tc:xacml:1.0:function:type-equal" to any value in the bag.

- type-bag

This function SHALL take any number of arguments of a single data-type and return a bag of type values containing the values of the arguments. An application of this function to zero arguments SHALL produce an empty bag of the specified data-type.

**A14.10 Set functions**

These functions operate on bags mimicking sets by eliminating duplicate elements from a bag. In an expression that contains any of these functions, if any argument is “Indeterminate”, then the expression SHALL evaluate to “Indeterminate”.

- type-intersection

This function SHALL take two arguments that are both a bag of type values. The expression SHALL return a bag of type values such that it contains only elements that are common between the two bags, which is determined by "urn:oasis:names:tc:xacml:1.0:function:type-equal". No duplicates as determined by "urn:oasis:names:tc:xacml:1.0:function:type-equal" SHALL exist in the result.

- type-at-least-one-member-of

This function SHALL take two arguments that are both a bag of type values. The expression SHALL evaluate to “True” if at least one element of the first argument is contained in the second argument as determined by "urn:oasis:names:tc:xacml:1.0:function:type-is-in".

- type-union

This function SHALL take two arguments that are both a bag of type values. The expression SHALL return a bag of type such that it contains all elements of both bags. No duplicates as determined by "urn:oasis:names:tc:xacml:1.0:function:type-equal" SHALL exist in the result.

- type-subset

This function SHALL take two arguments that are both a bag of type values. It SHALL return “True” if the first argument is a subset of the second argument. Each argument is considered to have its duplicates removed as determined by "urn:oasis:names:tc:xacml:1.0:function:type-equal" before subset calculation.
• type-set-equals

This function SHALL take two arguments that are both a bag of type values and SHALL return the result of applying "urn:oasis:names:tc:xacml:1.0:function:and" to the application of "urn:oasis:names:tc:xacml:1.0:function:type-subset" to the first and second arguments and the application of "urn:oasis:names:tc:xacml:1.0:function:type-subset" to the second and first arguments.

A14.11 Higher-order bag functions

This section describes functions in XACML that perform operations on bags such that functions may be applied to the bags in general.

In this section, a general-purpose functional language called Haskell [Haskell] is used to formally specify the semantics of these functions. Although the English description is adequate, a formal specification of the semantics is helpful.

For a quick summary, in the following Haskell notation, a function definition takes the form of clauses that are applied to patterns of structures, namely lists. The symbol "[]" denotes the empty list, whereas the expression "(x:xs)" matches against an argument of a non-empty list of which "x" represents the first element of the list, and "xs" is the rest of the list, which may be an empty list. We use the Haskell notion of a list, which is an ordered collection of elements, to model the XACML bags of values.

A simple Haskell definition of a familiar function "urn:oasis:names:tc:xacml:1.0:function:and" that takes a list of booleans is defined as follows:

```haskell
and:: [Bool] -> Bool
and [] = "True"
and (x:xs) = x && (and xs)
```

The first definition line denoted by a "::" formally describes the data-type of the function, which takes a list of booleans, denoted by "[Bool]", and returns a boolean, denoted by "Bool". The second definition line is a clause that states that the function "and" applied to the empty list is "True". The second definition line is a clause that states that for a non-empty list, such that the first element is "x", which is a value of data-type Bool, the function "and" applied to x SHALL be combined with, using the logical conjunction function, which is denoted by the infix symbol "&&", the result of recursively applying the function "and" to the rest of the list. Of course, an application of the "and" function is "True" if and only if the list to which it is applied is empty or every element of the list is "True". For example, the evaluation of the following Haskell expressions,

```haskell
(and []), (and ["True"]), (and ["True","True"]), (and ["True","True","False"])
```

evaluate to "True", "True", "True", and "False", respectively.

In an expression that contains any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to "Indeterminate".

• any-of

This function applies a boolean function between a specific primitive value and a bag of values, and SHALL return "True" if and only if the predicate is "True" for at least one element of the bag.

This function SHALL take three arguments. The first argument SHALL be a <Function> element that names a boolean function that takes two arguments of primitive types. The second argument SHALL be a value of a primitive data-type. The third argument SHALL
be a bag of a primitive data-type. The expression SHALL be evaluated as if the function
named in the <Function> element is applied to the second argument and each element
of the third argument (the bag) and the results are combined with
"urn:oasis:names:tc:xacml:1.0:function:or".

In Haskell, the semantics of this operation are as follows:

\[
\text{any}\_\text{of} :: (a \to b \to \text{Bool}) \to a \to [b] \to \text{Bool} \\
\text{any}\_\text{of} f a [] = \text{"False"} \\
\text{any}\_\text{of} f a (x:xs) = (f a x) || (\text{any}\_\text{of} f a xs)
\]

In the above notation, "f" is the function name to be applied, "a" is the primitive value, and
"(x:xs)" represents the first element of the list as "x" and the rest of the list as "xs".

For example, the following expression SHALL return "True":

\[
<\text{Apply} \text{ FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of"}> \\
<\text{Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"}/> \\
<\text{AttributeValue} \\
<\text{DataType="http://www.w3.org/2001/XMLSchema#string"} Paul</\text{AttributeValue}> \\
<\text{Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag"}> \\
<\text{AttributeValue} \\
<\text{DataType="http://www.w3.org/2001/XMLSchema#string"} John</\text{AttributeValue}> \\
<\text{Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag"}> \\
<\text{AttributeValue} \\
<\text{DataType="http://www.w3.org/2001/XMLSchema#string"} Paul</\text{AttributeValue}> \\
<\text{Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag"}> \\
<\text{AttributeValue} \\
<\text{DataType="http://www.w3.org/2001/XMLSchema#string"} George</\text{AttributeValue}> \\
<\text{Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag"}> \\
<\text{AttributeValue} \\
<\text{DataType="http://www.w3.org/2001/XMLSchema#string"} Ringo</\text{AttributeValue}> \\
</\text{Apply}> \\
</\text{Apply}>
\]

This expression is "True" because the first argument is equal to at least one of the
elements of the bag.

all_of

This function applies a boolean function between a specific primitive value and a bag of
values, and returns "True" if and only if the predicate is "True" for every element of the bag.

This function SHALL take three arguments. The first argument SHALL be a <Function>
element that names a boolean function that takes two arguments of primitive types. The
second argument SHALL be a value of a primitive data-type. The third argument SHALL
be a bag of a primitive data-type. The expression SHALL be evaluated as if the function
named in the <Function> element were applied to the second argument and each
element of the third argument (the bag) and the results were combined using
"urn:oasis:names:tc:xacml:1.0:function:and".

In Haskell, the semantics of this operation are as follows:

\[
\text{all}\_\text{of} :: (a \to b \to \text{Bool}) \to a \to [b] \to \text{Bool} \\
\text{all}\_\text{of} f a [] = \text{"False"} \\
\text{all}\_\text{of} f a (x:xs) = (f a x) && (\text{all}\_\text{of} f a xs)
\]

In the above notation, "f" is the function name to be applied, "a" is the primitive value, and
"(x:xs)" represents the first element of the list as "x" and the rest of the list as "xs".

For example, the following expression SHALL evaluate to "True":

\[
"urn:oasis:names:tc:xacml:1.0:function:and"
\]

\[
<\text{Apply} \text{ FunctionId="urn:oasis:names:tc:xacml:1.0:function:and"}> \\
<\text{Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"}/> \\
<\text{AttributeValue} \\
<\text{DataType="http://www.w3.org/2001/XMLSchema#string"} Paul</\text{AttributeValue}> \\
<\text{Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag"}> \\
<\text{AttributeValue} \\
<\text{DataType="http://www.w3.org/2001/XMLSchema#string"} John</\text{AttributeValue}> \\
<\text{Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag"}> \\
<\text{AttributeValue} \\
<\text{DataType="http://www.w3.org/2001/XMLSchema#string"} Paul</\text{AttributeValue}> \\
<\text{Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag"}> \\
<\text{AttributeValue} \\
<\text{DataType="http://www.w3.org/2001/XMLSchema#string"} George</\text{AttributeValue}> \\
<\text{Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag"}> \\
<\text{AttributeValue} \\
<\text{DataType="http://www.w3.org/2001/XMLSchema#string"} Ringo</\text{AttributeValue}> \\
</\text{Apply}> \\
</\text{Apply}>
\]
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">
  <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater"/>
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#integer">9</AttributeValue>
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
</Apply>

This expression is "True" because the first argument is greater than all of the elements of the bag.

- any-of-any

This function applies a boolean function between each element of a bag of values and each element of another bag of values, and returns "True" if and only if the predicate is "True" for at least one comparison.

This function SHALL take three arguments. The first argument SHALL be a <Function> element that names a boolean function that takes two arguments of primitive types. The second argument SHALL be a bag of a primitive data-type. The third argument SHALL be a bag of a primitive data-type. The expression SHALL be evaluated as if the function named in the <Function> element were applied between every element in the second argument and every element of the third argument (the bag) and the results were combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics are that the result of the expression SHALL be "True" if and only if the applied predicate is "True" for any comparison of elements from the two bags.

In Haskell, taking advantage of the “any_of” function defined above, the semantics of the “any_of_any” function are as follows:

```haskell
any_of_any :: ( a -> b -> Bool ) -> [a] -> [b] -> Bool
any_of_any f []     ys = "False"
any_of_any f (x:xs) ys = (any_of f x ys) || (any_of_any f xs ys)
```

In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first element of the list as ‘x’ and the rest of the list as “xs”.

For example, the following expression SHALL evaluate to "True":

```haskell
any_of any :: ( a -> b -> Bool ) -> [a] -> [b] -> Bool
any_of any f []     ys = "False"
any_of any f (x:xs) ys = (any_of f x ys) || (any_of_any f xs ys)
```
This expression is "True" because at least one of the elements of the first \textbf{bag}, namely "Ringo", is equal to at least one of the string values of the second \textbf{bag}.

- all-of-any

This function applies a boolean function between the elements of two \textbf{bags}. The expression is "True" if and only if the predicate is "True" between each and all of the elements of the first \textbf{bag} collectively against at least one element of the second \textbf{bag}.

This function SHALL take three arguments. The first argument SHALL be a <Function> element that names a boolean function that takes two arguments of primitive types. The second argument SHALL be a bag of a primitive data-type. The third argument SHALL be a bag of a primitive data-type. The expression SHALL be evaluated as if function named in the <Function> element were applied between every element in the second argument and every element of the third argument (the bag) using "urn:oasis:names:tc:xacml:1.0:function:and". The semantics are that the result of the expression SHALL be "True" if and only if the applied predicate is "True" for each element of the first \textbf{bag} and any element of the second \textbf{bag}.

In Haskell, taking advantage of the "any_of" function defined in Haskell above, the semantics of the "all_of_any" function are as follows:

\[
\text{all_of_any} :: ( a -> b -> \text{Bool} ) -> [a] -> [b] -> \text{Bool} \\
\text{all_of_any} f [] ys = \text{"False"} \\
\text{all_of_any} f (x:xs) ys = (\text{any_of} f x ys) \&\&(\text{all_of_any} f \text{ xs} \text{ ys})
\]

In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first element of the list as "x" and the rest of the list as "xs".

For example, the following expression SHALL evaluate to "True":

```haskell
all_of_any (==) [0] [0,1,2,3,4]`
This expression is "True" because all of the elements of the first bag, each “10” and “20”, are greater than at least one of the integer values “1”, “3”, “5”, “21” of the second bag.

- any-of-all

This function applies a boolean function between the elements of two bags. The expression SHALL be "True" if and only if the predicate is "True" between at least one of the elements of the first bag collectively against all the elements of the second bag.

This function SHALL take three arguments. The first argument SHALL be a <Function> element that names a boolean function that takes two arguments of primitive types. The second argument SHALL be a bag of a primitive data-type. The third argument SHALL be a bag of a primitive data-type. The expression SHALL be evaluated as if the function named in the <Function> element were applied between every element in the second argument and every element of the third argument (the bag) and the results were combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics are that the result of the expression SHALL be "True" if and only if the applied predicate is "True" for any element of the first bag compared to all the elements of the second bag.

In Haskell, taking advantage of the "all_of" function defined in Haskell above, the semantics of the "any_of_all" function are as follows:

```haskell
any_of_all :: (a -> b -> Bool) -> [a] -> [b] -> Bool
any_of_all f []     ys = "False"
any_of_all f (x:xs) ys = (all_of f x ys) || (any_of_all f xs ys)
```

In the above notation, “f” is the function name to be applied and “(x:xs)” represents the first element of the list as “x” and the rest of the list as “xs”.

For example, the following expression SHALL evaluate to "True":
This expression is "True" because at least one element of the first bag, namely "5", is greater than all of the integer values "1", "2", "3", "4" of the second bag.

- all-of-all

This function applies a boolean function between the elements of two bags. The expression SHALL be "True" if and only if the predicate is "True" between each and all of the elements of the first bag collectively against all the elements of the second bag.

This function SHALL take three arguments. The first argument SHALL be a \(<Function>\) element that names a boolean function that takes two arguments of primitive types. The second argument SHALL be a bag of a primitive data-type. The third argument SHALL be a bag of a primitive data-type. The expression is evaluated as if the function named in the \(<Function>\) element were applied between every element in the second argument and every element of the third argument (the bag) and the results were combined using "urn:oasis:names:tc:xacml:1.0:function:and". The semantics are that the result of the expression is "True" if and only if the applied predicate is "True" for all elements of the first bag compared to all the elements of the second bag.

In Haskell, taking advantage of the "all_of" function defined in Haskell above, the semantics of the "all_of_all" function is as follows:

\[
\text{all_of_all} :: (a -> b -> \text{Bool}) -> [a] -> [b] -> \text{Bool}\\
\text{all_of_all} \ f \ [] \ y = \text{"False"} \\
\text{all_of_all} \ f \ (x:xs) \ ys = (\text{all_of_all} \ f \ x \ ys) && (\text{all_of_all} \ f \ xs \ ys)
\]

In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first element of the list as "x" and the rest of the list as "xs".

For example, the following expression SHALL evaluate to "True":

\[
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-all"\\
<Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater"/\\
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
<AttributeValue DataValue="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
<AttributeValue DataValue="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
</Apply>
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
<AttributeValue DataValue="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
<AttributeValue DataValue="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
<AttributeValue DataValue="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
<AttributeValue DataValue="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
</Apply>
</Apply>
\]
This expression is "True" because all elements of the first bag, "5" and "6", are each greater than all of the integer values "1", "2", "3", "4" of the second bag.

- map

This function converts a bag of values to another bag of values.

This function SHALL take two arguments. The first function SHALL be a <Function> element naming a function that takes a single argument of a primitive data-type and returns a value of a primitive data-type. The second argument SHALL be a bag of a primitive data-type. The expression SHALL be evaluated as if the function named in the <Function> element were applied to each element in the bag resulting in a bag of the converted value. The result SHALL be a bag of the primitive data-type that is the same data-type that is returned by the function named in the <Function> element.

In Haskell, this function is defined as follows:

```
map :: (a -> b) -> [a] -> [b]
```

```
map f [] = []
map f (x:xs) = (fx):(map f xs)
```

In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first element of the list as "x" and the rest of the list as "xs".

For example, the following expression,

```
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:map">
  <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
      <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>
      <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>
    </Apply>
  </Function>
</Apply>
```

evaluates to a bag containing "hello" and "world!".
A14.12 Special match functions

These functions operate on various types and evaluate to
"http://www.w3.org/2001/XMLSchema#boolean" based on the specified standard matching
algorithm. In an expression that contains any of these functions, if any argument is "Indeterminate",
then the expression SHALL evaluate to "Indeterminate".

- regexp-string-match

This function decides a regular expression match. It SHALL take two arguments of
"http://www.w3.org/2001/XMLSchema#string" and SHALL return an
"http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
expression and the second argument SHALL be a general string. The function
specification SHALL be that of the "xf:matches" function with the arguments reversed [XF
Section 6.3.15].

- x500Name-match

This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-
type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It
shall return "True" if and only if the first argument matches some terminal sequence of
RDNs from the second argument when compared using x500Name-equal.

- rfc822Name-match

This function SHALL take two arguments, the first is of data-type
"http://www.w3.org/2001/XMLSchema#string" and the second is of data-type
"urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an
"http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if
the first argument matches the second argument according to the following specification.

An RFC822 name consists of a local-part followed by "@" followed by domain-part. The
local-part is case-sensitive, while the domain-part (which is usually a DNS name) is not
case-sensitive. 4

The second argument contains a complete rfc822Name. The first argument is a complete
or partial rfc822Name used to select appropriate values in the second argument as follows.

In order to match a particular mailbox in the second argument, the first argument must
specify the complete mail address to be matched. For example, if the first argument is
"Anderson@sun.com", this matches a value in the second argument of
"Anderson@sun.com" and "Anderson@SUN.COM", but not "Anne.Anderson@sun.com",
"anderson@sun.com" or "Anderson@east.sun.com".

In order to match any mail address at a particular domain in the second argument, the first
argument must specify only a domain name (usually a DNS name). For example, if the first
argument is "sun.com", this matches a value in the first argument of "Anderson@sun.com"
or "Baxter@SUN.COM", but not "Anderson@east.sun.com".

In order to match any mail address in a particular domain in the second argument, the first
argument must specify the desired domain-part with a leading ".". For example, if the first
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.
A14.13 XPath-based functions

This section specifies functions that take XPath expressions for arguments. An XPath expression evaluates to a node-set, which is a set of XML nodes that match the expression. A node or node-set is not in the formal data-type system of XACML. All comparison or other operations on node-sets are performed in the isolation of the particular function specified. The XPath expressions in these functions are restricted to the XACML request context. The <xacml-context:Request> element is a context node for every XPath expression. The following functions are defined:

- xpath-node-count

  This function SHALL take an "http://www.w3.org/2001/XMLSchema#string" as an argument, which SHALL be interpreted as an XPath expression, and evaluates to an "http://www.w3.org/2001/XMLSchema#integer". The value returned from the function SHALL be the count of the nodes within the node-set that matches the given XPath expression.

- xpath-node-equal

  This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments, which SHALL be interpreted as XPath expressions, and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if any XML node from the node-set matched by the first argument equals according to the "op:node-equal" function [XF Section 13.1.6] any XML node from the node-set matched by the second argument.

- xpath-node-match

  This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments, which SHALL be interpreted as XPath expressions and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if either of the following two conditions is satisfied: (1) Any XML node from the node-set matched by the first argument is equal according to "op:node-equal" [XF Section 13.1.6] to any XML node from the node-set matched by the second argument. (2) Any attribute and element node below any XML node from the node-set matched by the first argument is equal according to "op:node-equal" [XF Section 13.1.6] to any XML node from the node-set matched by the second argument.

  NOTE: The first condition is equivalent to "xpath-node-equal", and guarantees that "xpath-node-equal" is a special case of "xpath-node-match".

A14.14 Extension functions and primitive types

Functions and primitive types are specified by string identifiers allowing for the introduction of functions in addition to those specified by XACML. This approach allows one to extend the XACML module with special functions and special primitive data-types.

In order to preserve some integrity to the XACML evaluation strategy, the result of all function applications SHALL depend only on the values of its arguments. Global and hidden parameters SHALL NOT affect the evaluation of an expression. Functions SHALL NOT have side effects, as evaluation order cannot be guaranteed in a standard way.
Appendix B. XACML identifiers (normative)

This section defines standard identifiers for commonly used entities. All XACML-defined identifiers have the common base:

```xml
urn:oasis:names:tc:xacml:1.0
```

B.1. XACML namespaces

There are currently two defined XACML namespaces.

Policies are defined using this identifier.

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

Request and response contexts are defined using this identifier.

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

B.2. Access subject categories

This identifier indicates the system entity that initiated the access request. That is, the initial entity in a request chain. If subject category is not specified, this is the default value.

```xml
urn:oasis:names:tc:xacml:1.0:subject-category:access-subject
```

This identifier indicates the system entity that will receive the results of the request. Used when it is distinct from the access-subject.

```xml
urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject
```

This identifier indicates a system entity through which the access request was passed. There may be more than one. No means is provided to specify the order in which they passed the message.

```xml
urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject
```

This identifier indicates a system entity associated with a local or remote codebase that generated the request. Corresponding subject attributes might include the URL from which it was loaded and/or the identity of the code-signer. There may be more than one. No means is provided to specify the order they processed the request.

```xml
urn:oasis:names:tc:xacml:1.0:subject-category:codebase
```

This identifier indicates a system entity associated with the computer that initiated the access request. An example would be an IPsec identity.

```xml
urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine
```

B.3. XACML functions

This identifier is the base for all the identifiers in the table of functions. See Section A.1.

```xml
urn:oasis:names:tc:xacml:1.0:function
```

B.4. Data-types

The following identifiers indicate useful data-types.

X.500 distinguished name

```xml
urn:oasis:names:tc:xacml:1.0:attribute-value
```
An x500Name contains an ITU-T Rec. X.520 Distinguished Name. The valid syntax for such a name is described in IETF RFC 2253 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names".

RFC822 Name

urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name

An rfc822Name contains an "e-mail name". The valid syntax for such a name is described in IETF RFC 2821, Section 4.1.2, Command Argument Syntax, under the term "Mailbox".

The following data-type identifiers are defined by XML Schema.

http://www.w3.org/2001/XMLSchema#string
http://www.w3.org/2001/XMLSchema#boolean
http://www.w3.org/2001/XMLSchema#integer
http://www.w3.org/2001/XMLSchema#double
http://www.w3.org/2001/XMLSchema#time
http://www.w3.org/2001/XMLSchema#date
http://www.w3.org/2001/XMLSchema#dateTime
http://www.w3.org/2001/XMLSchema#anyURI
http://www.w3.org/2001/XMLSchema#hexBinary
http://www.w3.org/2001/XMLSchema#base64Binary

The following data-type identifiers correspond to the dayTimeDuration and yearMonthDuration data-types defined in [XF Sections 8.2.2 and 8.2.1, respectively].

http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration
http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration

### B.5. Subject attributes

These identifiers indicate **attributes** of a **subject**. When used, they SHALL appear within a `<Subject>` element of the request **context**. They SHALL be accessed via a `<SubjectAttributeDesignator>` or an `<AttributeSelector>` element pointing into a `<Subject>` element of the request **context**.

At most one of each of these attributes is associated with each subject. Each attribute associated with authentication included within a single `<Subject>` element relates to the same authentication event.

This identifier indicates the name of the **subject**. The default format is http://www.w3.org/2001/XMLSchema#string. To indicate other formats, use **Data**Type attributes listed in B.4

urn:oasis:names:tc:xacml:1.0:subject:subject-id

This identifier indicates the **subject** category. "access-subject" is the default.

urn:oasis:names:tc:xacml:1.0:subject:category

This identifier indicates the security domain of the **subject**. It identifies the administrator and policy that manages the name-space in which the **subject** id is administered.

urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier

This identifier indicates a public key used to confirm the **subject's** identity.

urn:oasis:names:tc:xacml:1.0:subject:key-info

This identifier indicates the time at which the **subject** was authenticated.

urn:oasis:names:tc:xacml:1.0:subject:authentication-time

This identifier indicates the method used to authenticate the **subject**.

urn:oasis:names:tc:xacml:1.0:subject:authentication-method
This identifier indicates the time at which the subject initiated the access request, according to the PEP.
urn:oasis:names:tc:xacml:1.0:subject:request-time

This identifier indicates the time at which the subject’s current session began, according to the PEP.
urn:oasis:names:tc:xacml:1.0:subject:session-start-time

The following identifiers indicate the location where authentication credentials were activated. They are intended to support the corresponding entities from the SAML authentication statement.

This identifier indicates that the location is expressed as an IP address.
urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address

This identifier indicates that the location is expressed as a DNS name.
urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dn-name

Where a suitable attribute is already defined in LDAP [LDAP-1, LDAP-2], the XACML identifier SHALL be formed by adding the attribute name to the URI of the LDAP specification. For example, the attribute name for the userPassword defined in the rfc2256 SHALL be:
http://www.ietf.org/rfc/rfc2256.txt#userPassword

B.6. Resource attributes

These identifiers indicate attributes of the resource. When used, they SHALL appear within the <Resource> element of the request context. They SHALL be accessed via a <ResourceAttributeDesignator> or an <AttributeSelector> element pointing into the <Resource> element of the request context.

This identifier indicates the entire URI of the resource.
urn:oasis:names:tc:xacml:1.0:resource:resource-id

A resource attribute used to indicate values extracted from the resource.
urn:oasis:names:tc:xacml:1.0:resource:resource-content

This identifier indicates the last (rightmost) component of the file name. For example, if the URI is: “file:///home/my/status#pointer”, the simple-file-name is “status”.
urn:oasis:names:tc:xacml:1.0:resource:simple-file-name

This identifier indicates that the resource is specified by an XPath expression.
urn:oasis:names:tc:xacml:1.0:resource:xpath

This identifier indicates a UNIX file-system path.
urn:oasis:names:tc:xacml:1.0:resource:ufs-path

This identifier indicates the scope of the resource, as described in Section 7.8.
urn:oasis:names:tc:xacml:1.0:resource:scope

The allowed value for this attribute is of data-type http://www.w3.org/2001/XMLSchema#string, and is either “Immediate”, “Children” or “Descendants”.

B.7. Action attributes

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 <ActionAttributeDesignator> or an <AttributeSelector> element pointing into the <Action> element of the request context.
B.8. Environment attributes

These identifiers indicate attributes of the environment within which the decision request is to be evaluated. When used in the decision request, they SHALL appear in the <Environment> element of the request context. They SHALL be accessed via an <EnvironmentAttributeDesignator> or an <AttributeSelector> element pointing into the <Environment> element of the request context.

This identifier indicates the current time at the PDP. In practice it is the time at which the request context was created.

The following status code identifiers are defined.

This identifier indicates success.

This identifier indicates that attributes necessary to make a policy decision were not available.

This identifier indicates that some attribute value contained a syntax error, such as a letter in a numeric field.

This identifier indicates that an error occurred during policy evaluation. An example would be division by zero.

B.10. Combining algorithms

The deny-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

The deny-overrides policy-combining algorithm has the following value for policyCombiningAlgId:

The permit-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

The permit-overrides policy-combining algorithm has the following value for policyCombiningAlgId:
The first-applicable rule-combining algorithm has the following value for ruleCombiningAlgId:

```
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable
```

The first-applicable policy-combining algorithm has the following value for policyCombiningAlgId:

```
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable
```

The only-one-applicable-policy policy-combining algorithm has the following value for policyCombiningAlgId:

```
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable
```

The ordered-deny-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

```
```

The ordered-deny-overrides policy-combining algorithm has the following value for policyCombiningAlgId:

```
```

The ordered-permit-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

```
```

The ordered-permit-overrides policy-combining algorithm has the following value for policyCombiningAlgId:

```
```
Appendix C. Combining algorithms (normative)

This section contains a description of the rule-combining and policy-combining algorithms specified by XACML.

C.1. Deny-overrides.

The following specification defines the “Deny-overrides” rule-combining algorithm of a policy.

In the entire set of rules in the policy, if any rule evaluates to "Deny", then the result of the rule combination SHALL be "Deny". If any rule evaluates to "Permit" and all other rules evaluate to "NotApplicable", then the result of the rule combination SHALL be "Permit". In other words, "Deny" takes precedence, regardless of the result of evaluating any of the other rules in the combination. If all rules are found to be "NotApplicable" to the decision request, then the rule combination SHALL evaluate to "NotApplicable".

If an error occurs while evaluating the target or condition of a rule that contains an effect value of "Deny" then the evaluation SHALL continue to evaluate subsequent rules, looking for a result of "Deny". If no other rule evaluates to "Deny", then the combination SHALL evaluate to "Indeterminate", with the appropriate error status.

If at least one rule evaluates to "Permit", all other rules that do not have evaluation errors evaluate to "Permit" or "NotApplicable" and all rules that do have evaluation errors contain effects of "Permit", then the result of the combination SHALL be "Permit".

The following pseudo-code represents the evaluation strategy of this rule-combining algorithm.

```java
Decision denyOverridesRuleCombiningAlgorithm(Rule rule[])
{
    Boolean atLeastOneError = false;
    Boolean potentialDeny = false;
    Boolean atLeastOnePermit = false;
    for (i=0 ; i < lengthOf(rules) ; i++)
    {
        Decision decision = evaluate(rule[i]);
        if (decision == Deny)
        {
            return Deny;
        }
        if (decision == Permit)
        {
            atLeastOnePermit = true;
            continue;
        }
        if (decision == NotApplicable)
        {
            continue;
        }
        if (decision == Indeterminate)
        {
            atLeastOneError = true;
        }
        if (effect(rule[i]) == Deny)
        {
            potentialDeny = true;
        }
    }
```

```java
continue;
```
The following specification defines the “Deny-overrides” **policy-combining algorithm** of a **policy set**.

In the entire set of **policies** in the **policy set**, if any **policy** evaluates to "Deny", then the result of the **policy** combination SHALL be "Deny". In other words, "Deny" takes precedence, regardless of the result of evaluating any of the other **policies** in the **policy set**. If all **policies** are found to be "NotApplicable" to the **decision request**, then the **policy set** SHALL evaluate to "NotApplicable".

If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set** SHALL evaluate to "Deny".

The following pseudo-code represents the evaluation strategy of this **policy-combining algorithm**.

```java
Decision denyOverridesPolicyCombiningAlgorithm(Policy policy[]) {
    Boolean atleastOnePermit = false;
    for( i=0 ; i < lengthOf(policy) ; i++ )
    {
        Decision decision = evaluate(policy[i]);
        if (decision == Deny)
        {
            return Deny;
        }
        if (decision == Permit)
        {
            atleastOnePermit = true;
            continue;
        }
        if (decision == NotApplicable)
        {
            continue;
        }
        if (decision == Indeterminate)
        {
            return Deny;
        }
        if (atleastOnePermit)
        {
            return Permit;
        }
        return NotApplicable;
    }
}
```

**Obligations** of the individual **policies** shall be combined as described in Section 7.11.
C.2. Ordered-deny-overrides (non-normative)

The following specification defines the "Ordered-deny-overrides" rule-combining algorithm of a policy.

The behavior of this algorithm is identical to that of the Deny-overrides rule-combining algorithm with one exception. The order in which the collection of rules is evaluated SHALL match the order as listed in the policy.

The following specification defines the "Ordered-deny-overrides" policy-combining algorithm of a policy set.

The behavior of this algorithm is identical to that of the Deny-overrides policy-combining algorithm with one exception. The order in which the collection of policies is evaluated SHALL match the order as listed in the policy set.

C.3. Permit-overrides

The following specification defines the "Permit-overrides" rule-combining algorithm of a policy.

In the entire set of rules in the policy, if any rule evaluates to "Permit", then the result of the rule combination SHALL be "Permit". If any rule evaluates to "Deny" and all other rules evaluate to "NotApplicable", then the policy SHALL evaluate to "Deny". In other words, "Permit" takes precedence, regardless of the result of evaluating any of the other rules in the policy. If all rules are found to be "NotApplicable" to the decision request, then the policy SHALL evaluate to "NotApplicable".

If an error occurs while evaluating the target or condition of a rule that contains an effect of "Permit" then the evaluation SHALL continue looking for a result of "Permit". If no other rule evaluates to "Permit", then the policy SHALL evaluate to "Indeterminate", with the appropriate error status.

If at least one rule evaluates to "Deny", all other rules that do not have evaluation errors evaluate to "Deny" or "NotApplicable" and all rules that do have evaluation errors contain an effect value of "Deny", then the policy SHALL evaluate to "Deny".

The following pseudo-code represents the evaluation strategy of this rule-combining algorithm.

```java
Decision permitOverridesRuleCombiningAlgorithm(Rule rule[])
{
    Boolean atLeastOneError   = false;
    Boolean potentialPermit   = false;
    Boolean atLeastOneDeny    = false;
    for( i=0 ; i < lengthOf(rule) ; i++ )
    {
        Decision decision = evaluate(rule[i]);
        if (decision == Deny)
        {
            atLeastOneDeny = true;
            continue;
        }        
        if (decision == Permit)
        {
            return Permit;
        }        
        if (decision == NotApplicable)
        {
            continue;
        }
    }
}
```
The following specification defines the "Permit-overrides" policy-combining algorithm of a policy set.

In the entire set of policies in the policy set, if any policy evaluates to "Permit", then the result of the policy combination SHALL be "Permit". In other words, "Permit" takes precedence, regardless of the result of evaluating any of the other policies in the policy set. If all policies are found to be "NotApplicable" to the decision request, then the policy set SHALL evaluate to "NotApplicable".

If an error occurs while evaluating the target of a policy, a reference to a policy is considered invalid or the policy evaluation results in "Indeterminate", then the policy set SHALL evaluate to "Indeterminate", with the appropriate error status, provided no other policies evaluate to "Permit" or "Deny".

The following pseudo-code represents the evaluation strategy of this policy-combining algorithm.

```java
Decision permitOverridesPolicyCombiningAlgorithm(Policy policy[]) {
    Boolean atLeastOneError = false;
    Boolean atLeastOneDeny = false;
    for (i=0 ; i < lengthOf(policy) ; i++ )
    {
        Decision decision = evaluate(policy[i]);
        if (decision == Deny)
            atLeastOneDeny = true;
        continue;
        if (decision == Permit)
            return Permit;
    }
    if (decision == NotApplicable)
        return NotApplicable;
    return Indeterminate;
}
```
if (decision == Indeterminate)
{
    atleastOneError = true;
    continue;
}
if (atleastOneDeny)
{
    return Deny;
}
if (atleastOneError)
{
    return Indeterminate;
}
return NotApplicable;

Obligations of the individual policies shall be combined as described in Section 7.11.

C.4. Ordered-permit-overrides (non-normative)

The following specification defines the "Ordered-permit-overrides" rule-combining algorithm of a policy.

The behavior of this algorithm is identical to that of the Permit-overrides rule-combining algorithm with one exception. The order in which the collection of rules is evaluated SHALL match the order as listed in the policy.

The following specification defines the "Ordered-permit-overrides" policy-combining algorithm of a policy set.

The behavior of this algorithm is identical to that of the Permit-overrides policy-combining algorithm with one exception. The order in which the collection of policies is evaluated SHALL match the order as listed in the policy set.

C.5. First-applicable

The following specification defines the "First-Applicable" rule-combining algorithm of a policy.

Each rule SHALL be evaluated in the order in which it is listed in the policy. For a particular rule, if the target matches and the condition evaluates to "True", then the evaluation of the policy SHALL halt and the corresponding effect of the rule SHALL be the result of the evaluation of the policy (i.e. "Permit" or "Deny"). For a particular rule selected in the evaluation, if the target evaluates to "False" or the condition evaluates to "False", then the next rule in the order SHALL be evaluated. If no further rule in the order exists, then the policy SHALL evaluate to "NotApplicable".

If an error occurs while evaluating the target or condition of a rule, then the evaluation SHALL halt, and the policy shall evaluate to "Indeterminate", with the appropriate error status.

The following pseudo-code represents the evaluation strategy of this rule-combining algorithm.

```java
Decision firstApplicableEffectRuleCombiningAlgorithm(Rule rule[])
{
    for( i = 0 ; i < lengthOf(rule) ; i++ )
    {
...
Decision decision = evaluate(rule[i]);
if (decision == Deny)
    return Deny;
if (decision == Permit)
    return Permit;
if (decision == NotApplicable)
    continue;
if (decision == Indeterminate)
    return Indeterminate;
return NotApplicable;
}

The following specification defines the “First-applicable” policy-combining algorithm of a policy set.

Each policy is evaluated in the order that it appears in the policy set. For a particular policy, if the target evaluates to “True” and the policy evaluates to a determinate value of “Permit” or “Deny”, then the evaluation SHALL halt and the policy set SHALL evaluate to the effect value of that policy. For a particular policy, if the target evaluate to “False”, or the policy evaluates to “NotApplicable”, then the next policy in the order SHALL be evaluated. If no further policy exists in the order, then the policy set SHALL evaluate to “NotApplicable”.

If an error were to occur when evaluating the target, or when evaluating a specific policy, the reference to the policy is considered invalid, or the policy itself evaluates to “Indeterminate”, then the evaluation of the policy-combining algorithm shall halt, and the policy set shall evaluate to “Indeterminate” with an appropriate error status.

The following pseudo-code represents the evaluation strategy of this policy-combination algorithm.

Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy policy[]) {
    for (i = 0 ; i < lengthOf(policy) ; i++)
    {
        Decision decision = evaluate(policy[i]);
        if (decision == Deny)
            return Deny;
        if (decision == Permit)
            return Permit;
        if (decision == NotApplicable)
            continue;
        if (decision == Indeterminate)
            return Indeterminate;
    }
    return NotApplicable;
}
Obligations of the individual policies shall be combined as described in Section 7.11.

C.6. Only-one-applicable

The following specification defines the "Only-one-applicable" policy-combining algorithm of a policy set.

In the entire set of policies in the policy set, if no policy is considered applicable by virtue of their targets, then the result of the policy combination algorithm SHALL be "NotApplicable". If more than one policy is considered applicable by virtue of their targets, then the result of the policy combination algorithm SHALL be "Indeterminate".

If only one policy is considered applicable by evaluation of the policy targets, then the result of the policy-combining algorithm SHALL be the result of evaluating the policy.

If an error occurs while evaluating the target of a policy, or a reference to a policy is considered invalid or the policy evaluation results in "Indeterminate, then the policy set SHALL evaluate to "Indeterminate", with the appropriate error status.

The following pseudo-code represents the evaluation strategy of this policy combining algorithm.

```java
Decision onlyOneApplicablePolicyPolicyCombiningAlgorithm(Policy policy[]) {
    Boolean atLeastOne = false;
    Policy selectedPolicy = null;
    ApplicableResult appResult;

    for (i = 0; i < lengthOf(policy) ; i++)
    {
        appResult = isApplicable(policy[i]);
        if (appResult == Indeterminate)
        {
            return Indeterminate;
        }
        else
        {
            atLeastOne = true;
            selectedPolicy = policy[i];
        }
        if (appResult == NotApplicable)
        {
            continue;
        }
        else
        {
            return NotApplicable;
        };
    }
    return selectedPolicy;
}
```
Appendix D. Acknowledgments

The following individuals contributed to the development of the specification:

Anne Anderson
Bill Parducci
Carlisle Adams
Daniel Engovatov
Don Flinn
Ernesto Damiani
Gerald Brose
Hal Lockhart
James MacLean
John Merrells
Ken Yagen
Konstantin Beznosov
Michiharu Kudo
Pierangela Samarati
Pirasenna Velandai Thiyagarajan
Polar Humenn
Satoshi Hada
Sekhar Vajhala
Seth Proctor
Simon Godik
Steve Anderson
Steve Crocker
Suresh Damodaran
Tim Moses
## Appendix E. Revision history

<table>
<thead>
<tr>
<th>Rev</th>
<th>Date</th>
<th>By whom</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS V1.0</td>
<td>18 Feb 2003</td>
<td>XACML Technical Committee</td>
<td>OASIS Standard</td>
</tr>
</tbody>
</table>
Appendix F. Notices

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS’s procedures with respect to rights in OASIS specifications can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification, can be obtained from the OASIS Executive Director.

OASIS has been notified of intellectual property rights claimed in regard to some or all of the contents of this specification. For more information consult the online list of claimed rights.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.

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

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to OASIS, except as needed for the purpose of developing OASIS specifications, in which case the procedures for copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an “AS IS” basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.