Abstract:
This specification defines version 3.0 of the eXtensible Access Control Markup Language (XACML).

Status:
This document was last revised or approved by the OASIS eXtensible Access Control Markup Language (XACML) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the “Send A Comment” button on the Technical Committee’s web page at http://www.oasis-open.org/committees/xacml/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/xacml/ipr.php).

Citation format:
When referencing this specification the following citation format should be used:

[XACML-V3.0]
Notices

Copyright © OASIS Open 2012. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

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 section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, 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 OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

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' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee 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 implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.
Table of Contents

1  Introduction .................................................................................................................. 9
  1.1 Glossary (non-normative) .......................................................................................... 9
    1.1.1 Preferred terms ................................................................................................. 9
    1.1.2 Related terms ................................................................................................. 11
  1.2 Terminology .............................................................................................................. 11
  1.3 Schema organization and namespaces ................................................................... 12
  1.4 Normative References ............................................................................................ 12
  1.5 Non-Normative References ..................................................................................... 13
2  Background (non-normative) ....................................................................................... 14
  2.1 Requirements .......................................................................................................... 14
  2.2 Rule and policy combining ...................................................................................... 15
  2.3 Combining algorithms ............................................................................................ 15
  2.4 Multiple subjects .................................................................................................... 16
  2.5 Policies based on subject and resource attributes ................................................ 16
  2.6 Multi-valued attributes .......................................................................................... 16
  2.7 Policies based on resource contents ....................................................................... 16
  2.8 Operators ............................................................................................................... 17
  2.9 Policy distribution .................................................................................................. 17
  2.10 Policy indexing ..................................................................................................... 17
  2.11 Abstraction layer .................................................................................................. 18
  2.12 Actions performed in conjunction with enforcement ............................................ 18
  2.13 Supplemental information about a decision .......................................................... 18
3  Models (non-normative) ............................................................................................. 19
  3.1 Data-flow model ...................................................................................................... 19
  3.2 XACML context ........................................................................................................ 20
  3.3 Policy language model ............................................................................................ 21
    3.3.1 Rule ................................................................................................................ 21
    3.3.2 Policy ............................................................................................................. 22
    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 ............................................................................... 26
    4.1.3 Example response context ............................................................................ 28
  4.2 Example two ............................................................................................................ 28
    4.2.1 Example medical record instance .................................................................... 28
    4.2.2 Example request context ............................................................................... 29
    4.2.3 Example plain-language rules ........................................................................ 31
    4.2.4 Example XACML rule instances .................................................................... 31
5  Syntax (normative, with the exception of the schema fragments) .................................. 43
  5.1 Element <PolicySet> ............................................................................................... 43
  5.2 Element <Description> ......................................................................................... 45
  5.3 Element <PolicyIssuer> ......................................................................................... 45
5.4 Element <PolicySetDefaults> ................................................................. 45
5.5 Element <XPathVersion> ................................................................. 46
5.6 Element <Target> ............................................................................. 46
5.7 Element <AnyOf> ............................................................................. 46
5.8 Element <AllOf> .............................................................................. 47
5.9 Element <Match> .............................................................................. 47
5.10 Element <PolicySetIdReference> ......................................................... 48
5.11 Element <PolicyIdReference> ............................................................. 48
5.12 Simple type VersionType .................................................................. 48
5.13 Simple type VersionMatchType .......................................................... 49
5.14 Element <Policy> ............................................................................. 49
5.15 Element <PolicyDefaults> ................................................................. 51
5.16 Element <CombinerParameters> ......................................................... 51
5.17 Element <CombinerParameter> ........................................................... 52
5.18 Element <RuleCombinerParameters> .................................................... 52
5.19 Element <PolicyCombinerParameters> ................................................ 53
5.20 Element <PolicySetCombinerParameters> ............................................ 53
5.21 Element <Rule> ................................................................................. 54
5.22 Simple type EffectType ..................................................................... 55
5.23 Element <VariableDefinition> ............................................................ 55
5.24 Element <VariableReference> ............................................................ 55
5.25 Element <Expression> .................................................................... 56
5.26 Element <Condition> ...................................................................... 56
5.27 Element <Apply> ............................................................................. 56
5.28 Element <Function> ...................................................................... 57
5.29 Element <AttributeDesignator> ........................................................ 57
5.30 Element <AttributeSelector> ............................................................ 59
5.31 Element <AttributeValue> ................................................................. 60
5.32 Element <Obligations> .................................................................... 60
5.33 Element <AssociatedAdvice> ........................................................... 60
5.34 Element <Obligation> ..................................................................... 61
5.35 Element <Advice> .......................................................................... 61
5.36 Element <AttributeAssignment> ......................................................... 61
5.37 Element <ObligationExpressions> ...................................................... 62
5.38 Element <AdviceExpressions> ........................................................... 62
5.39 Element <ObligationExpression> ........................................................ 63
5.40 Element <AdviceExpression> ............................................................. 63
5.41 Element <AttributeAssignmentExpression> ....................................... 64
5.42 Element <Request> .......................................................................... 65
5.43 Element <RequestDefaults> ............................................................... 66
5.44 Element <Attributes> ..................................................................... 66
5.45 Element <Content> ......................................................................... 66
5.46 Element <Attribute> ....................................................................... 67
5.47 Element <Response> ....................................................................... 67
5.48 Element <Result> .......................................................................... 68
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.19.3 Missing attributes</td>
<td>86</td>
</tr>
<tr>
<td>7.20 Identifier equality</td>
<td>86</td>
</tr>
<tr>
<td>8 XACML extensibility points (non-normative)</td>
<td>88</td>
</tr>
<tr>
<td>8.1 Extensible XML attribute types</td>
<td>88</td>
</tr>
<tr>
<td>8.2 Structured attributes</td>
<td>88</td>
</tr>
<tr>
<td>9 Security and privacy considerations (non-normative)</td>
<td>89</td>
</tr>
<tr>
<td>9.1 Threat model</td>
<td>89</td>
</tr>
<tr>
<td>9.1.1 Unauthorized disclosure</td>
<td>89</td>
</tr>
<tr>
<td>9.1.2 Message replay</td>
<td>89</td>
</tr>
<tr>
<td>9.1.3 Message insertion</td>
<td>89</td>
</tr>
<tr>
<td>9.1.4 Message deletion</td>
<td>90</td>
</tr>
<tr>
<td>9.1.5 Message modification</td>
<td>90</td>
</tr>
<tr>
<td>9.1.6 NotApplicable results</td>
<td>90</td>
</tr>
<tr>
<td>9.1.7 Negative rules</td>
<td>90</td>
</tr>
<tr>
<td>9.1.8 Denial of service</td>
<td>91</td>
</tr>
<tr>
<td>9.2 Safeguards</td>
<td>91</td>
</tr>
<tr>
<td>9.2.1 Authentication</td>
<td>91</td>
</tr>
<tr>
<td>9.2.2 Policy administration</td>
<td>91</td>
</tr>
<tr>
<td>9.2.3 Confidentiality</td>
<td>92</td>
</tr>
<tr>
<td>9.2.4 Policy integrity</td>
<td>92</td>
</tr>
<tr>
<td>9.2.5 Policy identifiers</td>
<td>92</td>
</tr>
<tr>
<td>9.2.6 Trust model</td>
<td>93</td>
</tr>
<tr>
<td>9.2.7 Privacy</td>
<td>93</td>
</tr>
<tr>
<td>9.3 Unicode security issues</td>
<td>94</td>
</tr>
<tr>
<td>9.4 Identifier equality</td>
<td>94</td>
</tr>
<tr>
<td>10 Conformance</td>
<td>95</td>
</tr>
<tr>
<td>10.1 Introduction</td>
<td>95</td>
</tr>
<tr>
<td>10.2 Conformance tables</td>
<td>95</td>
</tr>
<tr>
<td>10.2.1 Schema elements</td>
<td>95</td>
</tr>
<tr>
<td>10.2.2 Identifier Prefixes</td>
<td>96</td>
</tr>
<tr>
<td>10.2.3 Algorithms</td>
<td>96</td>
</tr>
<tr>
<td>10.2.4 Status Codes</td>
<td>97</td>
</tr>
<tr>
<td>10.2.5 Attributes</td>
<td>97</td>
</tr>
<tr>
<td>10.2.6 Identifiers</td>
<td>97</td>
</tr>
<tr>
<td>10.2.7 Data-types</td>
<td>98</td>
</tr>
<tr>
<td>10.2.8 Functions</td>
<td>98</td>
</tr>
<tr>
<td>10.2.9 Identifiers planned for future deprecation</td>
<td>103</td>
</tr>
<tr>
<td>Appendix A. Data-types and functions (normative)</td>
<td>105</td>
</tr>
<tr>
<td>A.1 Introduction</td>
<td>105</td>
</tr>
<tr>
<td>A.2 Data-types</td>
<td>105</td>
</tr>
<tr>
<td>A.3 Functions</td>
<td>107</td>
</tr>
<tr>
<td>A.3.1 Equality predicates</td>
<td>107</td>
</tr>
<tr>
<td>A.3.2 Arithmetic functions</td>
<td>109</td>
</tr>
<tr>
<td>A.3.3 String conversion functions</td>
<td>110</td>
</tr>
<tr>
<td>A.3.4 Numeric data-type conversion functions</td>
<td>110</td>
</tr>
<tr>
<td>Appendix</td>
<td>Section</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>B.1</td>
<td>B.1</td>
</tr>
<tr>
<td>B.2</td>
<td>B.2</td>
</tr>
<tr>
<td>B.3</td>
<td>B.3</td>
</tr>
<tr>
<td>B.4</td>
<td>B.4</td>
</tr>
<tr>
<td>B.5</td>
<td>B.5</td>
</tr>
<tr>
<td>B.6</td>
<td>B.6</td>
</tr>
<tr>
<td>B.7</td>
<td>B.7</td>
</tr>
<tr>
<td>B.8</td>
<td>B.8</td>
</tr>
<tr>
<td>B.9</td>
<td>B.9</td>
</tr>
<tr>
<td>C.1</td>
<td>C.1</td>
</tr>
<tr>
<td>C.2</td>
<td>C.2</td>
</tr>
<tr>
<td>C.3</td>
<td>C.3</td>
</tr>
<tr>
<td>C.4</td>
<td>C.4</td>
</tr>
<tr>
<td>C.5</td>
<td>C.5</td>
</tr>
<tr>
<td>C.6</td>
<td>C.6</td>
</tr>
<tr>
<td>C.7</td>
<td>C.7</td>
</tr>
<tr>
<td>C.8</td>
<td>C.8</td>
</tr>
<tr>
<td>C.9</td>
<td>C.9</td>
</tr>
<tr>
<td>C.10</td>
<td>C.10</td>
</tr>
<tr>
<td>C.11</td>
<td>C.11</td>
</tr>
<tr>
<td>C.12</td>
<td>C.12</td>
</tr>
<tr>
<td>C.13</td>
<td>C.13</td>
</tr>
<tr>
<td>D.</td>
<td>D.</td>
</tr>
<tr>
<td>E.</td>
<td>E.</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Glossary (non-normative)

1.1.1 Preferred terms

Access
Performing an action

Access control
Controlling access in accordance with a policy or policy set

Action
An operation on a resource

Advice
A supplementary piece of information in a policy or policy set which is provided to the PEP with the decision of the PDP.

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 (see also – named attribute)

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 and advice

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 predicates 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, coordinates with Policy Information Points to add attribute values to the request context, 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 predicates 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

Identifier equality
The identifier equality operation which is defined in section 7.20.

Issuer
A set of attributes describing the source of a policy

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

Obligation
An operation specified in a rule, policy or policy set that should be performed by the PEP 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 or advice. 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. This term is defined in a joint effort by the IETF Policy Framework Working Group and the Distributed Management Task Force (DMTF)/Common Information Model (CIM) in [RFC3198]. This term corresponds to “Access Decision Function” (ADF) in [ISO10181-3].

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

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 or advice. 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, a condition and (optionally) a set of obligations or advice. 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 Terminology

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

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

Listings of XACML schema 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 3.0 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 bold-face italic are intended to have the meaning defined in the Glossary.

1.3 Schema organization and namespaces

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

urn:oasis:names:tc:xacml:3.0:core:schema:wd-17

1.4 Normative References


1.5 Non-Normative References


2 Background (non-normative)

The "economics of scale" have driven computing platform vendors to develop products with very generalised 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. Moreover, 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.
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 Policy, 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. Some examples of standard combining algorithms are defined (see Appendix C for a full list of standard combining algorithms):

- Deny-overrides (Ordered and Unordered),
- Permit-overrides (Ordered and Unordered),
- First-applicable and
- Only-one-applicable.

In the case of the Deny-overrides algorithm, 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 case of the Permit-overrides algorithm, 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 and condition 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.

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

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. Different attribute categories are used to differentiate between subjects acting in different capacities. Some standard values for these attribute categories 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 contained in the request context may be identified by the <AttributeDesignator> element. This element contains a URN that identifies the attribute. Alternatively, the <AttributeSelector> element may contain an XPath expression over the <Content> element of the subject 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 the resource may be identified by the <AttributeDesignator> element. This element contains a URN that identifies the attribute. Alternatively, the <AttributeSelector> element may contain an XPath expression over the <Content> element of the resource 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 (see Section A.3).

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
<Content> element of the resource 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 2.5.

### 2.8 Operators

Information security policies operate upon attributes of subjects, the resource, the action and the environment 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 Perritt [Perritt93]. 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. In this case, the **PDP** should form a database query to retrieve just those **policies** that are applicable to the set of decision requests to which it expects to respond. Additionally, the **PDP** should evaluate the `<Target>` element of the retrieved **policy or policy set** statements as defined by the XACML specification.

2. Alternatively, the **PDP** may be loaded with all available **policies** and evaluate their `<Target>` elements 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 was 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 independently 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 3.0 of XACML. Therefore, bilateral agreement between a **PAP** and the **PEP** that will enforce its **policies** is required for correct interpretation. **PEPs** that conform to v3.0 of XACML are required to deny **access** unless they understand and can discharge all of the `<Obligations>` elements associated with the applicable **policy**. `<Obligations>` elements are returned to the **PEP** for enforcement.

### 2.13 Supplemental information about a decision

In some applications it is helpful to specify supplemental information about a decision. XACML provides facilities to specify supplemental information about a decision with the `<Advice>` element. Such advice may be safely ignored by the **PEP**.
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.

![Data-flow diagram](image)

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 **subject**, **resource**, **action**, **environment** and other categories.

4. The **context handler** constructs an XACML request **context**, optionally adds **attributes**, and sends it to the **PDP**.

5. The PDP requests any additional **subject**, **resource**, **action**, **environment** and other categories (not shown) **attributes** from the **context handler**.

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

7. The **PIP** obtains the requested **attributes**.

8. The **PIP** returns the requested **attributes** to the **context handler**.

9. Optionally, the **context handler** includes the **resource** in the **context**.

10. The **context handler** sends the requested **attributes** and (optionally) the **resource** to the **PDP**. The **PDP** evaluates the **policy**.

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

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

13. The **PEP** fulfills the **obligations**.

14. (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 over the `<Content>` elements of the **context**, or attribute designators that identify the **attribute** by its category, identifier, data-type and (optionally) its issuer. 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-url)

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

Typical categories of **attributes** in the **context** are the **subject**, **resource**, **action** and **environment**, but users may define their own categories as needed. See appendix B.2 for suggested **attribute** categories.

See Section 7.3.5 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.

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**;
- a **condition**;
- **obligation** expressions, and
- **advice** expressions

These are discussed in the following sub-sections.
3.3.1.1 Rule target

The target defines the set of requests to which the rule is intended to apply in the form of a logical expression on attributes in the request. 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 the corresponding entity is omitted from the target. An XACML PDP verifies that the matches defined by the target are satisfied by the attributes in the request context.

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 URIs are examples of structured resource name-forms. 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 "med.example.com" is a legal RFC822 name identifying the set of mail addresses hosted by the med.example.com mail server. The XPath value md:record/md:patient/ is a legal XPath 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 10.2.9).

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.

3.3.1.4 Obligation expressions

Obligation expressions may be added by the writer of the rule.

When a PDP evaluates a rule containing obligation expressions, it evaluates the obligation expressions into obligations and returns certain of those obligations to the PEP in the response context. Section 7.18 explains which obligations are to be returned.

3.3.1.5 Advice

Advice expressions may be added by the writer of the rule.

When a PDP evaluates a rule containing advice expressions, it evaluates the advice expressions into advice and returns certain of those advice to the PEP in the response context. Section 7.18 explains which advice are to be returned. In contrast to obligations, advice may be safely ignored by the PEP.

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;
• obligation expressions and
• advice expressions

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

3.3.2.1 Policy target

An XACML <PolicySet>, <Policy> or <Rule> element contains a <Target> element that specifies the set of requests 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. A policy may have combining parameters that affect the operation of the rule-combining algorithm.

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

3.3.2.3 Obligation expressions

Obligation expressions may be added by the writer of the policy.

When a PDP evaluates a policy containing obligation expressions, it evaluates the obligation expressions into obligations and returns certain of those obligations to the PEP in the response context. Section 7.18 explains which obligations are to be returned.

3.3.2.4 Advice

Advice expressions may be added by the writer of the policy.

When a PDP evaluates a policy containing advice expressions, it evaluates the advice expressions into advice and returns certain of those advice to the PEP in the response context. Section 7.18 explains which advice are to be returned. In contrast to obligations, advice may be safely ignored by the PEP.
3.3.3 Policy set

A policy set comprises four main components:

- a target;
- a policy-combining algorithm-identifier
- a set of policies;
- obligation expressions, and
- advice expressions

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. A policy set may have combining parameters that affect the operation of the policy-combining algorithm.

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

3.3.3.2 Obligation expressions

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

When a PDP evaluates a policy set containing obligations expressions, it evaluates the obligation expressions into obligations and returns certain of those obligations to the PEP in its response context. Section 7.18 explains which obligations are to be returned.

3.3.3.3 Advice expressions

Advice expressions may be added by the writer of the policy set.

When a PDP evaluates a policy set containing advice expressions, it evaluates the advice expressions into advice and returns certain of those advice to the PEP in the response context. Section 7.18 explains which advice are to be returned. In contrast to obligations, advice may be safely ignored by the PEP.
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 (identified by its domain name: med.example.com) has an access control policy that states, in English:

Any user with an e-mail name in the "med.example.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 obligation expressions.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17.xsd"
    PolicyId="urn:oasis:names:tc:xacml:3.0:example:SimplePolicy1"
    Version="1.0"
    RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-overrides">
  <Description>Medi Corp access control policy</Description>
  <Target/>
  <Rule RuleId= "urn:oasis:names:tc:xacml:3.0:example:SimpleRule1" Effect="Permit">
    <Description>Any subject with an e-mail name in the med.example.com domain can perform any action on any resource.</Description>
    <Target>
      <AnyOf>
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">med.example.com</AttributeValue>
            <AttributeDesignator MustBePresent="false" Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
              <AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
            </AttributeDesignator>
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
  </Rule>
</Policy>
```

[a1] is a standard XML document tag indicating which version of XML is being used and what the character encoding is.

[a2] introduces the XACML Policy itself.
[a3] - [a4] are XML namespace declarations.
[a3] gives a URN for the XACML policies schema.
[a7] assigns a name to this policy instance. The name of a policy has to be unique for a given PDP so that there is no ambiguity if one policy is referenced from another policy. The version attribute specifies the version of this policy is “1.0”.
[a9] 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 the 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.
[a10] - [a12] provide a text description of the policy. This description is optional.
[a13] describes the decision requests to which this policy applies. If the attributes in a decision request do not match the values specified in the policy target, then the remainder of the policy does not need to be evaluated. This target section is 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.
[a14] introduces the one and only rule in this simple policy.
[a15] specifies the identifier for this rule. Just as for a policy, each rule must have a unique identifier (at least unique for any PDP that will be using the policy).
[a16] 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, if the rule is satisfied, it 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, then the rule returns a result of “Indeterminate”. As mentioned above, the rule-combining algorithm for the policy specifies how various rule values are combined into a single policy value.
[a17] - [a20] provide a text description of this rule. This description is optional.
[a21] 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 attributes 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 rule evaluation.
The rule target is similar to the target of the policy itself, but with one important difference. [a22] - [a36] spells out a specific value that the subject in the decision request must match. The <Match> element specifies a matching function in the MatchId attribute, a literal value of “med.example.com” and a pointer to a specific subject attribute in the request context by means of the <AttributeDesignator> element with an attribute category which specifies the access subject. The matching function will be used to compare the literal value with the value of the subject attribute. 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”.
[a38] closes the rule. 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> element (which could also be a set of conditions to be ANDed or ORed together).
[a39] closes the policy. 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 that executes 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:

xacml-3.0-core-spec-csprd04-en Standards Track Work Product Copyright © OASIS Open 2012. All Rights Reserved.
19 April 2012 Page 26 of 154
[b1] - [b2] contain the header information for the request context, and are used the same way as the header for the policy explained above.

The first <Attributes> element contains attributes of the entity making the access request. There can be multiple subjects in the form of additional <Attributes> elements with different categories, and each subject can have multiple attributes. In this case, in [b6] - [b13], 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 second <Attributes> element contains attributes of the resource to which the subject (or subjects) has requested access. Lines [b14] - [b21] contain the one attribute of the resource to which Bart Simpson has requested access: the resource identified by its file URI, which is "file://medico/record/patient/BartSimpson".

The third <Attributes> element contains attributes of the action that the subject (or subjects) wishes to take on the resource. [b22] - [b29] describe the identity of the action Bart Simpson wishes to take, which is "read".

[b30] closes the request context. A more complex request context may have contained some attributes not associated with the subject, the resource or the action. Environment would be an example of such an attribute category. These would have been placed in additional <Attributes> elements. Examples of such attributes are attributes describing the environment or some application specific category of attributes.

The PDP processing this request context locates the policy in its policy repository. It compares the attributes in the request context with the policy target. Since the policy target is empty, the policy matches this context.

The PDP now compares the attributes in the request context with the target of the one rule in this policy. The requested resource matches the <Target> element and the requested action matches the <Target> element, but the requesting subject-id attribute does not match "med.example.com".
4.1.3 Example response context

As a result of evaluating the policy, 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:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <Result>
        <Decision>NotApplicable</Decision>
    </Result>
</Response>
```

[c1] - [c2] contain the same sort of header information for the response as was described above for a policy.

The <Result> element in lines [c3] - [c5] 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”. Therefore, the PEP is required to deny access.

[c6] 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 obligation expressions.

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 Medi Corp.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<record xmlns="urn:example:med:schemas:record"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <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>
        </patientContact>
        <patientDoB>1992-03-21</patientDoB>
    </patient>
    <parentGuardian>
        <parentGuardianId>HS001</parentGuardianId>
        <parentGuardianName>
            <first>Homer</first>
            <last>Simpson</last>
        </parentGuardianName>
        <parentGuardianContact>
            <street>27 Shelbyville Road</street>
            <city>Springfield</city>
            <zip>12345</zip>
            <phone>555.123.4567</phone>
        </parentGuardianContact>
        <patient-number>555555</patient-number>
    </parentGuardian>
</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.

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
         ReturnPolicyIdList="false">
  <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <Attribute IncludeInResult="false">
      <AttributeId="urn:oasis:names:tc:xacml:1.0:example:attribute:role"
        Issue="med.example.com">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">CN=Julius Hibbert</AttributeValue>
      </Attribute>
    </Attribute>
  </Attributes>
</Request>
```
3.0 Standard namespace declarations.

Access subject attributes are placed in the urn:oasis:names:tc:xacml:3.0:subject-category:access-subject attribute category of the <Request> element. Each attribute consists of the attribute meta-data and the attribute value. There is only one subject involved in this request. This value of the attribute category denotes the identity for which the request was issued.

Subject subject-id attribute.

Subject role attribute.

Subject physician-id attribute.
Resource attributes are placed in the urn:oasis:names:tc:xacml:3.0:attribute-category:resource attribute category of the <Request> element. Each attribute consists of attribute meta-data and an attribute value.

Resource content. The XML resource instance, access to all or part of which may be requested, is placed here.

The identifier of the Resource instance for which access is requested, which is an XPath expression into the <Content> element that selects the data to be accessed.

Action attributes are placed in the urn:oasis:names:tc:xacml:3.0:attribute-category:action attribute category of the <Request> element.

Action identifier.

4.2.3 Example plain-language rules

The following plain-language rules are to be enforced:

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

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.

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.

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. It also illustrates the use of the <VariableDefinition> element to define a function that may be used throughout the policy. The following XACML <Rule> instance expresses Rule 1:

```
<Policy Version="1.0">
  <PolicyDefaults />
  <Target />
  <VariableDefinition VariableId="17590034"> </VariableDefinition>
  <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">
      <AttributeDesignator MustBePresent="false"
        Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
        AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:patient-number">
        <Content DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </Apply>
    </Apply>
  </Apply>
</Policy>
```
A person may read any medical record in the http://www.med.example.com/schemas/record.xsd namespace for which he or she is the designated patient.

XPath expressions in the 3.0 core XML namespace decl -spec.

For which he or she is the designated patient. It defines a function that evaluates the truth of the statement: the patient-number subject attribute is equal to the patient-number in the resource.


[111] XPath expressions in the policy are to be interpreted according to the 1.0 version of the XPath specification.

[14] - [133] A <VariableDefinition> element. It defines a function that evaluates the truth of the statement: the patient-number subject attribute is equal to the patient-number in the resource.
The FunctionId attribute names the function to be used for comparison. In this case, comparison is done with the "urn:oasis:names:tc:xacml:1.0:function:string-equal" function; this function takes two arguments of type "http://www.w3.org/2001/XMLSchema#string".

The first argument of the variable definition is a function specified by the FunctionId attribute. Since "urn:oasis:names:tc:xacml:1.0:function:string-equal" takes arguments of type "http://www.w3.org/2001/XMLSchema#string" and AttributeDesignator selects a bag of type "http://www.w3.org/2001/XMLSchema#string", "urn:oasis:names:tc:xacml:1.0:function:string-one-and-only" is used. This function guarantees that its argument evaluates to a bag containing exactly one value.

The AttributeDesignator selects a bag of values for the patient-number subject attribute in the request context.

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

The <AttributeSelector> element selects a bag of values from the resource content using a free-form XPath expression. In this case, it selects the value of the patient-number in the resource. Note that the namespace prefixes in the XPath expression are resolved with the standard XML namespace declarations.

Rule identifier.

Rule effect declaration. When a rule evaluates to ‘True’ it emits the value of the Effect attribute. This value is then 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 the rule is intended to evaluate.

The <AnyOf> element contains a disjunctive sequence of <AllOf> elements. In this example, there is just one.

The <AllOf> element encloses the conjunctive sequence of Match elements. In this example, there are two.

The first <Match> 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 "urn:example:med:schemas:record".

The MatchId attribute names the matching function.

Literal attribute value to match.

The <AttributeDesignator> element selects the target namespace from the resource contained in the request context. The attribute name is specified by the AttributeId.

The second <Match> element. This match compares the results of two XPath expressions applied to the <Content> element of the resource category. The second XPath expression is the location path to the requested XML element and the first XPath expression is the literal value "md:record". The "xpath-node-match" function evaluates to "True" if the requested XML element is below the "md:record" element.

The <AnyOf> element contains a disjunctive sequence of <AllOf> elements. In this case, there is just one <AllOf> element.

The <AllOf> element contains a conjunctive sequence of <Match> elements. In this case, there is just one <Match> element.
The <Match> element compares its first and second child elements according to the matching function. The 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 attribute in the request context is compared with the literal value "read".

The <Condition> element. A condition must evaluate to "True" for the rule to be applicable. This condition contains a reference to a variable definition defined elsewhere in the policy.

### 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:date-add-yearMonthDuration" to calculate the date of the patient’s sixteenth birthday. It also illustrates the use of predicate expressions, with the functionId "urn:oasis:names:tc:xacml:1.0:function:and". This example has one function embedded in the <Condition> element and another one referenced in a <VariableDefinition> element.
[g16] - [g41] The <VariableDefinition> element contains part of the condition (i.e. is the patient under 16 years of age?). 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.

The <VariableDefinition> element contains part of the condition (i.e. is the patient under 16 years of age?). 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.

[g16] - [g40] "urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal" is used to compare the two date arguments.

```
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
    <Match>
      <VariableReference VariableId="17590035"/>
    </Match>
    <Match>
      <VariableReference VariableId="17590035"/>
    </Match>
  </Apply>
  <Match>
    <AttributeSelector Category="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
      <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action"
        MustBePresent="false"
      />
      <AttributeSelector Category="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
        <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action"
          MustBePresent="false"
        />
      </AttributeSelector>
      <AttributeSelector Category="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
        <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action"
          MustBePresent="false"
        />
      </AttributeSelector>
    </AttributeSelector>
    <AttributeSelector Category="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
      <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action"
        MustBePresent="false"
      />
    </AttributeSelector>
    <Match>
      <VariableReference VariableId="17590035"/>
    </Match>
    <Match>
      <VariableReference VariableId="17590035"/>
    </Match>
  </Apply>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
      <Match>
        <VariableReference VariableId="17590035"/>
      </Match>
      <Match>
        <VariableReference VariableId="17590035"/>
      </Match>
    </Apply>
  </Apply>
</Apply>
```

[g18] - [g25] The first date argument uses “urn:oasis:names:tc:xacml:1.0:function:date-one-and-only” to ensure that the *bag* of values selected by its argument contains exactly one value of type

“http://www.w3.org/2001/XMLSchema#date”.

[g20] The current date is evaluated by selecting the “urn:oasis:names:tc:xacml:1.0:environment:current-date” *environment attribute*.

[g26] - [g39] The second date argument uses “urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration” to compute the date of the patient’s sixteenth birthday by adding 16 years to the patient’s date of birth. The first of its arguments is of type “http://www.w3.org/2001/XMLSchema#date” and the second is of type “http://www.w3.org/TR/2007/REC-xpath-functions-20070123/#dt-yearMonthDuration”.

[g30] The `<AttributeSelector>` element selects the patient’s date of birth by taking the XPath expression over the *resource* content.

[g36] - [g38] Year Month Duration of 16 years.

[g51] - [g92] Rule declaration and *rule target*. See *Rule* 1 in Section 4.2.4.1 for the detailed explanation of these elements.

[g93] - [g115] The `<Condition>` element. The *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. It contains one embedded `<Apply>` element and one referenced `<VariableDefinition>` element.

[g94] The *condition* uses 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.

[g95] - [g112] The first part of the *condition* is evaluated (i.e. is the requestor the designated parent or guardian?). The function is “urn:oasis:names:tc:xacml:1.0:function:string-equal” and it takes two arguments of type “http://www.w3.org/2001/XMLSchema#string”.


[g98] designates the first argument. The value of the *subject attribute* “urn:oasis:names:tc:xacml:3.0:example:attribute:parent-guardian-id” is selected from the request *context* using the `<AttributeDesignator>` element.

[g104] As above, the “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used to ensure that the *bag* of values selected by it’s argument contains exactly one value of type “http://www.w3.org/2001/XMLSchema#string”.

[g106] The second argument selects the value of the `<md:parentGuardianId>` element from the *resource* content using the `<AttributeSelector>` element. This element contains a free-form XPath expression, pointing into the `<Content>` element of the resource category. Note that all namespace prefixes in the XPath expression are resolved with standard namespace declarations. The *AttributeSelector* evaluates to the *bag* of values of type “http://www.w3.org/2001/XMLSchema#string”.

[g113] references the `<VariableDefinition>` element, where the second part of the *condition* is defined.

**4.2.4.3 Rule 3**

*Rule* 3 illustrates the use of an *obligation* expression.

```xml
<xml version="1.0" encoding="UTF-8"?
Policy
 xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
 xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```
<PolicyId>urn:oasis:names:tc:xacml:3.0:example:policyid:3</PolicyId>

Version=“1.0”

RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides”>

<Description>
Policy for any medical record in the http://www.med.example.com/schemas/record.xsd namespace
</Description>

<XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>

<Target>
<AnyOf>
<AttributeDesignator>
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
</AttributeValue>
</AttributeDesignator>
</AnyOf>
</Target>

<Rule Id="urn:oasis:names:tc:xacml:1.0:ruleid:3"
Effect="Permit">
<Description>
A physician may write any medical element in a record for which he or she is the designated primary care physician, provided an email is sent to the patient
</Description>
<Target>
<AnyOf>
<AttributeDesignator>
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
</AttributeValue>
</AttributeDesignator>
</AnyOf>
</Rule>

<Rule Id="urn:oasis:names:tc:xacml:1.0:ruleid:3"
Effect="Permit">
<Description>
A physician may write any medical element in a record for which he or she is the designated primary care physician, provided an email is sent to the patient
</Description>
<Target>
<AnyOf>
<AttributeDesignator>
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
</AttributeValue>
</AttributeDesignator>
</AnyOf>
</Rule>
The <Policy> element includes standard namespace declarations as well as policy specific parameters, such as PolicyId and RuleCombiningAlgId.

Policy identifier. This parameter allows the policy to be referenced by a policy set.

The Rule-combining algorithm identifies the algorithm for combining the outcomes of rule evaluation.

Free-form description of the policy.

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>
The only <Rule> element included in this <Policy>. Two parameters are specified in the 
rule header: RuleId and Effect.

The rule target further constrains the policy target. The <Match> element targets the rule at subjects whose 
"urn:oasis:names:tc:xacml:3.0:example:attribute:role" subject attribute is equal to "physician".

The <Match> element targets the rule at resources that match the XPath expression 
"md:record/md:medical".

The <Match> element targets the rule at actions whose 
"urn:oasis:names:tc:xacml:1.0:action:action" action attribute is equal to "write".

The <Condition> element. For the rule to be applicable to the decision request, the 
condition must evaluate to "True". This condition compares the value of the 
"urn:oasis:names:tc:xacml:3.0:example:attribute:physician-id" subject attribute with the value of the 
<registrationId> element in the medical record that is being accessed.

The <ObligationExpression> 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 "Permit" or "Deny" authorization decision. The element contains a single obligation expression, which will be evaluated into an obligation when the policy is evaluated.

The <ObligationExpression> element consists of the ObligationId attribute, the 
authorization decision value for which it must be fulfilled, and a set of attribute assignments.

The ObligationId attribute identifies the obligation. In this case, the PEP is required to send email.

The FulfillOn attribute defines the authorization decision value for which the obligation derived from the obligation expression must be fulfilled. In this case, the obligation must be fulfilled when access is permitted.

The first parameter indicates where the PEP will find the email address in the resource. The PDP will evaluate the <AttributeSelector> and return the result to the PEP inside the resulting obligation.

The second parameter contains literal text for the email body.

The third parameter indicates where the PEP will find further text for the email body in the resource. The PDP will evaluate the <AttributeDesignator> and return the result to the PEP inside the resulting obligation.

4.2.4.4 Rule 4

Rule 4 illustrates the use of the "Deny" Effect value, and a <Rule> with no <Condition> element.
medical elements of a patient record in the
http://www.med.example.com/records.xsd namespace.
</Description>
</Target>
<AnyOf>
<Match>
  <AttributeValue
    MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#string">
      administrator</AttributeValue>

  <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
    AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
    DataType="http://www.w3.org/2001/XMLSchema#string"/>
</AnyOf>
</AnyOf>
<Match>
  <AttributeValue
    MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
    <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#anyURI">
      urn:example:med:schemas:record</AttributeValue>

  <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
    AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
    DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
</Match>
</AnyOf>
<Match>
  <AttributeDesignator
    XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
    AttributeId="urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
    DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
</Match>
</AnyOf>
<Match>
  <AttributeValue
    MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
    <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#anyURI">
      md:record/md:medical</AttributeValue>

  <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
    AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
    DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
</Match>
</AnyOf>
<AnyOf>
<Match>
  <AttributeValue
    MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#string">
      read</AttributeValue>

  <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
    AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
    DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
</Match>
</AnyOf>
<AnyOf>
<Match>
  <AttributeValue
    MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#string">
      write</AttributeValue>

  <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
    AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
    DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
</Match>
</AnyOf>
<AnyOf>
<Match>
  <AttributeValue
    MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#string">
      True</AttributeValue>

  <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
    AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
    DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
</Match>
</AnyOf>
The Rule effect. Every rule that evaluates to “True” emits the rule effect as its value. This rule is “Deny” meaning that according to this rule, access must be denied when it evaluates to “True”.

Free form description of the rule.

The Rule target. The Rule target defines the set of decision requests that are applicable to the rule.

The <Match> element targets the rule at subjects whose

The <AnyOf> element contains one <AllOf> element, which (in turn) contains two <Match> elements. The target matches if the resource identified by the request context matches both resource match criteria.

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 policy set illustrates the combined policies. Policy 3 is included by reference and policy 2 is explicitly included.

```xml
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
PolicySetId="urn:oasis:names:tc:xacml:3.0:example:policysetid:1"
Version="1.0"
PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides">
  <Description>
    Example policy set.
  </Description>
  <Target>
    <AnyOf>
      <AllOf>
        <Match>
          <AttributeDesignator
            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">
              urn:example:med:schemas:record</AttributeValue>
          </AttributeDesignator>
          <AttributeDesignator
            MustBePresent="false">
            <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">
              urn:oasis:names:tc:xacml:2.0:resource:target
            </AttributeValue>
          </AttributeDesignator>
        </Match>
      </AllOf>
      <AnyOf>
        <AllOf>
          <Match>
            <AttributeDesignator
              MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue
               (DataType="http://www.w3.org/2001/XMLSchema#string">
                urn:example:med:schemas:record</AttributeValue>
            </AttributeDesignator>
            <AttributeDesignator
              MustBePresent="false">
              <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#string">
                urn:oasis:names:tc:xacml:2.0:resource:target
            </AttributeValue>
          </AttributeDesignator>
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
</PolicySet>
```
The PolicySet element declaration. Standard XML namespace declarations are included.

The PolicySetId attribute is used for identifying this policy set for possible inclusion in another policy set.

The policy-combining algorithm identifier. Policies and policy sets in this policy set are combined according to the specified policy-combining algorithm when the authorization decision is computed.

Free form description of the policy set.

The policy set <Target> element defines the set of decision requests that are applicable to this <PolicySet> element.

PolicyIdReference includes a policy by id.

Policy 2 is explicitly included in this policy set. The rules in Policy 2 are omitted for clarity.
5 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 either directly using the <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.

A <PolicySet> element may be evaluated, in which case the evaluation procedure defined in Section 7.13 SHALL be used.

If a <PolicySet> element contains references to other policy sets or policies in the form of URLs, then these references MAY be resolvable. Policy sets and policies included in a <PolicySet> element MUST be combined using the algorithm identified by the PolicyCombiningAlgId attribute. <PolicySet> is treated exactly like a <Policy> in all policy-combining algorithms.

A <PolicySet> element MAY contain a <PolicyIssuer> element. The interpretation of the <PolicyIssuer> element is explained in the separate administrative policy profile [XACMLAdmin].

The <Target> element defines the applicability of the <PolicySet> element to a set of decision requests. If the <Target> element within the <PolicySet> element matches the request context, then the <PolicySet> element MAY be used by the PDP in making its authorization decision. See Section 7.13.

The <ObligationExpressions> element contains a set of obligation expressions that MUST be evaluated into obligations by the PDP and the resulting obligations MUST be fulfilled by the PEP in conjunction with the authorization decision. If the PEP does not understand or cannot fulfill any of the obligations, then it MUST act according to the PEP bias. See Section 7.2 and 7.18.

The <AdviceExpressions> element contains a set of advice expressions that MUST be evaluated into advice by the PDP. The resulting advice MAY be safely ignored by the PEP in conjunction with the authorization decision. See Section 7.18.

```
<xs:element name="PolicySet" type="xacml:PolicySetType"/>
<xs:complexType name="PolicySetType">
  <xs:sequence>
    <xs:element ref="xacml:Description" minOccurs="0"/>
    <xs:element ref="xacml:PolicyIssuer" 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:element ref="xacml:CombinerParameters"/>
      <xs:element ref="xacml:PolicyCombinerParameters"/>
    </xs:choice>
    <xs:element ref="xacml:ObligationExpressions" minOccurs="0"/>
    <xs:element ref="xacml:AdviceExpressions" minOccurs="0"/>
  </xs:sequence>
```
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.

- **Version [Required]**
  - The version number of the PolicySet.

- **PolicyCombiningAlgId [Required]**
  - The identifier of the policy-combining algorithm by which the `<PolicySet>`, `<CombinerParameters>`, `<PolicyCombinerParameters>` and `<PolicySetCombinerParameters>` components MUST be combined. Standard policy-combining algorithms are listed in Appendix C. Standard policy-combining algorithm identifiers are listed in Section B.9.

- **MaxDelegationDepth [Optional]**
  - If present, limits the depth of delegation which is authorized by this policy set. See the delegation profile [XACMLAdmin].

- **Description [Optional]**
  - A free-form description of the policy set.

- **PolicyIssuer [Optional]**
  - Attributes of the issuer of the policy set.

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

- **Target [Required]**
  - The `<Target>` element defines the applicability of a policy set 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 that is included in this policy set.

- **Policy [Any Number]**
  - A policy that is included in this policy set.

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

- **PolicyIdReference [Any Number]**
  - A reference to a policy that MUST be included in this policy set. If the `<PolicyIdReference>` is a URL, then it MAY be resolvable.
<ObligationExpressions> [Optional]
Contains the set of <ObligationExpression> elements. See Section 7.18 for a description of how the set of obligations to be returned by the PDP shall be determined.

<AdviceExpressions> [Optional]
Contains the set of <AdviceExpression> elements. See Section 7.18 for a description of how the set of advice to be returned by the PDP shall be determined.

<CombinerParameters> [Optional]
Contains a sequence of <CombinerParameter> elements. The parameters apply to the combining algorithm as such and it is up to the specific combining algorithm to interpret them and adjust its behavior accordingly.

<PolicyCombinerParameters> [Optional]
Contains a sequence of <CombinerParameter> elements that are associated with a particular <Policy> or <PolicyIdReference> element within the <PolicySet>. It is up to the specific combining algorithm to interpret them and adjust its behavior accordingly.

<PolicySetCombinerParameters> [Optional]
Contains a sequence of <CombinerParameter> elements that are associated with a particular <PolicySet> or <PolicySetIdReference> element within the <PolicySet>. It is up to the specific combining algorithm to interpret them and adjust its behavior accordingly.

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

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

5.3 Element <PolicyIssuer>
The <PolicyIssuer> element contains attributes describing the issuer of the policy or policy set. The use of the policy issuer element is defined in a separate administration profile [XACMLAdmin]. A PDP which does not implement the administration profile MUST report an error or return an Indeterminate result if it encounters this element.

<xs:element name="PolicyIssuer" type="xacml:PolicyIssuerType"/>
<xs:complexType name="PolicyIssuerType">
<xs:sequence>
<xs:element ref="xacml:Content" minOccurs="0"/>
<xs:element ref="xacml:Attribute" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

The <PolicyIssuer> element is of PolicyIssuerType complex type. The <PolicyIssuer> element contains the following elements:

<Content> [Optional]
Free form XML describing the issuer. See Section 5.45.

<Attribute> [Zero to many]
An attribute of the issuer. See Section 5.46.

5.4 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"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

The <PolicySetDefaults> element is of DefaultsType complex type.

The <PolicySetDefaults> element contains the following elements:

5.5 Element <XPathVersion>

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

The URI for the XPath 1.0 specification is "http://www.w3.org/TR/1999/REC-xpath-19991116".

The URI for the XPath 2.0 specification is "http://www.w3.org/TR/2007/REC-xpath2-20070123".

The <XPathVersion> element is REQUIRED if the XACML enclosing policy set or policy contains <AttributeSelector> elements or XPath-based functions.

5.6 Element <Target>

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 a <PolicySet> and <Policy> element and MAY appear as a child of a <Rule> element.

The <Target> element SHALL contain a conjunctive sequence of <AnyOf> 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 <AnyOf> element of the <Target> element and the corresponding section of the <Request> element.

The <Target> element is of TargetType complex type.

The <Target> element contains the following elements:

5.7 Element <AnyOf>

The <AnyOf> element SHALL contain a disjunctive sequence of <AllOf> elements.
<xs:sequence>
  </xs:complexType>

The `<AnyOf>` element is of `<AnyOfType>` complex type.

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

- `<AllOf>` [One to Many, Required]
  
  See Section 5.8.

### 5.8 Element `<AllOf>`

The `<AllOf>` element SHALL contain a **conjunctive sequence** of `<Match>` elements.

```xml
<xs:element name="AllOf" type="xacml:AllOfType"/>
<xs:complexType name="AllOfType">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element ref="xacml:Match"/>
  </xs:sequence>
</xs:complexType>
```

The `<AllOf>` element is of `<AllOfType>` complex type.

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

- `<Match>` [One to Many]
  
  A **conjunctive sequence** of individual matches of the `attributes` in the request `context` and the embedded `attribute` values. See Section 5.9.

### 5.9 Element `<Match>`

The `<Match>` element SHALL identify a set of entities by matching `attribute` values in an `<Attributes>` element of the request `context` with the embedded `attribute` value.

```xml
<xs:element name="Match" type="xacml:MatchType"/>
<xs:complexType name="MatchType">
  <xs:sequence>
    <xs:element ref="xacml:AttributeValue"/>
    <xs:choice>
      <xs:element ref="xacml:AttributeDesignator"/>
      <xs:element ref="xacml:AttributeSelector"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
</xs:complexType>
```

The `<Match>` element is of `<MatchType>` complex type.

The `<Match>` 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 7.6.

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

- **AttributeDesignator** [Required choice]
  
  MAY be used to identify one or more `attribute` values in an `<Attributes>` element of the request `context`.

- **AttributeSelector** [Required choice]
MAY be used to identify one or more attribute values in a <Content> element of the request context.

5.10 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> element. However, the mechanism for resolving a policy set reference to the corresponding policy set is outside the scope of this specification.

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

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

- Version [Optional]
  Specifies a matching expression for the version of the policy set referenced.

- EarliestVersion [Optional]
  Specifies a matching expression for the earliest acceptable version of the policy set referenced.

- LatestVersion [Optional]
  Specifies a matching expression for the latest acceptable version of the policy set referenced.

The matching operation is defined in Section 5.13. Any combination of these attributes MAY be present in a <PolicySetIdReference>. The referenced policy set MUST match all expressions. If none of these attributes is present, then any version of the policy set is acceptable. In the case that more than one matching version can be obtained, then the most recent one SHOULD be used.

5.11 Element <PolicyIdReference>

The <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> element. However, the mechanism for resolving a policy reference to the corresponding policy is outside the scope of this specification.

Element <PolicyIdReference> is of xacml:IdReferenceType complex type (see Section 5.10).

5.12 Simple type VersionType

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

```
<xs:simpleType name="VersionType">
  <xs:restriction base="xs:string">
    <xs:pattern value="([d]+.)*[d]+"/>
  </xs:restriction>
</xs:simpleType>
```
The version number is expressed as a sequence of decimal numbers, each separated by a period (.).
'd+' represents a sequence of one or more decimal digits.

5.13 Simple type VersionMatchType

Elements of this type SHALL contain a restricted regular expression matching a version number (see Section 5.12). The expression SHALL match versions of a referenced policy or policy set that are acceptable for inclusion in the referencing policy or policy set.

```
<xs:simpleType name="VersionMatchType">
  <xs:restriction base="xs:string">
    <xs:pattern value="((\d+|\*)\.(\d+|\*)\+|\d+|\*)"/>
  </xs:restriction>
</xs:simpleType>
```

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

5.14 Element <Policy>

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

A <Policy> element may be evaluated, in which case the evaluation procedure defined in Section 7.12 SHALL be used.

The main components of this element are the <Target>, <Rule>, <CombinerParameters>, <RuleCombinerParameters>, <ObligationExpressions> and <AdviceExpressions> elements and the RuleCombiningAlgId attribute.

A <Policy> element MAY contain a <PolicyIssuer> element. The interpretation of the <PolicyIssuer> element is explained in the separate administrative policy profile [XACMLAdmin].

The <Target> element defines the applicability of the <Policy> element to a set of decision requests. If the <Target> element within the <Policy> element matches the request context, then the <Policy> element MAY be used by the PDP in making its authorization decision. See Section 7.12.

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

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

The <ObligationExpressions> element contains a set of obligation expressions that MUST be evaluated into obligations by the PDP and the resulting obligations MUST be fulfilled by the PEP in conjunction with the authorization decision. If the PEP does not understand, or cannot fulfill, any of the obligations, then it MUST act according to the PEP bias. See Section 7.2 and 7.18.

The <AdviceExpressions> element contains a set of advice expressions that MUST be evaluated into advice by the PDP. The resulting advice MAY be safely ignored by the PEP in conjunction with the authorization decision. See Section 7.18.
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.

- **Version** [Required]
  - The version number of the Policy.

- **RuleCombiningAlgId** [Required]
  - The identifier of the rule-combining algorithm by which the <Policy>, <CombinerParameters> and <RuleCombinerParameters> components MUST be combined. Standard rule-combining algorithms are listed in Appendix C. Standard rule-combining algorithm identifiers are listed in Section B.9.

- **MaxDelegationDepth** [Optional]
  - If present, limits the depth of delegation which is authorized by this policy. See the delegation profile [XACMLAdmin].

- **<Description>** [Optional]
  - A free-form description of the policy. See Section 5.2.

- **<PolicyIssuer>** [Optional]
  - Attributes of the issuer of the policy.

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

- **<CombinerParameters>** [Optional]
  - A sequence of parameters to be used by the rule-combining algorithm. The parameters apply to the combining algorithm as such and it is up to the specific combining algorithm to interpret them and adjust its behavior accordingly.

- **<RuleCombinerParameters>** [Optional]
  - A sequence of <RuleCombinerParameter> elements that are associated with a particular <Rule> element within the <Policy>. It is up to the specific combining algorithm to interpret them and adjust its behavior accordingly.

- **<Target>** [Required]
  - The <Target> element defines 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.
<VariableDefinition> [Any Number]

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

A sequence of rules that MUST be combined according to the RuleCombiningAlgId attribute.

Rules whose <Target> elements and conditions match the decision request MUST be considered. Rules whose <Target> elements or conditions do not match the decision request SHALL be ignored.

A conjunctive sequence of obligation expressions which MUST be evaluated into obligations by the PDP. The corresponding obligations MUST be fulfilled by the PDP in conjunction with the authorization decision. See Section 7.18 for a description of how the set of obligations to be returned by the PDP SHALL be determined. See section 7.2 about enforcement of obligations.

A conjunctive sequence of advice expressions which MUST evaluated into advice by the PDP. The corresponding advice provide supplementary information to the PEP in conjunction with the authorization decision. See Section 7.18 for a description of how the set of advice to be returned by the PDP SHALL be determined.

5.15 Element <PolicyDefaults>

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

The <PolicyDefaults> element is of DefaultsType complex type.

The <PolicyDefaults> element contains the following elements:

- XPathVersion [Optional]
  Default XPath version.

5.16 Element <CombinerParameters>

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

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

Note that none of the combining algorithms specified in XACML 3.0 is parameterized.
The `<CombinerParameters>` element is of `CombinerParametersType` complex type.

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

- `<CombinerParameter>` [Any Number]
  - A single parameter. See Section 5.17.

Support for the `<CombinerParameters>` element is optional.

### 5.17 Element `<CombinerParameter>`

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

```xml
<xs:element name="CombinerParameter" type="xacml:CombinerParameterType"/>
<xs:complexType name="CombinerParameterType">
  <xs:sequence>
    <xs:element ref="xacml:AttributeValue"/>
  </xs:sequence>
  <xs:attribute name="ParameterName" type="xs:string" use="required"/>
</xs:complexType>
```

The `<CombinerParameter>` element is of `CombinerParameterType` complex type.

The `<CombinerParameter>` element contains the following attributes:

- **ParameterName** [Required]
  - The identifier of the parameter.
- **AttributeValue** [Required]
  - The value of the parameter.

Support for the `<CombinerParameter>` element is optional.

### 5.18 Element `<RuleCombinerParameters>`

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

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

Note that none of the *rule-combining algorithms* specified in XACML 3.0 is parameterized.

```xml
<xs:element name="RuleCombinerParameters" type="xacml:RuleCombinerParametersType"/>
<xs:complexType name="RuleCombinerParametersType">
  <xs:complexContent>
    <xs:extension base="xacml:CombinerParametersType">
      <xs:attribute name="RuleIdRef" type="xs:string" use="required"/>
    </xs:extension>
  </xs:complexType>
```

The `<RuleCombinerParameters>` element contains the following attribute:
RuleIdRef [Required]
The identifier of the <Rule> contained in the policy.

Support for the <RuleCombinerParameters> element is optional, only if support for combiner parameters is not implemented.

5.19 Element <PolicyCombinerParameters>
The <PolicyCombinerParameters> element conveys parameters associated with a particular policy within a policy set for a policy-combining algorithm.

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

Note that none of the policy-combining algorithms specified in XACML 3.0 is parameterized.

```xml
<xsd:element name="PolicyCombinerParameters" type="xacml:PolicyCombinerParametersType"/>
<xsd:complexType name="PolicyCombinerParametersType">
    <xsd:complexContent>
        <xsd:extension base="xacml:CombinerParametersType">
            <xsd:attribute name="PolicyIdRef" type="xs:anyURI" use="required"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

The <PolicyCombinerParameters> element is of PolicyCombinerParametersType complex type.
The <PolicyCombinerParameters> element contains the following attribute:

PolicyIdRef [Required]
The identifier of a <Policy> or the value of a <PolicyIdReference> contained in the policy set.

Support for the <PolicyCombinerParameters> element is optional, only if support for combiner parameters is not implemented.

5.20 Element <PolicySetCombinerParameters>
The <PolicySetCombinerParameters> element conveys parameters associated with a particular policy set within a policy set for a policy-combining algorithm.

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

Note that none of the policy-combining algorithms specified in XACML 3.0 is parameterized.

```xml
<xsd:element name="PolicySetCombinerParameters" type="xacml:PolicySetCombinerParametersType"/>
<xsd:complexType name="PolicySetCombinerParametersType">
```

The `<PolicySetCombinerParameters>` element is of `PolicySetCombinerParametersType` complex type.

The `<PolicySetCombinerParameters>` element contains the following attribute:

**PolicySetIdRef** [Required]

The identifier of a `<PolicySet>` or the value of a `<PolicySetIdReference>` contained in the policy set.

Support for the `<PolicySetCombinerParameters>` element is optional, only if support for combiner parameters is not implemented.

### 5.21 Element `<Rule>`

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

A `<Rule>` element may be evaluated, in which case the evaluation procedure defined in Section 7.10 SHALL be used.

The `<Rule>` element is of `RuleType` complex type.

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

**RuleId** [Required]

A string identifying this rule.

**Effect** [Required]

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

**Description** [Optional]

A free-form description of the rule.

**Target** [Optional]

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 7.7 for details.

**Condition** [Optional]

A predicate that MUST be satisfied for the rule to be assigned its Effect value.
<ObligationExpressions> [Optional]

A conjunctive sequence of obligation expressions which MUST be evaluated into obligations by the PDP. The corresponding obligations MUST be fulfilled by the PEP in conjunction with the authorization decision. See Section 7.18 for a description of how the set of obligations to be returned by the PDP SHALL be determined. See Section 7.2 about enforcement of obligations.

<AdviceExpressions> [Optional]

A conjunctive sequence of advice expressions which MUST be evaluated into advice by the PDP. The corresponding advice provide supplementary information to the PEP in conjunction with the authorization decision. See Section 7.18 for a description of how the set of advice to be returned by the PDP SHALL be determined.

5.22 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 <ObligationExpression> and <AdviceExpression> elements.

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

5.23 Element <VariableDefinition>

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

```
<xs:element name="VariableDefinition" type="xacml:VariableDefinitionType"/>
<xs:complexType name="VariableDefinitionType">
  <xs:sequence>
    <xs:element ref="xacml:Expression"/>
  </xs:sequence>
  <xs:attribute name="VariableId" type="xs:string" use="required"/>
</xs:complexType>
```

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

- <Expression> [Required]
- VariableId [Required]
  - The name of the variable definition.

5.24 Element <VariableReference>

The <VariableReference> element is used to reference a value defined within the same encompassing <Policy> element. The <VariableReference> element SHALL refer to the
<VariableDefinition> element by string\text{identifier} equality on the value of their respective VariableId attributes. One and only one <VariableDefinition> MUST exist within the same encompassing <Policy> element to which the <VariableReference> refers. There MAY be zero or more <VariableReference> elements that refer to the same <VariableDefinition> element.

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

The <VariableReference> element has the following attribute:

VariableId [Required]

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

5.25 Element <Expression>

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

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

<Apply>, <AttributeSelector>, <AttributeValue>, <Function>, <VariableReference> and <AttributeDesignator>.

5.26 Element <Condition>

The <Condition> element is a Boolean function over attributes or functions of attributes.

The <Condition> contains one <Expression> element, with the restriction that the <Expression> return data-type MUST be "http://www.w3.org/2001/XMLSchema#boolean". Evaluation of the <Condition> element is described in Section 7.9.

5.27 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 the members of the <Expression> element substitution group. See Section 5.25.
The `<Apply>` element is of `ApplyType` complex type.

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

- **FunctionId [Required]**
  - The identifier of the function to be applied to the arguments. XACML-defined functions are described in Appendix A.3.

- **Description [Optional]**
  - A free-form description of the `<Apply>` element.

- **Expression [Optional]**
  - Arguments to the function, which may include other functions.

### 5.28 Element `<Function>`

The `<Function>` element SHALL be used to name a function as an argument to the function defined by the parent `<Apply>` element.

The `<Function>` element is of `FunctionType` complex type.

The `<Function>` element contains the following attribute:

- **FunctionId [Required]**
  - The identifier of the function.

### 5.29 Element `<AttributeDesignator>`

The `<AttributeDesignator>` element retrieves a bag of values for a named attribute from the request context. A named attribute SHALL be considered present if there is at least one attribute that matches the criteria set out below.

The `<AttributeDesignator>` element SHALL return a bag containing all the attribute values that are matched by the named attribute. In the event that no matching attribute is present in the context, the
MustBePresent attribute governs whether this element returns an empty bag or "Indeterminate". See Section 7.3.5.

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

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

A named attribute SHALL match an attribute if the values of their respective Category, AttributeId, DataType and Issuer attributes match. The attribute designator’s Category MUST match, by URI identifier equality, the Category of the <Attributes> element in which the attribute is present. The attribute designator’s AttributeId MUST match, by URI identifier equality, the AttributeId of the attribute. The attribute designator’s DataType MUST match, by URI identifier equality, the DataType of the same attribute.

If the Issuer attribute is present in the attribute designator, then it MUST match, using the “urn:oasis:names:tc:xacml:1.0:function:string-equal” function, 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:

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

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

- **DataType** [Required]
  - The bag returned by the <AttributeDesignator> element SHALL contain values of this data-type.

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

- **MustBePresent** [Required]
  - This attribute governs whether the element returns "Indeterminate" or an empty bag in the event the named attribute is absent from the request context. See Section 7.3.5. Also see Sections 7.19.2 and 7.19.3.
5.30 Element <AttributeSelector>

The <AttributeSelector> element produces a bag of unnamed and uncategorized attribute values. The values shall be constructed from the node(s) selected by applying the XPath expression given by the element's Path attribute to the XML content indicated by the element's Category attribute. Support for the <AttributeSelector> element is OPTIONAL.

See section 7.3.7 for details of <AttributeSelector> evaluation.

```
<xs:element name="AttributeSelector" type="xacml:AttributeSelectorType"
substitutionGroup="xacml:Expression"/>
<xs:complexType name="AttributeSelectorType">
  <xs:complexContent>
    <xs:extension base="xacml:ExpressionType">
      <xs:attribute name="Category" type="xs:anyURI"
        use="required"/>
      <xs:attribute name="ContextSelectorId" type="xs:anyURI"
        use="optional"/>
      <xs:attribute name="Path" type="xs:string"
        use="required"/>
      <xs:attribute name="DataType" type="xs:anyURI"
        use="required"/>
      <xs:attribute name="MustBePresent" type="xs:boolean"
        use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

The <AttributeSelector> element is of AttributeSelectorType complex type.

The <AttributeSelector> element has the following attributes:

Category [Required]

This attribute SHALL specify the attributes category of the <Content> element containing the XML from which nodes will be selected. It also indicates the attributes category containing the applicable ContextSelectorId attribute, if the element includes a ContextSelectorId xml attribute.

ContextSelectorId [Optional]

This attribute refers to the attribute (by its AttributeId) in the request context in the category given by the Category attribute. The referenced attribute MUST have data type urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression, and must select a single node in the <Content> element. The XPathCategory attribute of the referenced attribute MUST be equal to the Category attribute of the attribute selector.

Path [Required]

This attribute SHALL contain an XPath expression to be evaluated against the specified XML content. See Section 7.3.7 for details of the XPath evaluation during <AttributeSelector> processing.

DataType [Required]

The attribute specifies the datatype of the values returned from the evaluation of this <AttributeSelector> element.

MustBePresent [Required]

This attribute governs whether the element returns “Indeterminate” or an empty bag in the event the XPath expression selects no node. See Section 7.3.5. Also see Sections 7.19.2 and 7.19.3.
5.31 Element <AttributeValue>

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

```xml
<xsd:element name="AttributeValue" type="xacml:AttributeValueType" substitutionGroup="xacml:Expression"/>
<xsd:complexType name="AttributeValueType" mixed="true">
  <xsd:complexContent mixed="true">
    <xsd:extension base="xacml:ExpressionType">
      <xsd:sequence>
        <xsd:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="DataType" type="xs:anyURI" use="required"/>
      <xsd:anyAttribute namespace="##any" processContents="lax"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

The <AttributeValue> element is of AttributeValueType complex type.

The <AttributeValue> element has the following attributes:

- **DataType** [Required]
  - The data-type of the attribute value.

5.32 Element <Obligations>

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

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

The <Obligations> element is of ObligationsType complexType.

The <Obligations> element contains the following element:

- **Obligation** [One to Many]
  - A sequence of obligations. See Section 5.34.

5.33 Element <AssociatedAdvice>

The <AssociatedAdvice> element SHALL contain a set of <Advice> elements.

```xml
<xsd:element name="AssociatedAdvice" type="xacml:AssociatedAdviceType"/>
<xsd:complexType name="AssociatedAdviceType">
  <xsd:sequence>
    <xsd:element ref="xacml:Advice" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

The <AssociatedAdvice> element is of AssociatedAdviceType complexType.

The <AssociatedAdvice> element contains the following element:

- **Advice** [One to Many]
  - A sequence of advice. See Section 5.35.
5.34 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**.

```xml
<xs:element name="Obligation" type="xacml:ObligationType"/>
<xs:complexType name="ObligationType">
  <xs:sequence>
    <xs:element ref="xacml:AttributeAssignment" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
</xs:complexType>
```

The <Obligation> element is of **ObligationType** complexType. See Section 7.18 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**.

- **<AttributeAssignment>** [Optional]
  - **Obligation** arguments assignment. The values of the **obligation** arguments SHALL be interpreted by the **PEP**.

5.35 Element <Advice>

The <Advice> element SHALL contain an identifier for the **advice** and a set of **attributes** that form arguments of the supplemental information defined by the **advice**.

```xml
<xs:element name="Advice" type="xacml:AdviceType"/>
<xs:complexType name="AdviceType">
  <xs:sequence>
    <xs:element ref="xacml:AttributeAssignment" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="AdviceId" type="xs:anyURI" use="required"/>
</xs:complexType>
```

The <Advice> element is of **AdviceType** complexType. See Section 7.18 for a description of how the set of **advice** to be returned by the **PDP** is determined.

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

- **AdviceId** [Required]
  - **Advice** identifier. The value of the **advice** identifier MAY be interpreted by the **PEP**.

- **<AttributeAssignment>** [Optional]
  - **Advice** arguments assignment. The values of the **advice** arguments MAY be interpreted by the **PEP**.

5.36 Element <AttributeAssignment>

The <AttributeAssignment> element is used for including arguments in **obligation** and **advice** expressions. It SHALL contain an **AttributeId** and the corresponding **attribute** value, by extending the **AttributeValue** type definition. The <AttributeAssignment> element MAY be used in any way that is consistent with the schema syntax, which is a sequence of **<xs:any>** elements. The value specified SHALL be understood by the **PEP**, but it is not further specified by XACML. See Section 7.18. Section 4.2.4.3 provides a number of examples of arguments included in **obligation** expressions.
The `<AttributeAssignment>` element is of `AttributeAssignmentType` complex type.
The `<AttributeAssignment>` element contains the following attributes:

**AttributeId** [Required]
The attribute identifier.

**Category** [Optional]
An optional category of the `attribute`. If this attribute is missing, the `attribute` has no category.
The PEP SHALL interpret the significance and meaning of any Category attribute. Non-normative note: an expected use of the category is to disambiguate `attribute` which are relayed from the request.

**Issuer** [Optional]
An optional issuer of the `attribute`. If this attribute is missing, the `attribute` has no issuer. The PEP SHALL interpret the significance and meaning of any Issuer attribute. Non-normative note: an expected use of the issuer is to disambiguate `attribute` which are relayed from the request.

### 5.37 Element `<ObligationExpressions>`
The `<ObligationExpressions>` element SHALL contain a set of `<ObligationExpression>` elements.

The `<ObligationExpressions>` element is of `ObligationExpressionsType` complexType.
The `<ObligationExpressions>` element contains the following element:

**<ObligationExpression>** [One to Many]
A sequence of obligations expressions. See Section 5.39.

### 5.38 Element `<AdviceExpressions>`
The `<AdviceExpressions>` element SHALL contain a set of `<AdviceExpression>` elements.

The `<AdviceExpressions>` element is of `AdviceExpressionsType` complexType.
The `<AdviceExpressions>` element contains the following element:

```
<AdviceExpression> [One to Many]
```

A sequence of advice expressions. See Section 5.40.

### 5.39 Element `<ObligationExpression>`

The `<ObligationExpression>` element evaluates to an obligation and SHALL contain an identifier for an obligation and a set of expressions that form arguments of the action defined by the obligation. The FulfillOn attribute SHALL indicate the effect for which this obligation must be fulfilled by the PEP.

```
<x:element name="ObligationExpression"
    type="xacml:ObligationExpressionType"/>
<x:complexType name="ObligationExpressionType">
    <xs:sequence>
        <xs:element ref="xacml:AttributeAssignmentExpression" minOccurs="0"
            maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
    <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required"/>
</xs:complexType>
```

The `<ObligationExpression>` element is of ObligationExpressionType complexType. See Section 7.18 for a description of how the set of obligations to be returned by the PDP is determined.

The `<ObligationExpression>` 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 must be fulfilled by the PEP.

- **<AttributeAssignmentExpression> [Optional]**
  - Obligation arguments in the form of expressions. The expressions SHALL be evaluated by the PDP to constant `<AttributeValue>` elements or bags, which shall be the attribute assignments in the `<Obligation>` returned to the PEP. If an `<AttributeAssignmentExpression>` evaluates to an atomic attribute value, then there MUST be one resulting `<AttributeAssignment>` which MUST contain this single attribute value. If the `<AttributeAssignmentExpression>` evaluates to a bag, then there MUST be a resulting `<AttributeAssignment>` for each of the values in the bag. If the bag is empty, there shall be no `<AttributeAssignment>` from this `<AttributeAssignmentExpression>`. The values of the obligation arguments SHALL be interpreted by the PEP.

### 5.40 Element `<AdviceExpression>`

The `<AdviceExpression>` element evaluates to an advice and SHALL contain an identifier for an advice and a set of expressions that form arguments of the supplemental information defined by the advice. The AppliesTo attribute SHALL indicate the effect for which this advice must be provided to the PEP.

```
<x:element name="AdviceExpression" type="xacml:AdviceExpressionType"/>
<x:complexType name="AdviceExpressionType">
    <xs:sequence>
        <xs:element ref="xacml:AttributeAssignmentExpression" minOccurs="0"
            maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="AdviceId" type="xs:anyURI" use="required"/>
    <xs:attribute name="AppliesTo" type="xacml:EffectType" use="required"/>
</xs:complexType>
```
The `<AdviceExpression>` element is of `AdviceExpressionType` complexType. See Section 7.18 for a description of how the set of advice to be returned by the PDP is determined.

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

- **AdviceId** [Required]
  - The *Advice* identifier. The value of the *advice* identifier MAY be interpreted by the PEP.

- **AppliesTo** [Required]
  - The *effect* for which this *advice* must be provided to the PEP.

- **<AttributeAssignmentExpression>** [Optional]
  - *Advice* arguments in the form of expressions. The expressions SHALL be evaluated by the PDP to constant `<AttributeValue>` elements or bags, which shall be the attribute assignments in the `<Advice>` returned to the PEP. If an `<AttributeAssignmentExpression>` evaluates to an atomic *attribute* value, then there MUST be one resulting `<AttributeAssignment>` which MUST contain this single *attribute* value. If the `<AttributeAssignmentExpression>` evaluates to a *bag*, then there MUST be a resulting `<AttributeAssignment>` for each of the values in the *bag*. If the *bag* is empty, there shall be no `<AttributeAssignment>` from this `<AttributeAssignmentExpression>`. The values of the *advice* arguments MAY be interpreted by the PEP.

### 5.41 Element `<AttributeAssignmentExpression>`

The `<AttributeAssignmentExpression>` element is used for including arguments in *obligations* and *advice*. It SHALL contain an *AttributeId* and an expression which SHALL be evaluated into the corresponding *attribute* value. The value specified SHALL be understood by the PEP, but it is not further specified by XACML. See Section 7.18. Section 4.2.4.3 provides a number of examples of arguments included in *obligations*.

```xml
<xs:element name="AttributeAssignmentExpression" type="xacml:AttributeAssignmentExpressionType"/>
<xs:complexType name="AttributeAssignmentExpressionType">
  <xs:sequence>
    <xs:element ref="xacml:Expression"/>
  </xs:sequence>
  <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
  <xs:attribute name="Category" type="xs:anyURI" use="optional"/>
  <xs:attribute name="Issuer" type="xs:string" use="optional"/>
</xs:complexType>
```

The `<AttributeAssignmentExpression>` element is of `AttributeAssignmentExpressionType` complex type.

The `<AttributeAssignmentExpression>` element contains the following attributes:

- **<Expression>** [Required]
  - The expression which evaluates to a constant *attribute* value or a bag of zero or more attribute values. See section 5.25.

- **AttributeId** [Required]
  - The *attribute* identifier. The value of the AttributeId attribute in the resulting `<AttributeAssignment>` element MUST be equal to this value.

- **Category** [Optional]
  - An optional category of the *attribute*. If this attribute is missing, the *attribute* has no category.

  The value of the Category attribute in the resulting `<AttributeAssignment>` element MUST be equal to this value.
Issuer [Optional]

An optional issuer of the attribute. If this attribute is missing, the attribute has no issuer. The value of the Issuer attribute in the resulting <AttributeAssignment> element MUST be equal to this value.

5.42 Element <Request>

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

The <Request> element contains <Attributes> elements. There may be multiple <Attributes> elements with the same Category attribute if the PDP implements the multiple decision profile, see [Multi]. Under other conditions, it is a syntax error if there are multiple <Attributes> elements with the same Category (see Section 7.19.2 for error codes).

```
<xs:element name="Request" type="xacml:RequestType"/>
<xs:complexType name="RequestType">
  <xs:sequence>
    <xs:element ref="xacml:RequestDefaults" minOccurs="0"/>
    <xs:element ref="xacml:Attributes" maxOccurs="unbounded"/>
    <xs:element ref="xacml:MultiRequests" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="ReturnPolicyIdList" type="xs:boolean" use="required"/>
  <xs:attribute name="CombinedDecision" type="xs:boolean" use="required"/>
</xs:complexType>
```

The <Request> element is of RequestType complex type.

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

ReturnPolicyIdList [Required]

This attribute is used to request that the PDP return a list of all fully applicable policies and policy sets which were used in the decision as a part of the decision response.

CombinedDecision [Required]

This attribute is used to request that the PDP combines multiple decisions into a single decision. The use of this attribute is specified in [Multi]. If the PDP does not implement the relevant functionality in [Multi], then the PDP must return an Indeterminate with a status code of urn:oasis:names:tc:xacml:1.0:status:processing-error if it receives a request with this attribute set to "true".

<RequestDefaults> [Optional]

Contains default values for the request, such as XPath version. See section 5.43.

<Attributes> [One to Many]

Specifies information about attributes of the request context by listing a sequence of <Attribute> elements associated with an attribute category. One or more <Attributes> elements are allowed. Different <Attributes> elements with different categories are used to represent information about the subject, resource, action, environment or other categories of the access request.

<MultiRequests> [Optional]

Lists multiple request contexts by references to the <Attributes> elements. Implementation of this element is optional. The semantics of this element is defined in [Multi]. If the implementation does not implement this element, it MUST return an Indeterminate result if it encounters this element. See section 5.50.
5.43 Element <RequestDefaults>

The <RequestDefaults> element SHALL specify default values that apply to the <Request> element.

```xml
<xs:element name="RequestDefaults" type="xacml:RequestDefaultsType"/>
<xs:complexType name="RequestDefaultsType">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="xacml:XPathVersion"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

The <RequestDefaults> element is of RequestDefaultsType complex type.

The <RequestDefaults> element contains the following elements:

- **XPathVersion** [Optional]

  Default XPath version for XPath expressions occurring in the request.

5.44 Element <Attributes>

The <Attributes> element specifies attributes of a subject, resource, action, environment or another category by listing a sequence of <Attribute> elements associated with the category.

```xml
<xs:element name="Attributes" type="xacml:AttributesType"/>
<xs:complexType name="AttributesType">
  <xs:sequence>
    <xs:element ref="xacml:Content" minOccurs="0" maxOccurs="unbounded"/>
    <xs:attribute name="Category" type="xs:anyURI" use="required"/>
    <xs:attribute ref="xml:id" use="optional"/>
  </xs:sequence>
  <xs:complexType name="SubjectType">
    <xs:element name="Content" type="xacml:ContentType"/>
  </xs:complexType>
</xs:complexType>
```

The <Attributes> element is of AttributesType complex type.

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

- **Category** [Required]

  This attribute indicates which attribute category the contained attributes belong to. The Category attribute is used to differentiate between attributes of subject, resource, action, environment or other categories.

- **xml:id** [Optional]

  This attribute provides a unique identifier for this <Attributes> element. See [XMLid] It is primarily intended to be referenced in multiple requests. See [Multi].

- **Content** [Optional]

  Specifies additional sources of attributes in free form XML document format which can be referenced using <AttributeSelector> elements.

- **Attribute** [Any Number]

  A sequence of attributes that apply to the category of the request.

5.45 Element <Content>

The <Content> element is a notional placeholder for additional attributes, typically the content of the resource.
The `<Content>` element is of `ContentType` complex type.
The `<Content>` element has exactly one arbitrary type child element.

### 5.46 Element `<Attribute>`

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

The `<Attribute>` element is of `AttributeType` complex type.
The `<Attribute>` element contains the following attributes and elements:

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

- **Issuer** [Optional]
  The `Attribute` issuer. For example, 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.

- **IncludeInResult** [Default: false]
  Whether to include this `attribute` in the result. This is useful to correlate requests with their responses in case of multiple requests.

- **<AttributeValue>** [One to Many]
  One or more `attribute` values. Each `attribute` value MAY have contents that are empty, occur once or occur multiple times.

### 5.47 Element `<Response>`

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>` element 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 by some implementations, in particular those that support the XACML Profile for Requests for Multiple Resources [Multi]. Support for multiple results is OPTIONAL.
The `<Response>` element is of `ResponseType` complex type.

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

- `<Result>` [One to Many]

  An authorization decision result. See Section 5.48.

### 5.48 Element `<Result>`

The `<Result>` element represents an authorization decision result. It MAY include a set of obligations that MUST be fulfilled by the PEP. If the PEP does not understand or cannot fulfill an obligation, then the action of the PEP is determined by its bias, see section 7.1. It MAY include a set of advice with supplemental information which MAY be safely ignored by the PEP.

```
<x:complexType name="ResultType">
    <x:sequence>
        <x:element ref="xacml:Decision" />
        <x:element ref="xacml:Status" minOccurs="0" />
        <x:element ref="xacml:Obligations" minOccurs="0" />
        <x:element ref="xacml:AssociatedAdvice" minOccurs="0" />
        <x:element ref="xacml:Attributes" minOccurs="0" maxOccurs="unbounded" />
        <x:element ref="xacml:PolicyIdentifierList" minOccurs="0" />
    </x:sequence>
</x:complexType>
```

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

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

- `<Decision>` [Required]

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

- `<Status>` [Optional]

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

- `<Obligations>` [Optional]

  A list of obligations that MUST be fulfilled by the PEP. If the PEP does not understand or cannot fulfill an obligation, then the action of the PEP is determined by its bias, see section 7.2. See Section 7.18 for a description of how the set of obligations to be returned by the PDP is determined.

- `<AssociatedAdvice>` [Optional]

  A list of advice that provide supplemental information to the PEP. If the PEP does not understand an advice, the PEP may safely ignore the advice. See Section 7.18 for a description of how the set of advice to be returned by the PDP is determined.

- `<Attributes>` [Optional]

  A list of attributes that were part of the request. The choice of which attributes are included here is made with the IncludeInResult attribute of the `<Attribute>` elements of the request. See section 5.46.
<PolicyIdentifierList> [Optional]

If the ReturnPolicyIdList attribute in the <Request> is true (see section 5.42), a PDP that
implements this optional feature MUST return a list of all policies which were found to be fully
applicable. That is, all policies where both the <Target> matched and the <Condition>
evaluated to true, whether or not the <Effect> was the same or different from the <Decision>.

5.49 Element <PolicyIdentifierList>

The <PolicyIdentifierList> element contains a list of policy and policy set identifiers of policies
which have been applicable to a request. The list is unordered.

The <PolicyIdentifierList> element contains the following elements.

<PolicyIdReference> [Any number]

The identifier and version of a policy which was applicable to the request. See section 5.11. The
<PolicyIdReference> element MUST use the Version attribute to specify the version and
MUST NOT use the LatestVersion or EarliestVersion attributes.

<PolicySetIdReference> [Any number]

The identifier and version of a policy set which was applicable to the request. See section 5.10. The
<PolicySetIdReference> element MUST use the Version attribute to specify the
version and MUST NOT use the LatestVersion or EarliestVersion attributes.

5.50 Element <MultiRequests>

The <MultiRequests> element contains a list of requests by reference to <Attributes> elements in
the enclosing <Request> element. The semantics of this element are defined in [Multi]. Support for this
element is optional. If an implementation does not support this element, but receives it, the
implementation MUST generate an "Indeterminate" response.

The <MultiRequests> element contains the following elements.

<RequestReference> [one to many]

Defines a request instance by reference to <Attributes> elements in the enclosing
<Request> element. See section 5.51.

5.51 Element <RequestReference>

The <RequestReference> element defines an instance of a request in terms of references to
<Attributes> elements. The semantics of this element are defined in [Multi]. Support for this element
is optional.
The `<RequestReference>` element contains the following elements.

- `<AttributesReference>` [one to many]
  
  A reference to an `<Attributes>` element in the enclosing `<Request>` element. See section 5.52.

### 5.52 Element `<AttributesReference>`

The `<AttributesReference>` element makes a reference to an `<Attributes>` element. The meaning of this element is defined in [Multi]. Support for this element is optional.

The `<AttributesReference>` element contains the following attributes.

- **ReferenceId** [required]
  
  A reference to the `xml:id` attribute of an `<Attributes>` element in the enclosing `<Request>` element.

### 5.53 Element `<Decision>`

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

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.

### 5.54 Element `<Status>`

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

"NotApplicable": the **PDP** does not have any **policy** that applies to this decision request.
<xs:element ref="xacml:StatusCode"/>
<xs:element ref="xacml:StatusMessage" minOccurs="0"/>
<xs:element ref="xacml:StatusDetail" minOccurs="0"/>
</xs:sequence>
</xs: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.

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

<xs:element name="StatusCode" type="xacml:StatusCodeType"/>
<xs:complexType name="StatusCodeType">
<xs:sequence>
<xs:element ref="xacml: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.8 for a list of values.

<StatusCode> [Any Number]
Minor status code. This status code qualifies its parent status code.

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

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

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

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

<xs:element name="StatusDetail" type="xacml:StatusDetailType"/>
<xs:complexType name="StatusDetailType">
<xs:sequence>
<xs: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 `<MissingAttributeDetail>` elements.

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

- `urn:oasis:names:tc:xacml:1.0:status:processing-error`
  - A PDP MUST NOT return a `<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.

### 5.58 Element `<MissingAttributeDetail>`

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

```xml
<xs:element name="MissingAttributeDetail"
    type="xacml:MissingAttributeDetailType"/>
<xs:complexType name="MissingAttributeDetailType">
    <xs:sequence>
        <xs:element ref="xacml:AttributeValue" minOccurs="0"
            maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Category" type="xs:anyURI" use="required"/>
    <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:complexType>
```

The `<MissingAttributeDetail>` element is of MissingAttributeDetailType complex type.

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

- `<AttributeValue>` [Optional]
  - The required value of the missing attribute.
- `Category` [Required]
  - The category identifier of the missing attribute.
- `AttributeId` [Required]
  - The identifier of the missing attribute.
- `DataType` [Required]
  - The data-type of the missing attribute.
- `Issuer` [Optional]
  - This attribute, if supplied, SHALL specify the required Issuer of the missing attribute.
If the **PDP** includes `<AttributeValue>` elements in the `<MissingAttributeDetail>` 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**.
6 XPath 2.0 definitions

The XPath 2.0 specification leaves a number of aspects of behavior implementation defined. This section defines how XPath 2.0 SHALL behave when hosted in XACML.

http://www.w3.org/TR/2007/REC-xpath20-20070713/#id-impl-defined-items defines the following items:

1. The version of Unicode that is used to construct expressions.
   XACML leaves this implementation defined. It is RECOMMENDED that the latest version is used.

2. The statically-known collations.
   XACML leaves this implementation defined.

3. The implicit timezone.
   XACML defined the implicit time zone as UTC.

4. The circumstances in which warnings are raised, and the ways in which warnings are handled.
   XACML leaves this implementation defined.

5. The method by which errors are reported to the external processing environment.
   An XPath error causes an XACML Indeterminate value in the element where the XPath error occurs. The StatusCode value SHALL be “urn:oasis:names:tc:xacml:1.0:status:processing-error”. Implementations MAY provide additional details about the error in the response or by some other means.

6. Whether the implementation is based on the rules of XML 1.0 or 1.1.
   XACML is based on XML 1.0.

7. Whether the implementation supports the namespace axis.
   XACML leaves this implementation defined. It is RECOMMENDED that users of XACML do not make use of the namespace axis.

8. Any static typing extensions supported by the implementation, if the Static Typing Feature is supported.
   XACML leaves this implementation defined.

http://www.w3.org/TR/2007/REC-xpath-datamodel-20070713/#implementation-defined defines the following items:

1. Support for additional user-defined or implementation-defined types is implementation-defined.
   It is RECOMMENDED that implementations of XACML do not define any additional types and it is RECOMMENDED that users of XACML do not make user of any additional types.

2. Some typed values in the data model are undefined. Attempting to access an undefined property is always an error. Behavior in these cases is implementation-defined and the host language is responsible for determining the result.
   An XPath error causes an XACML Indeterminate value in the element where the XPath error occurs. The StatusCode value SHALL be “urn:oasis:names:tc:xacml:1.0:status:processing-error”. Implementations MAY provide additional details about the error in the response or by some other means.

http://www.w3.org/TR/2007/REC-xpath-functions-20070713/#impl-def defines the following items:

1. The destination of the trace output is implementation-defined.
   XACML leaves this implementation defined.

2. For xs:integer operations, implementations that support limited-precision integer operations must either raise an error [err:FOAR0002] or provide an implementation-defined mechanism that allows users to choose between raising an error and returning a result that is modulo the largest representable integer value.
   XACML leaves this implementation defined. If an implementation chooses to raise an error, the
3. For xs:decimal values the number of digits of precision returned by the numeric operators is implementation-defined.

XACML leaves this implementation defined.

4. If the number of digits in the result of a numeric operation exceeds the number of digits that the implementation supports, the result is truncated or rounded in an implementation-defined manner.

XACML leaves this implementation defined.

5. It is implementation-defined which version of Unicode is supported.

XACML leaves this implementation defined. It is RECOMMENDED that the latest version is used.

6. For fn:normalize-unicode, conforming implementations must support normalization form "NFC" and may support normalization forms "NFD", "NFKC", "NFKD", "FULLY-NORMALIZED". They may also support other normalization forms with implementation-defined semantics.

XACML leaves this implementation defined.

7. The ability to decompose strings into collation units suitable for substring matching is an implementation-defined property of a collation.

XACML leaves this implementation defined.

8. All minimally conforming processors must support year values with a minimum of 4 digits (i.e., YYYY) and a minimum fractional second precision of 1 millisecond or three digits (i.e., s.sss). However, conforming processors may set larger implementation-defined limits on the maximum number of digits they support in these two situations.

XACML leaves this implementation defined, and it is RECOMMENDED that users of XACML do not expect greater limits and precision.

9. The result of casting a string to xs:decimal, when the resulting value is not too large or too small but nevertheless has too many decimal digits to be accurately represented, is implementation-defined.

XACML leaves this implementation defined.

10. Various aspects of the processing provided by fn:doc are implementation-defined.

Implementations may provide external configuration options that allow any aspect of the processing to be controlled by the user.

XACML leaves this implementation defined.

11. The manner in which implementations provide options to weaken the stable characteristic of fn:collection and fn:doc are implementation-defined.

XACML leaves this implementation defined.
7 Functional requirements

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

Note that in each case an implementation is conformant as long as it produces the same result as is specified here, regardless of how and in what order the implementation behaves internally.

7.1 Unicode issues

7.1.1 Normalization

In Unicode, some equivalent characters can be represented by more than one different Unicode character sequence. See [CMF]. The process of converting Unicode strings into equivalent character sequences is called "normalization" [UAX15]. Some operations, such as string comparison, are sensitive to normalization. An operation is normalization-sensitive if its output(s) are different depending on the state of normalization of the input(s); if the output(s) are textual, they are deemed different only if they would remain different were they to be normalized.

For more information on normalization see [CM].

An XACML implementation MUST behave as if each normalization-sensitive operation normalizes input strings into Unicode Normalization Form C ("NFC"). An implementation MAY use some other form of internal processing (such as using a non-Unicode, "legacy" character encoding) as long as the externally visible results are identical to this specification.

7.1.2 Version of Unicode

The version of Unicode used by XACML is implementation defined. It is RECOMMENDED that the latest version is used. Also note security issues in section 9.3.

7.2 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 as described in one of the following sub-sections

In any case any advice in the decision may be safely ignored by the PEP.

7.2.1 Base PEP

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

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

If the decision is “Not Applicable”, then the PEP’s behavior is undefined.

If the decision is “Indeterminate”, then the PEP’s behavior is undefined.

7.2.2 Deny-biased PEP

If the decision is "Permit", then the PEP SHALL permit access. If obligations accompany the decision, then the PEP SHALL permit access only if it understands and it can and will discharge those obligations.
All other decisions SHALL result in the denial of access.

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

7.2.3 Permit-biased PEP

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

All other decisions SHALL result in the permission of access.

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

7.3 Attribute evaluation

Attributes are represented in the request context by the context handler, regardless of whether or not they appeared in the original decision request, and are referred to in the policy by attribute designators and attribute selectors. A named attribute is the term used for the criteria that the specific attribute designators use to refer to particular attributes in the <Attributes> elements of the request context.

7.3.1 Structured attributes

<AddressValue> elements MAY contain an instance of a structured XML data-type, for example <ds:KeyInfo>. XACML 3.0 supports several ways for comparing the contents of such elements.

1. In some cases, such elements MAY be compared using one of the XACML functions, such as "string-regexp-match", described below. This requires that the element be given the data-type "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:

```
<AddressValue DataTypes="http://www.w3.org/2001/XMLSchema#string">
  &lt;ds:KeyName&gt;jhibbert-key&lt;/ds:KeyName&gt;
</AddressValue>
```

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

2. The structured attribute MAY be made available in the <Content> element of the appropriate attribute category and an <AttributeSelector> element MAY be used to select the contents 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. The structured attribute MAY be made available in the <Content> element of the appropriate attribute category and an <AttributeSelector> element MAY be used to select 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 A.3.15. This method requires support by the PDP for the optional XPath expressions and XPath functions features.

See also Section 7.3.

7.3.2 Attribute bags

XACML defines implicit collections of its data-types. XACML refers to a collection of values that are of a single data-type as a bag. Bags of data-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 free form XML. The result of an XPath expression is termed a node-set, which contains all the nodes from the XML content 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 data-type.

### 7.3.3 Multivalued attributes

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

### 7.3.4 Attribute Matching

A named attribute includes specific criteria with which to match `attributes` in the `context`. An attribute specifies a Category, AttributeId and `DataType`, and a named attribute also specifies the Issuer. A named attribute SHALL match an attribute if the values of their respective Category, AttributeId, `DataType` and optional Issuer attributes match. The Category of the named attribute MUST match, by `URI` equality, the Category of the corresponding `context` attribute. The AttributeId of the named attribute MUST match, by `URI` equality, the AttributeId of the corresponding `context` attribute. The `DataType` of the named attribute MUST match, by `URI` equality, the `DataType` of the corresponding `context` attribute. If Issuer is supplied in the named attribute, then it MUST match, using the `urn:oasis:names:tc:xacml:1.0:function:string-equality` function, the Issuer of the corresponding `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 corresponding `context` attribute. 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`.

### 7.3.5 Attribute Retrieval

The PDP SHALL request the values of `attributes` in the request `context` from the `context handler`. The `context handler` MAY also add `attributes` to the request `context` without the PDP requesting them. 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 by whatever means it deems appropriate., including by retrieving them from one or more `Policy Information Points`. 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 7.17. However, a `PDP` MAY choose not to return such information for security reasons.

Regardless of any dynamic modifications of the request `context` during policy evaluation, the PDP SHALL behave as if each `bag` of `attribute` values is fully populated in the `context` before it is first tested, and is thereafter immutable during evaluation. (That is, every subsequent test of that `attribute` shall use the same `bag` of values that was initially tested.)
7.3.6 Environment Attributes

Standard environment attributes are listed in Section B.7. 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. In the case of date and time attributes, the supplied value SHALL have the semantics of the "date and time that apply to the decision request".

7.3.7 AttributeSelector evaluation

An <AttributeSelector> element will be evaluated according to the following processing model.

1. Construct an XML data structure suitable for xpath processing from the <Content> element in the attributes category given by the Category attribute. The data structure shall be constructed so that the document node of this structure contains a single document element which corresponds to the single child element of the <Content> element. The constructed data structure shall be equivalent to one that would result from parsing a stand-alone XML document consisting of the contents of the <Content> element (including any comment and processing-instruction markup). Namespace declarations which are not "visibly utilized", as defined by [exc-c14n], MAY not be present and MUST NOT be utilized by the XPath expression in step 3. The data structure must meet the requirements of the applicable xpath version.

2. Select a context node for xpath processing from this data structure. If there is a ContextSelectorId attribute, the context node shall be the node selected by applying the XPath expression given in the attribute value of the designated attribute (in the attributes category given by the <AttributeSelector> Category attribute). It shall be an error if this evaluation returns no node or more than one node, in which case the return value MUST be an "Indeterminate" with a status code "urn:oasis:names:tc:xacml:1.0:status:syntax-error". If there is no ContextSelectorId, the document node of the data structure shall be the context node.

3. Evaluate the XPath expression given in the Path attribute against the xml data structure, using the context node selected in the previous step. It shall be an error if this evaluation returns anything other than a sequence of nodes (possibly empty), in which case the <AttributeSelector> MUST return "Indeterminate" with a status code "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

4. If the data type is a primitive data type, convert the text value of each selected node to the desired data type, as specified in the DataType attribute. Each value shall be constructed using the appropriate constructor function from [XF] Section 5 listed below, corresponding to the specified data type.

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

If the DataType is not one of the primitive types listed above, then the return values shall be
constructed from the nodeset in a manner specified by the particular DataType extension specification. If the data type extension does not specify an appropriate constructor function, then the <AttributeSelector> MUST return "Indeterminate" with a status code "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

If an error occurs when converting the values returned by the XPath expression to the specified DataType, then the result of the <AttributeSelector> MUST be "Indeterminate", with a status code "urn:oasis:names:tc:xacml:1.0:status:processing-error".

7.4 Expression evaluation

XACML specifies expressions in terms of the elements listed below, 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> elements SHALL agree with the respective argument types of the function that is named by the FunctionId attribute. The resultant type of the <Apply> 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 these elements to be in the substitution group of the <Expression> element:

- <xacml:AttributeValue>
- <xacml:AttributeDesignator>
- <xacml:AttributeSelector>
- <xacml:Apply>
- <xacml:Function>
- <xacml:VariableReference>

7.5 Arithmetic evaluation

IEEE 754 [IEEE754] 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)

7.6 Match evaluation

The attribute matching element <Match> appears in the <Target> element of rules, policies and policy sets. This element represents a Boolean expression over attributes of the request context. A matching element contains a MatchId attribute that specifies the function to be used in performing the match evaluation, an <AttributeValue> 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 takes 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, as explained below. The DataType of the <AttributeValue>
SHALL match the data-type of the first argument expected by the MatchId function. The DataType of the <AttributeDesignator> or <AttributeSelector> element SHALL match the data-type of the second argument expected by the MatchId function.

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 two single base types as its inputs. 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 <AttributeValue> 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, if all function applications evaluate to "False", then the result of the entire expression SHALL be "False".

It is also 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:

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

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:

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

### 7.7 Target evaluation

An empty target matches any request. Otherwise the target value SHALL be "Match" if all the AnyOf specified in the target match values in the request context. Otherwise, if any one of the AnyOf specified in the target is "No Match", then the target SHALL be "No Match". Otherwise, the target SHALL be "Indeterminate". The target match table is shown in Table 1.

<table>
<thead>
<tr>
<th>&lt;AnyOf&gt; values</th>
<th>Target value</th>
</tr>
</thead>
</table>

19 April 2012
Standards Track Work Product Copyright © OASIS Open 2012. All Rights Reserved.
Table 1 Target match table

<table>
<thead>
<tr>
<th>Match Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>All “Match”</td>
<td>“Match”</td>
</tr>
<tr>
<td>At least one “No Match”</td>
<td>“No Match”</td>
</tr>
<tr>
<td>Otherwise</td>
<td>“Indeterminate”</td>
</tr>
</tbody>
</table>

The AnyOf SHALL match values in the request **context** if at least one of their **<AllOf>** elements matches a value in the request **context**. The AnyOf table is shown in Table 2.

<table>
<thead>
<tr>
<th>&lt;AllOf&gt; values</th>
<th>&lt;AnyOf&gt; Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>At least one “Match”</td>
<td>“Match”</td>
</tr>
<tr>
<td>None matches and at least one “Indeterminate”</td>
<td>“Indeterminate”</td>
</tr>
<tr>
<td>All “No match”</td>
<td>“No match”</td>
</tr>
</tbody>
</table>

Table 2 AnyOf match table

An AllOf SHALL match a value in the request **context** if the value of all its **<Match>** elements is “True”. The AllOf table is shown in Table 3.

<table>
<thead>
<tr>
<th>&lt;Match&gt; values</th>
<th>&lt;AllOf&gt; Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>All “True”</td>
<td>“Match”</td>
</tr>
<tr>
<td>No “False” and at least one “Indeterminate”</td>
<td>“Indeterminate”</td>
</tr>
<tr>
<td>At least one “False”</td>
<td>“No match”</td>
</tr>
</tbody>
</table>

Table 3 AllOf match table

7.8 VariableReference Evaluation

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

In any place where a **<VariableReference>** occurs, it has the effect as if the text of the **<Expression>** element defined in the **<VariableDefinition>** element replaces the **<VariableReference>** element. Any evaluation scheme that preserves this semantic is acceptable.

For instance, the expression in the **<VariableDefinition>** element may be evaluated to a particular value and cached for multiple references without consequence. (i.e. the value of an **<Expression>** element remains the same for the entire **policy** evaluation.) This characteristic is one of the benefits of XACML being a declarative language.

A variable reference containing circular references is invalid. The PDP MUST detect circular references either at policy loading time or during runtime evaluation. If the PDP detects a circular reference during runtime the variable reference evaluates to “Indeterminate” with status code urn:oasis:names:tc:xacml:1.0:status:processing-error.

7.9 Condition evaluation

The **condition** value SHALL be “True” if the **<Condition>** element is absent, or if it evaluates to “True”. Its value SHALL be “False” if the **<Condition>** element evaluates to “False”. The **condition** value
SHALL be “Indeterminate”, if the expression contained in the <Condition> element evaluates to “Indeterminate.”

7.10 Extended Indeterminate

Some combining algorithms are defined in terms of an extended set of “Indeterminate” values. The extended set associated with the “Indeterminate” contains the potential effect values which could have occurred if there would not have been an error causing the “Indeterminate”. The possible extended set “Indeterminate” values are:

- “Indeterminate(D)”: an “Indeterminate” from a policy or rule which could have evaluated to “Deny”, but not “Permit”.
- “Indeterminate(P)”: an “Indeterminate” from a policy or rule which could have evaluated to “Permit”, but not “Deny”.
- “Indeterminate(DP)”: an “Indeterminate” from a policy or rule which could have evaluated to “Deny” or “Permit”.

The combining algorithms which are defined in terms of the extended “Indeterminate” make use of the additional information to allow for better treatment of errors in the algorithms.

The final decision returned by a PDP cannot be an extended Indeterminate. Any such decision at the top level policy or policy set is returned as a plain Indeterminate in the response from the PDP.

The tables in the following four sections define how extended “Indeterminate” values are produced during Rule, Policy and PolicySet evaluation.

7.11 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 4.

<table>
<thead>
<tr>
<th>Target</th>
<th>Condition</th>
<th>Rule Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Match” or no target</td>
<td>“True”</td>
<td>Effect</td>
</tr>
<tr>
<td>“Match” or no target</td>
<td>“False”</td>
<td>“NotApplicable”</td>
</tr>
<tr>
<td>“Match” or no target</td>
<td>“Indeterminate”</td>
<td>“Indeterminate(P)” if the Effect is Permit, or “Indeterminate(D)” if the Effect is Deny</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(P)” if the Effect is Permit, or “Indeterminate(D)” if the Effect is Deny</td>
</tr>
</tbody>
</table>

Table 4 Rule truth table.

7.12 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, according to the specified rule-combining algorithm, rules.

The policy truth table is shown in Table 5.
### 7.13 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, and, according to the specified policy-combining algorithm, policies and policy sets, the policy set truth table is shown in Table 6.

<table>
<thead>
<tr>
<th>Target</th>
<th>Rule values</th>
<th>Policy Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Match&quot;</td>
<td>Don't care</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>See Table 7</td>
<td>See Table 7</td>
</tr>
</tbody>
</table>

Table 5 Policy truth table

Note that none of the rule-combining algorithms defined by XACML 3.0 take parameters. However, non-standard combining algorithms MAY take parameters. In such a case, the values of these parameters associated with the rules, MUST be taken into account when evaluating the policy. The parameters and their types should be defined in the specification of the combining algorithm. If the implementation supports combiner parameters and if combiner parameters are present in a policy, then the parameter values MUST be supplied to the combining algorithm implementation.

### 7.14 Policy and Policy set value for Indeterminate Target

If the target of a policy or policy set evaluates to "Indeterminate", the value of the policy or policy set as a whole is determined by the value of the combining algorithm according to Table 7.

<table>
<thead>
<tr>
<th>Combining algorithm Value</th>
<th>Policy set or policy Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;NotApplicable&quot;</td>
<td>&quot;NotApplicable&quot;</td>
</tr>
<tr>
<td>&quot;Permit&quot;</td>
<td>&quot;Indeterminate(P)&quot;</td>
</tr>
<tr>
<td>&quot;Deny&quot;</td>
<td>&quot;Indeterminate(D)&quot;</td>
</tr>
<tr>
<td>&quot;Indeterminate&quot;</td>
<td>&quot;Indeterminate(DP)&quot;</td>
</tr>
<tr>
<td>&quot;Indeterminate(DP)&quot;</td>
<td>&quot;Indeterminate(DP)&quot;</td>
</tr>
<tr>
<td>&quot;Indeterminate(P)&quot;</td>
<td>&quot;Indeterminate(P)&quot;</td>
</tr>
</tbody>
</table>
7.15 PolicySetIdReference and PolicyIdReference evaluation

A policy set id reference or a policy id reference is evaluated by resolving the reference and evaluating the referenced policy set or policy.

If resolving the reference fails, the reference evaluates to "Indeterminate" with status code urn:oasis:names:tc:xacml:1.0:status:processing-error.

A policy set id reference or a policy id reference containing circular references is invalid. The PDP MUST detect circular references either at policy loading time or during runtime evaluation. If the PDP detects a circular reference during runtime the reference evaluates to "Indeterminate" with status code urn:oasis:names:tc:xacml:1.0:status:processing-error.

7.16 Hierarchical resources

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

7.17 Authorization decision

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

The PDP MUST evaluate the policy set as specified in Sections 5 and 7. 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 SHALL be returned.

7.18 Obligations and advice

A rule, policy, or policy set may contain one or more obligation or advice expressions. When such a rule, policy, or policy set is evaluated, the obligation or advice expression SHALL be evaluated to an obligation or advice respectively, which SHALL be passed up to the next level of evaluation (the enclosing or referencing policy, policy set, or authorization decision) only if the result of the rule, policy, or policy set being evaluated matches the value of the FulfillOn attribute of the obligation or the AppliesTo attribute of the advice. If any of the attribute assignment expressions in an obligation or advice expression with a matching FulfillOn or AppliesTo attribute evaluates to "Indeterminate", then the whole rule, policy, or policy set SHALL be "Indeterminate". If the FulfillOn or AppliesTo attribute does not match the result of the combining algorithm or the rule evaluation, then any indeterminate in an obligation or advice expression has no effect.

As a consequence of this procedure, no obligations or advice SHALL be returned to the PEP if the rule, 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 rule, 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 rules, policy sets and policies, each of which returns "Permit" or "Deny", then the set of obligations and advice returned by the PDP to the PEP will include only the obligations and advice associated with those paths where the result at each level of evaluation is the same as the result being returned by the PDP. In situations where any lack of determinism is unacceptable, a deterministic combining algorithm, such as ordered-deny-overrides, should be used.
7.19 Exception handling

XACML specifies behavior for the **PDP** in the following situations.

### 7.19.1 Unsupported functionality

If the **PDP** attempts to evaluate a **policy set** or **policy** that contains an optional element type or function 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 function.

### 7.19.2 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".

### 7.19.3 Missing attributes

The absence of matching attributes in the request context for any of the attribute designators attribute or selectors that are found in the **policy** will result in an enclosing `<AllOf>` element to return a value of "Indeterminate", if the designator or selector has the MustBePresent XML attribute set to true, as described in Sections 5.29 and 5.30 and may result in a `<Decision>` element containing the "Indeterminate" value. If, in this case a status code is supplied, then the value "urn:oasis:names:tc:xacml:1.0:status:missing-attribute" SHALL be used, to indicate that more information is needed in order for a definitive decision to be rendered. In this case, the `<Status>` element MAY list the names and data-types of any attributes that are needed by the **PDP** to refine its decision (see Section 5.58). 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:status: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:status:missing-attribute", it MUST NOT list the names and data-types of any attribute 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.

### 7.20 Identifier equality

XACML makes use of URLs and strings as identifiers. When such identifiers are compared for equality, the comparison MUST be done so that the identifiers are equal if they have the same length and the characters in the two identifiers are equal codepoint by codepoint.

The following is a list of the identifiers which MUST use this definition of equality.

The content of the element `<XPathVersion>`.

The XML attribute `Value` in the element `<StatusCode>`.
The XML attributes `Category`, `AttributeId`, `DataType` and `Issuer` in the element `<MissingAttributeDetail>`.  

The XML attribute `Category` in the element `<Attributes>`.  

The XML attributes `AttributeId` and `Issuer` in the element `<Attribute>`.  

The XML attribute `ObligationId` in the element `<Obligation>`.  

The XML attribute `AdviceId` in the element `<Advice>`.  

The XML attributes `AttributeId` and `Category` in the element `<AttributeAssignment>`.  

The XML attribute `ObligationId` in the element `<ObligationExpression>`.  

The XML attribute `AdviceId` in the element `<AdviceExpression>`.  

The XML attributes `AttributeId`, `Category` and `Issuer` in the element `<AttributeAssignmentExpression>`.  

The XML attributes `PolicySetId` and `PolicyCombiningAlgId` in the element `<PolicySet>`.  

The XML attribute `ParameterName` in the element `<CombinerParameter>`.  

The XML attribute `RuleIdRef` in the element `<RuleCombinerParameters>`.  

The XML attribute `PolicyIdRef` in the element `<PolicyCombinerParameters>`.  

The XML attribute `PolicySetIdRef` in the element `<PolicySetCombinerParameters>`.  

The anyURI in the content of the complex type `IdReferenceType`.  

The XML attributes `PolicyId` and `RuleCombiningAlgId` in the element `<Policy>`.  

The XML attribute `RuleId` in the element `<Rule>`.  

The XML attribute `MatchId` in the element `<Match>`.  

The XML attribute `VariableId` in the element `<VariableDefinition>`.  

The XML attribute `VariableId` in the element `<VariableReference>`.  

The XML attributes `Category`, `ContextSelectorId` and `DataType` in the element `<AttributeSelector>`.  

The XML attributes `Category`, `AttributeId`, `DataType` and `Issuer` in the element `<AttributeDesignator>`.  

The XML attribute `DataType` in the element `<AttributeValue>`.  

The XML attribute `FunctionId` in the element `<Function>`.  

The XML attribute `FunctionId` in the element `<Apply>`.  

It is RECOMMENDED that extensions to XACML use the same definition of identifier equality for similar identifiers.  

It is RECOMMENDED that extensions which define identifiers do not define identifiers which could be easily misinterpreted by people as being subject to other kind of processing, such as URL character escaping, before matching.
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.

Category, AttributeId, DataType, FunctionId, MatchId, ObligationId, AdviceId, PolicyCombiningAlgId, RuleCombiningAlgId, StatusCode, SubjectCategory.

See Section 5 for definitions of these attribute types.

8.2 Structured attributes

<AttributeValue> elements MAY contain an instance of a structured XML data-type. Section 7.3.1 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 message 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 to protect the 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/or large fines in the case of medical or financial data.

Unauthorized disclosure is addressed by confidentiality safeguards.

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 safeguards.

Note that encryption of the message does not mitigate a replay attack since the message is simply 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 message sequence integrity safeguards 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...
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 message sequence integrity safeguards 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. A message integrity safeguard can prevent a successful message modification attack.

### 9.1.6 NotApplicable results

A result of "NotApplicable" means that the PDP could not locate a policy whose target matched the information in the decision request. In general, it is highly recommended that a "Deny" effect policy be used, so that when a PDP would have returned "NotApplicable", a result of "Deny" is returned instead.

In some security models, however, such as those found in many web servers, an authorization decision 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 be closely aligned with the data syntax used by the applications that will be submitting the decision request. A failure to match will result in "NotApplicable" and be treated as "Permit". So an unintended failure to match may allow unintended access.

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 may be 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. In a more open environment, where decision requests may be received from applications that 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. Note, however, that according to Section 7.2, a PEP must deny access unless it receives an explicit "Permit" authorization decision.

### 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 violations, 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 over 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 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 violations in two common cases: 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 result. 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
administrates all policies, but when administration is distributed, as XACML allows, this type of situation
must be explicitly guarded against.

9.1.8 Denial of service

A denial of service attack is one in which the adversary overloads an XACML actor with excessive
computations or network traffic such that legitimate users cannot access the services provided by the
actor.
The urn:oasis:names:tc:xacml:3.0:function:access-permitted function may lead to hard to predict behavior
in the PDP. It is possible that the function is invoked during the recursive invocations of the PDP such that
loops are formed. Such loops may in some cases lead to large numbers of requests to be generated
before the PDP can detect the loop and abort evaluation. Such loops could cause a denial of service at
the PDP, either because of a malicious policy or because of a mistake in a policy.

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
either 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 considerations related to transmitting or exchanging XACML <Policy> elements are outside the scope of the XACML standard. While it is 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.

If a PDP is provided with policies from distinct sources which might not be fully trusted, as in the use of the administration profile [XACMLAdmin], there is a concern that someone could intentionally publish a policy with an id which collides with another policy. This could cause policy references that point to the wrong policy, and may cause other unintended consequences in an implementation which is predicated upon having unique policy identifiers.

If this issue is a concern it is RECOMMENDED that distinct policy issuers or sources are assigned distinct namespaces for policy identifiers. One method is to make sure that the policy identifier begins with a string which has been assigned to the particular policy issuer or source. The remainder of the policy identifier is an issuer-specific unique part. For instance, Alice from Example Inc. could be assigned the policy identifiers which begin with http://example.com/xacml/policyId/alice/. The PDP or another trusted component can then verify that the authenticated source of the policy is Alice at Example Inc, or otherwise reject the policy. Anyone else will be unable to publish policies with identifiers which collide with the policies of Alice.

9.2.6 Trust model

Discussions of authentication, integrity and confidentiality safeguards 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 models 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 data) but are responsible for verifying it themselves.
- 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 requirements 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.
9.3 Unicode security issues

There are many security considerations related to use of Unicode. An XACML implementation SHOULD follow the advice given in the relevant version of [UTR36].

9.4 Identifier equality

Section 7.20 defines the identifier equality operation for XACML. This definition of equality does not do any kind of canonicalization or escaping of characters. The identifiers defined in the XACML specification have been selected to not include any ambiguity regarding these aspects. It is RECOMMENDED that identifiers defined by extensions also do not introduce any identifiers which might be mistaken for being subject to processing, like for instance URL character encoding using “%".
10 Conformance

10.1 Introduction

The XACML specification addresses the following aspect of conformance:

The XACML specification defines a number of functions, etc. that have somewhat special applications, 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 to XACML v3.0. A set of test cases has been created to assist in this process. These test cases can be located from the OASIS XACML TC Web page. The site hosting the test cases contains a full description of the test cases and how to execute them.

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

The implementation MUST follow sections 5, 6, 7, Appendix A, Appendix B and Appendix C where they apply to implemented items in the following tables.

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:Advice</td>
<td>M</td>
</tr>
<tr>
<td>xacml:AdviceExpression</td>
<td>M</td>
</tr>
<tr>
<td>xacml:AdviceExpressions</td>
<td>M</td>
</tr>
<tr>
<td>xacml:AllOf</td>
<td>M</td>
</tr>
<tr>
<td>xacml:AnyOf</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Apply</td>
<td>M</td>
</tr>
<tr>
<td>xacml:AssociatedAdvice</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Attribute</td>
<td>M</td>
</tr>
<tr>
<td>xacml:AttributeAssignment</td>
<td>M</td>
</tr>
<tr>
<td>xacml:AttributeAssignmentExpression</td>
<td>M</td>
</tr>
<tr>
<td>xacml:AttributeDesignator</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Attributes</td>
<td>M</td>
</tr>
<tr>
<td>xacml:AttributeSelector</td>
<td>O</td>
</tr>
<tr>
<td>xacml:AttributesReference</td>
<td>O</td>
</tr>
<tr>
<td>xacml:AttributeValue</td>
<td>M</td>
</tr>
<tr>
<td>xacml:CombinerParameter</td>
<td>O</td>
</tr>
<tr>
<td>xacml:CombinerParameters</td>
<td>O</td>
</tr>
<tr>
<td>xacml:Condition</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Content</td>
<td>O</td>
</tr>
<tr>
<td>xacml:Decision</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Description</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Expression</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Function</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Match</td>
<td>M</td>
</tr>
<tr>
<td>xacml:MissingAttributeDetail</td>
<td>M</td>
</tr>
<tr>
<td>xacml:MultiRequests</td>
<td>O</td>
</tr>
<tr>
<td>xacml:Obligation</td>
<td>M</td>
</tr>
<tr>
<td>xacml:ObligationExpression</td>
<td>M</td>
</tr>
<tr>
<td>xacml:ObligationExpressions</td>
<td>M</td>
</tr>
<tr>
<td>xacml:Obligations</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>
<th>M/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn:oasis:names:tc:xacml:3.0</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:2.0</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:2.0:conformance-test</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:2.0:context</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:2.0:example</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:2.0:function</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:2.0:policy</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:resource</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:action</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:environment</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:status</td>
<td></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>overrides</td>
<td></td>
</tr>
<tr>
<td>overrides</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-</td>
<td>M</td>
</tr>
<tr>
<td>overrides</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-</td>
<td>M</td>
</tr>
<tr>
<td>applicable</td>
<td></td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-</td>
<td>M</td>
</tr>
<tr>
<td>one-</td>
<td></td>
</tr>
</tbody>
</table>
### 10.2.4 Status Codes

Implementation support for the `<StatusCode>` 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>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:status:ok</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:status:processing-error</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:status:syntax-error</td>
<td>M</td>
</tr>
</tbody>
</table>

### 10.2.5 Attributes

The implementation MUST support the *attributes* associated with the following 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 *context handler*. 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 uses 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>0</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address</td>
<td>0</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:authentication-method</td>
<td>0</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:authentication-time</td>
<td>0</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:key-info</td>
<td>0</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:request-time</td>
<td>0</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:session-start-time</td>
<td>0</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:subject-id</td>
<td>0</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier</td>
<td>0</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>0</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/2001/XMLSchema#duration">http://www.w3.org/2001/XMLSchema#duration</a></td>
<td>M</td>
</tr>
<tr>
<td><a href="http://www.w3.org/2001/XMLSchema#duration-type">http://www.w3.org/2001/XMLSchema#duration-type</a></td>
<td>M</td>
</tr>
<tr>
<td><a href="http://www.w3.org/2001/XMLSchema#URI">http://www.w3.org/2001/XMLSchema#URI</a></td>
<td>M</td>
</tr>
<tr>
<td><a href="http://www.w3.org/2001/XMLSchema#hexBinary">http://www.w3.org/2001/XMLSchema#hexBinary</a></td>
<td>M</td>
</tr>
<tr>
<td><a href="http://www.w3.org/2001/XMLSchema#base64Binary">http://www.w3.org/2001/XMLSchema#base64Binary</a></td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:data-type:x500Name</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:2.0:data-type:ipAddress</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:2.0:data-type:dnspName</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: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:3.0:function:dayTimeDuration-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-equal</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case</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>
</tbody>
</table>
### 10.2.9 Identifiers planned for future deprecation

These identifiers are associated with previous versions of XACML and newer alternatives exist in XACML 3.0. They are planned to be deprecated at some unspecified point in the future. It is RECOMMENDED that these identifiers not be used in new policies and requests.

The implementation MUST properly process those features 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:rfc822Name-at-least-one-member-of</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:rfc822Name-union</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:rfc822Name-subset</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:rfc822Name-set-equals</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:3.0:function:access-permitted</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:xpath-node-count</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:xpath-node-match</td>
<td>O</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:2.0:function:uri-string-concatenate</td>
<td>M</td>
</tr>
<tr>
<td><a href="http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration">http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration</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>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-dayTimeDuration</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-yearMonthDuration</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dateTime- additive-yearMonthDuration</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-intersection</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-at-least-one-member-of</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-union</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-subset</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-set-equals</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-intersection</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-at-least-one-member-of</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-union</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-subset</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-set-equals</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-one-and-only</td>
<td>M</td>
</tr>
<tr>
<td>urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-bag-size</td>
<td>M</td>
</tr>
<tr>
<td>Function</td>
<td>M</td>
</tr>
<tr>
<td>----------------------------------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td><code>urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-is-in</code></td>
<td>M</td>
</tr>
<tr>
<td><code>urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-bag</code></td>
<td>M</td>
</tr>
<tr>
<td><code>urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-one-and-only</code></td>
<td>M</td>
</tr>
<tr>
<td><code>urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-bag-size</code></td>
<td>M</td>
</tr>
<tr>
<td><code>urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-is-in</code></td>
<td>M</td>
</tr>
<tr>
<td><code>urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-bag</code></td>
<td>M</td>
</tr>
<tr>
<td><code>urn:oasis:names:tc:xacml:1.0:function:any-of</code></td>
<td>M</td>
</tr>
<tr>
<td><code>urn:oasis:names:tc:xacml:1.0:function:all-of</code></td>
<td>M</td>
</tr>
<tr>
<td><code>urn:oasis:names:tc:xacml:1.0:function:any-of-any</code></td>
<td>M</td>
</tr>
<tr>
<td><code>urn:oasis:names:tc:xacml:1.0:function:map</code></td>
<td>M</td>
</tr>
</tbody>
</table>
Appendix A. Data-types and functions (normative)

A.1 Introduction

This section specifies the data-types and functions used in XACML to create predicates for conditions 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. It describes the primitive data-types and bags. The standard functions are named and their operational semantics are described.

A.2 Data-types

Although XML instances represent all data-types as strings, an XACML PDP must operate on 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/2001/XMLSchema#dayTimeDuration
- http://www.w3.org/2001/XMLSchema#yearMonthDuration
- urn:oasis:names:tc:xacml:1.0:data-type:x500Name
- urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name
- urn:oasis:names:tc:xacml:2.0:data-type:ipAddress
- urn:oasis:names:tc:xacml:2.0:data-type:dnSName
- urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression

For the sake of improved interoperability, it is RECOMMENDED that all time references be in UTC time.

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

XACML defines four data-types representing identifiers for subjects or resources; these are:

- "urn:oasis:names:tc:xacml:1.0:data-type:x500Name",
- "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
- "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" and
- "urn:oasis:names:tc:xacml:2.0:data-type:dnSName"

These types appear in several standard applications, such as TLS/SSL and electronic mail.

X.500 directory name
The "urn:oasis:names:tc:xacml:1.0:data-type:x500Name" primitive type represents 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".

RFC 822 name

The "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" primitive type represents an electronic mail address. The valid syntax for such a name is described in IETF RFC 2821, Section 4.1.2, Command Argument Syntax, under the term "Mailbox".

IP address

The "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" primitive type represents an IPv4 or IPv6 network address, with optional mask and optional port or port range. The syntax SHALL be:

ipAddress = address [ "/" mask ] [ ":" [ portrange ] ]

For an IPv4 address, the address and mask are formatted in accordance with the syntax for a "host" in IETF RFC 2396 "Uniform Resource Identifiers (URI): Generic Syntax", section 3.2.

For an IPv6 address, the address and mask are formatted in accordance with the syntax for an "ipv6reference" in IETF RFC 2732 "Format for Literal IPv6 Addresses in URL's". (Note that an IPv6 address or mask, in this syntax, is enclosed in literal "[" "]" brackets.)

DNS name

The "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" primitive type represents a Domain Name Service (DNS) host name, with optional port or port range. The syntax SHALL be:

dnsName = hostname [ ":" portrange ]

The hostname is formatted in accordance with IETF RFC 2396 "Uniform Resource Identifiers (URI): Generic Syntax", section 3.2, except that a wildcard "*" may be used in the left-most component of the hostname to indicate "any subdomain" under the domain specified to its right.

For both the "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" and "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" data-types, the port or port range syntax SHALL be:

portrange = portnumber | "-"portnumber | portnumber"-"

where "portnumber" is a decimal port number. If the port number is of the form "-x", where "x" is a port number, then the range is all ports numbered "x" and below. If the port number is of the form "x-", then the range is all ports numbered "x" and above. [This syntax is taken from the Java SocketPermission.]

XPath expression

The "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression" primitive type represents an XPath expression over the XML in a <Content> element. The syntax is defined by the XPath W3C recommendation. The content of this data type also includes the context in which namespaces prefixes in the expression are resolved, which distinguishes it from a plain string and the XACML attribute category of the <Content> element to which it applies. When the value is encoded in an <AttributeValue> element, the namespace context is given by the <AttributeValue> element and an XML attribute called XPathCategory gives the category of the <Content> element where the expression applies.

The XPath expression MUST be evaluated in a context which is equivalent of a stand alone XML document with the only child of the <Content> element as the document element. Namespace declarations which are not "visibly utilized", as defined by [exc-c14n], MAY not be present and MUST NOT be utilized by the XPath expression. The context node of the XPath expression is the document node of this stand alone document.
A.3 Functions

XACML specifies the following functions. Unless otherwise specified, if an argument of one of these functions were to evaluate to "Indeterminate", then the function SHALL be set to "Indeterminate".

Note that in each case an implementation is conformant as long as it produces the same result as is specified here, regardless of how and in what order the implementation behaves internally.

A.3.1 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.

- urn:oasis:names:tc:xacml:1.0:function:string-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". 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.
  Otherwise, it SHALL return "False". The comparison SHALL use Unicode codepoint collation, as
  defined for the identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by [XF].

- urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case
  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". The result SHALL be "True" if and only if the
  two strings are equal as defined by urn:oasis:names:tc:xacml:1.0:function:string-equal after they
  have both been converted to lower case with urn:oasis:names:tc:xacml:1.0:function:string-
  normalize-to-lower-case.

- urn:oasis:names:tc:xacml:1.0:function:boolean-equal
  This function SHALL take two arguments of data-type
  "http://www.w3.org/2001/XMLSchema#boolean" and SHALL return an
  "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and only if
  the arguments are equal. Otherwise, it SHALL return "False".

- urn:oasis:names:tc:xacml:1.0:function: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". The function SHALL return "True" if and only if
  the two arguments represent the same number.

- urn:oasis:names:tc:xacml:1.0:function: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 [IEEE754].

- urn:oasis:names:tc:xacml:1.0:function: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 10.4.9.

- urn:oasis:names:tc:xacml:1.0:function: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 10.4.12.
• urn:oasis:names:tc:xacml:1.0:function: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 10.4.6.

• urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-equal
  
  This function SHALL take two arguments of data-type
  "http://www.w3.org/2001/XMLSchema#dayTimeDuration" and SHALL return an
  "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
  according to the "op:duration-equal" function [XF] Section 10.4.5. Note that the lexical
  representation of each argument MUST be converted to a value expressed in fractional seconds
  [XF] Section 10.3.2.

• urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-equal
  
  This function SHALL take two arguments of data-type
  "http://www.w3.org/2001/XMLSchema#yearMonthDuration" and SHALL return an
  "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
  according to the "op:duration-equal" function [XF] Section 10.4.5. Note that the lexical
  representation of each argument MUST be converted to a value expressed in fractional seconds
  [XF] Section 10.3.2.

• urn:oasis:names:tc:xacml:1.0:function: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". The function SHALL convert the arguments to
  strings with urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI and return "True" if and
  only if the values of the two arguments are equal on a codepoint-by-codepoint basis.

• urn:oasis:names:tc:xacml:1.0:function: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. Otherwise,
  it SHALL return "False". 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".

• urn:oasis:names:tc:xacml:1.0:function: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". It
  SHALL return "True" if and only if the two arguments are equal. Otherwise, it SHALL return
  "False". 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. Perform the following operations:
  
  1. Normalize the domain-part of each argument to lower case
  2. Compare the expressions by applying the function
This function SHALL take two arguments of data-type "http://www.w3.org/2001/XMLSchema#hexBinary" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "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" function. Otherwise, it SHALL return "False". The conversion from the string representation to an octet sequence SHALL be as specified in [XS] Section 3.2.15.

This function SHALL take two arguments of data-type "http://www.w3.org/2001/XMLSchema#base64Binary" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "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" function. Otherwise, it SHALL return "False". The conversion from the string representation to an octet sequence SHALL be as specified in [XS] Section 3.2.16.

A.3.2 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" and "multiply" functions MAY take more than two arguments. Each function evaluation operating on doubles SHALL proceed as specified by their logical counterparts in IEEE 754 [IEEE754]. For all of these functions, if any argument is "Indeterminate", then the function SHALL evaluate to "Indeterminate". In the case of the divide functions, if the divisor is zero, then the function SHALL evaluate to "Indeterminate".

This function MUST accept two or more arguments.

This function MUST accept two or more arguments.

The result is the second argument subtracted from the first argument.

The result is the second argument subtracted from the first argument.

This function MUST accept two or more arguments.

This function MUST accept two or more arguments.

The result is the first argument divided by the second argument.

The result is the first argument divided by the second argument.

The result is remainder of the first argument divided by the second argument.

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 a value of the data-type "http://www.w3.org/2001/XMLSchema#double".

This function MUST accept two or more arguments.
A.3.3 String conversion functions

The following functions convert between values of the data-type "http://www.w3.org/2001/XMLSchema#string" primitive types.

- `urn:oasis:names:tc:xacml:1.0:function: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 white space characters. The whitespace characters are defined in the metasymbol S (Production 3) of [XML].

- `urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case`
  This function SHALL take one argument of data-type "http://www.w3.org/2001/XMLSchema#string" and SHALL normalize the value by converting each upper case character to its lower case equivalent. Case mapping shall be done as specified for the fn:lower-case function in [XF] with no tailoring for particular languages or environments.

A.3.4 Numeric data-type conversion functions

The following functions convert between the data-type "http://www.w3.org/2001/XMLSchema#integer" and "http://www.w3.org/2001/XMLSchema#double" primitive types.

- `urn:oasis:names:tc:xacml:1.0:function: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".

- `urn:oasis:names:tc:xacml:1.0:function: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" with the same numeric value. If the integer argument is outside the range which can be represented by a double, the result SHALL be Indeterminate, with the status code "urn:oasis:names:tc:xacml:1.0:status:processing-error".

A.3.5 Logical functions

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

- `urn:oasis:names:tc:xacml:1.0:function:or`
  This function SHALL return "False" if it has no arguments and SHALL return "True" if at least 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.

- `urn:oasis:names:tc:xacml:1.0:function: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.

- `urn:oasis:names:tc:xacml:1.0:function:n-of`
  The first argument to this function SHALL be of data-type http://www.w3.org/2001/XMLSchema#integer. The remaining arguments SHALL be of data-type
http://www.w3.org/2001/XMLSchema#boolean. The first argument specifies the minimum 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, and 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.

- urn:oasis:names:tc:xacml:1.0:function:not
  This function SHALL take one argument of data-type "http://www.w3.org/2001/XMLSchema#boolean". 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".

Note: When evaluating and, or, or n-of, it MAY NOT be necessary to attempt a full evaluation of each argument in order to determine whether the evaluation of the argument would result in "Indeterminate". Analysis of the argument regarding the availability of its 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.

A.3.6 Numeric comparison functions

These functions form a minimal set for comparing two numbers, yielding a Boolean result. For doubles they SHALL comply with the rules governed by IEEE 754 [IEEE754].

- urn:oasis:names:tc:xacml:1.0:function:integer-greater-than
- urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal
- urn:oasis:names:tc:xacml:1.0:function:integer-less-than
- urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal
- urn:oasis:names:tc:xacml:1.0:function:double-greater-than
- urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal
- urn:oasis:names:tc:xacml:1.0:function:double-less-than
- urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal

A.3.7 Date and time arithmetic functions

These functions perform arithmetic operations with date and time.

- urn:oasis:names:tc:xacml:3.0:function:dateTime-add-dayTimeDuration
  This function SHALL take two arguments, the first SHALL be of data-type "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be of data-type "http://www.w3.org/2001/XMLSchema#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.

- urn:oasis:names:tc:xacml:3.0:function:dateTime-add-yearMonthDuration
  This function SHALL take two arguments, the first SHALL be a "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a "http://www.w3.org/2001/XMLSchema#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.
A.3.8 Non-numeric comparison functions

These functions perform comparison operations on two arguments of non-numerical types.

- **urn:oasis:names:tc:xacml:3.0:function: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 first
  argument is lexicographically strictly greater than the second argument. Otherwise, it SHALL
  return "False". The comparison SHALL use Unicode codepoint collation, as defined for the
  identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by [XF].

- **urn:oasis:names:tc:xacml:3.0:function: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 "True" if and only if the first
  argument is lexicographically greater than or equal to the second argument. Otherwise, it SHALL
  return "False". The comparison SHALL use Unicode codepoint collation, as defined for the
  identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by [XF].
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 the first
argument is lexigraphically strictly less than the second argument. Otherwise, it SHALL return
"False". The comparison SHALL use Unicode codepoint collation, as defined for the identifier
http://www.w3.org/2005/xpath-functions/collation/codepoint by [XF].

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 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. Otherwise, it SHALL return
"False". Note: it is illegal to compare a time that includes a time-zone value with one that does
not. In such cases, the time-in-range function should be used.

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 and only 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. Otherwise, it SHALL return
"False". Note: it is illegal to compare a time that includes a time-zone value with one that does
not. In such cases, the time-in-range function should be used.

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 and only 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. Otherwise, it SHALL return
"False". Note: it is illegal to compare a time that includes a time-zone value with one that does
not. In such cases, the time-in-range function should be used.

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 and only 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. Otherwise, it SHALL return
"False". Note: it is illegal to compare a time that includes a time-zone value with one that does
not. In such cases, the time-in-range function should be used.
“False”. Regardless of its value, the third argument SHALL be interpreted as a time that is equal to, or later than by less than twenty-four hours, the second argument. If no time zone is provided for the first argument, it SHALL use the default time zone at the context handler. If no time zone is provided for the second or third arguments, then they SHALL use the time zone from the first argument.

- urn:oasis:names:tc:xacml:1.0:function: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 and only if the first argument is greater than the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" by [XS] part 2, section 3.2.7. Otherwise, it SHALL return "False". Note: if a dateTime value does not include a time-zone value, then an implicit time-zone value SHALL be assigned, as described in [XS].

- urn:oasis:names:tc:xacml:1.0:function:dateTime-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 and only 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" by [XS] part 2, section 3.2.7. Otherwise, it SHALL return "False". Note: if a dateTime value does not include a time-zone value, then an implicit time-zone value SHALL be assigned, as described in [XS].

- urn:oasis:names:tc:xacml:1.0:function:dateTime-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 and only if the first argument is less than the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" by [XS] part 2, section 3.2.7. Otherwise, it SHALL return "False". Note: if a dateTime value does not include a time-zone value, then an implicit time-zone value SHALL be assigned, as described in [XS].

- urn:oasis:names:tc:xacml:1.0:function:dateTime-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 and only 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" by [XS] part 2, section 3.2.7. Otherwise, it SHALL return "False". Note: if a dateTime value does not include a time-zone value, then an implicit time-zone value SHALL be assigned, as described in [XS].

- urn:oasis:names:tc:xacml:1.0:function:date-greater-than

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 return "True" if and only if the first argument is greater than the second argument according to the order relation specified for "http://www.w3.org/2001/XMLSchema#date" by [XS] part 2, section 3.2.9. Otherwise, it SHALL return "False". Note: if a date value does not include a time-zone value, then an implicit time-zone value SHALL be assigned, as described in [XS].

- urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-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 return "True" if and only 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#date" by [XS] part 2, section 3.2.9.
Otherwise, it SHALL return “False”. Note: if a date value does not include a time-zone value, then an implicit time-zone value SHALL be assigned, as described in [XS].

• urn:oasis:names:tc:xacml:1.0:function:date-less-than

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 return "True" if and only if the first
argument is less than the second argument according to the order relation specified for
"http://www.w3.org/2001/XMLSchema#date" by [XS] part 2, section 3.2.9. Otherwise, it SHALL
return “False”. Note: if a date value does not include a time-zone value, then an implicit time-
zone value SHALL be assigned, as described in [XS].

• urn:oasis:names:tc:xacml:3.0:function:date-less-than-or-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 return "True" if and only 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#date" by [XS] part 2, section 3.2.9. Otherwise, it
SHALL return “False”. Note: if a date value does not include a time-zone value, then an implicit
time-zone value SHALL be assigned, as described in [XS].

A.3.9 String functions

The following functions operate on strings and convert to and from other data types.

• urn:oasis:names:tc:xacml:2.0:function:string-concatenate

This function SHALL take two or more arguments of data-type
"http://www.w3.org/2001/XMLSchema#string" and SHALL return a
"http://www.w3.org/2001/XMLSchema#string". The result SHALL be the concatenation, in order,
of the arguments.

• urn:oasis:names:tc:xacml:3.0:function:boolean-from-string

This function SHALL take one argument of data-type
"http://www.w3.org/2001/XMLSchema#string", and SHALL return an
"http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be the string converted to a
boolean. If the argument is not a valid lexical representation of a boolean, then the result SHALL
be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.

• urn:oasis:names:tc:xacml:3.0:function:string-from-boolean

This function SHALL take one argument of data-type
"http://www.w3.org/2001/XMLSchema#boolean", and SHALL return an
"http://www.w3.org/2001/XMLSchema#string". The result SHALL be the boolean converted to a
string in the canonical form specified in [XS].

• urn:oasis:names:tc:xacml:3.0:function:integer-from-string

This function SHALL take one argument of data-type
"http://www.w3.org/2001/XMLSchema#string", and SHALL return an
"http://www.w3.org/2001/XMLSchema#integer". The result SHALL be the string converted to an
integer. If the argument is not a valid lexical representation of an integer, then the result SHALL
be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.

• urn:oasis:names:tc:xacml:3.0:function:string-from-integer

This function SHALL take one argument of data-type
"http://www.w3.org/2001/XMLSchema#integer", and SHALL return an
"http://www.w3.org/2001/XMLSchema#string". The result SHALL be the integer converted to a
string in the canonical form specified in [XS].

• urn:oasis:names:tc:xacml:3.0:function:double-from-string
This function SHALL take one argument of data-type "http://www.w3.org/2001/XMLSchema#string", and SHALL return an double. The result SHALL be the string converted to a double. If the argument is not a valid lexical representation of a double, then the result SHALL be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.

- urn:oasis:names:tc:xacml:3.0:function:string-from-double
  This function SHALL take one argument of data-type "http://www.w3.org/2001/XMLSchema#double", and SHALL return an string in the canonical form specified in [XS].

- urn:oasis:names:tc:xacml:3.0:function:time-from-string
  This function SHALL take one argument of data-type "http://www.w3.org/2001/XMLSchema#time", and SHALL return an string in the canonical form specified in [XS].

- urn:oasis:names:tc:xacml:3.0:function:date-from-string
  This function SHALL take one argument of data-type "http://www.w3.org/2001/XMLSchema#date", and SHALL return an string in the canonical form specified in [XS].

- urn:oasis:names:tc:xacml:3.0:function:dateTime-from-string
  This function SHALL take one argument of data-type "http://www.w3.org/2001/XMLSchema#dateTime", and SHALL return an string in the canonical form specified in [XS].

- urn:oasis:names:tc:xacml:3.0:function:anyURI-from-string
  This function SHALL take one argument of data-type "http://www.w3.org/2001/XMLSchema#anyURI", and SHALL return an string in the canonical form specified in [XS].
• urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI
  This function SHALL take one argument of data-type
  "http://www.w3.org/2001/XMLSchema#anyURI", and SHALL return an
  string in the form it was originally represented in XML form.

• urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-from-string
  This function SHALL take one argument of data-type
  "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
  dayTimeDuration, then the result SHALL be Indeterminate with status code

• urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-from-string
  This function SHALL take one argument of data-type
  "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
  yearMonthDuration, then the result SHALL be Indeterminate with status code

• urn:oasis:names:tc:xacml:3.0:function:string-from-yearMonthDuration
  This function SHALL take one argument of data-type
  "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
  yearMonthDuration from
  "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the string
  converted to a yearMonthDuration. If the argument is not a valid lexical representation of a
  yearMonthDuration, then the result SHALL be Indeterminate with status code

• urn:oasis:names:tc:xacml:3.0:function:string-from-x500Name
  This function SHALL take one argument of data-type
  "urn:oasis:names:tc:xacml:1.0:data-type:x500Name". The result SHALL be the string
  converted to an x500Name. If the argument is not a valid lexical representation of a X500Name, then the
  result SHALL be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.

• urn:oasis:names:tc:xacml:3.0:function:string-from-rfc822Name
  This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The result
  SHALL be the string converted to an rfc822Name. If the argument is not a valid lexical representation of an rfc822Name, then the
  result SHALL be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.
result SHALL be the rfc822Name converted to a string in the form it was originally represented in XML form.

- urn:oasis:names:tc:xacml:3.0:function:ipAddress-from-string
  This function SHALL take one argument of data-type
  "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
  "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress". The result SHALL be the string converted to an ipAddress. If the argument is not a valid lexical representation of an ipAddress, then the result SHALL be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.

- urn:oasis:names:tc:xacml:3.0:function:string-from-ipAddress
  This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the ipAddress converted to a string in the form it was originally represented in XML form.

- urn:oasis:names:tc:xacml:3.0:function:dnsName-from-string
  This function SHALL take one argument of data-type
  "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
  "urn:oasis:names:tc:xacml:2.0:data-type:dnsName". The result SHALL be the string converted to a dnsName. If the argument is not a valid lexical representation of a dnsName, then the result SHALL be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.

- urn:oasis:names:tc:xacml:3.0:function:string-from-dnsName
  This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:2.0:data-type:dnsName", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the dnsName converted to a string in the form it was originally represented in XML form.

- urn:oasis:names:tc:xacml:3.0:function:string-starts-with
  This function SHALL take two arguments of data-type
  "http://www.w3.org/2001/XMLSchema#string" and SHALL return a
  "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the second string begins with the first string, and false otherwise. Equality testing SHALL be done as defined for urn:oasis:names:tc:xacml:1.0:function:string-equal.

- urn:oasis:names:tc:xacml:3.0:function:anyURI-starts-with
  This function SHALL take a first argument of data-type
  "http://www.w3.org/2001/XMLSchema#string" and an a second argument of data-type
  "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return a
  "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the URI converted to a string with urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI begins with the string, and false otherwise. Equality testing SHALL be done as defined for urn:oasis:names:tc:xacml:1.0:function:string-equal.

- urn:oasis:names:tc:xacml:3.0:function:string-ends-with
  This function SHALL take two arguments of data-type
  "http://www.w3.org/2001/XMLSchema#string" and SHALL return a
  "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the second string ends with the first string, and false otherwise. Equality testing SHALL be done as defined for urn:oasis:names:tc:xacml:1.0:function:string-equal.

- urn:oasis:names:tc:xacml:3.0:function:anyURI-ends-with
  This function SHALL take a first argument of data-type
  "http://www.w3.org/2001/XMLSchema#string" and an a second argument of data-type
  "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return a
  "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the URI converted to a string with urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI ends with the string,
and false otherwise. Equality testing SHALL be done as defined for

• urn:oasis:names:tc:xacml:3.0:function:string-contains
This function SHALL take two arguments of data-type
"http://www.w3.org/2001/XMLSchema#string" and SHALL return a
"http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the second string
contains the first string, and false otherwise. Equality testing SHALL be done as defined for

• urn:oasis:names:tc:xacml:3.0:function:anyURI-contains
This function SHALL take a first argument of data-type
"http://www.w3.org/2001/XMLSchema#string" and an a second argument of data-type
"http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return a
"http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the URI converted
to a string with urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI contains the string, and
false otherwise. Equality testing SHALL be done as defined for

• urn:oasis:names:tc:xacml:3.0:function:string-substring
This function SHALL take a first argument of data-type
"http://www.w3.org/2001/XMLSchema#string" and a second and a third argument of type
"http://www.w3.org/2001/XMLSchema#integer" and SHALL return a
"http://www.w3.org/2001/XMLSchema#string". The result SHALL be the substring of the first
argument beginning at the position given by the second argument and ending at the position
before the position given by the third argument. The first character of the string has position zero.
The negative integer value -1 given for the third arguments indicates the end of the string. If the
second or third arguments are out of bounds, then the function MUST evaluate to indeterminate
with a status code of urn:oasis:names:tc:xacml:1.0:status:processing-error.

• urn:oasis:names:tc:xacml:3.0:function:anyURI-substring
This function SHALL take a first argument of data-type
"http://www.w3.org/2001/XMLSchema#anyURI" and a second and a third argument of type
"http://www.w3.org/2001/XMLSchema#integer" and SHALL return a
"http://www.w3.org/2001/XMLSchema#string". The result SHALL be the substring of the first
argument converted to a string with urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI
beginning at the position given by the second argument and ending at the position before the
position given by the third argument. The first character of the URI converted to a string has
position zero. The negative integer value -1 given for the third arguments indicates the end of the
string. If the second or third arguments are out of bounds, then the function MUST evaluate to
indeterminate with a status code of
urn:oasis:names:tc:xacml:1.0:status:processing-error. If the resulting substring
is not syntactically a valid URI, then the function MUST evaluate to Indeterminate with a status

A.3.10 Bag functions
These functions operate on a bag of 'type' values, where type is one of the primitive data-types, and x.x
is a version of XACML where the function has been defined. Some additional conditions defined for
each function below SHALL cause the expression to evaluate to "Indeterminate".

• urn:oasis:names:tc:xacml:x.x:function:type-one-and-only
This function SHALL take a bag of 'type' values as an argument and SHALL return a value of
'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".
• urn:oasis:names:tc:xacml:x.x:function: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.

• urn:oasis:names:tc:xacml:x.x:function:type-is-in
  This function SHALL take an argument of 'type' as the first argument and a bag of 'type' values as the second argument and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL evaluate to "True" if and only if the first argument matches by the "urn:oasis:names:tc:xacml:x.x:function:type-equal" any value in the bag. Otherwise, it SHALL return "False".

• urn:oasis:names:tc:xacml:x.x:function:type-bag
  This function SHALL take any number of arguments of '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.

A.3.11 Set functions

These functions operate on bags mimicking sets by eliminating duplicate elements from a bag.

• urn:oasis:names:tc:xacml:x.x:function:type-intersection
  This function SHALL take two arguments that are both a bag of 'type' values. It 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:x.x:function:type-equal". No duplicates, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal", SHALL exist in the result.

• urn:oasis:names:tc:xacml:x.x:function:type-at-least-one-member-of
  This function SHALL take two arguments that are both a bag of 'type' values. It SHALL return a "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL evaluate to "True" if and only if at least one element of the first argument is contained in the second argument as determined by "urn:oasis:names:tc:xacml:x.x:function:type-is-in".

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

• urn:oasis:names:tc:xacml:x.x:function:type-subset
  This function SHALL take two arguments that are both a bag of 'type' values. It SHALL return a "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first argument is a subset of the second argument. Each argument SHALL be considered to have had its duplicates removed, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal", before the subset calculation.

• urn:oasis:names:tc:xacml:x.x:function:type-set-equals
  This function SHALL take two arguments that are both a bag of 'type' values. It SHALL return a "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return the result of applying "urn:oasis:names:tc:xacml:1.0:function:and" to the application of "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the first and second arguments and the application of "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the second and first arguments.

A.3.12 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.
• urn:oasis:names:tc:xacml:1.3.0:function:any-of

This function applies a Boolean function between specific primitive values 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 n+1 arguments, where n is one or greater. The first argument SHALL be an <Function> element that names a Boolean function that takes n arguments of primitive types. Under the remaining n arguments, n-1 parameters SHALL be values of primitive data-types and one SHALL be a bag of a primitive data-type. The expression SHALL be evaluated as if the function named in the <Function> argument were applied to the n-1 non-bag arguments and each element of the bag argument and the results are combined with "urn:oasis:names:tc:xacml:1.0:function:or".

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

```
<Apply FunctionId="urn:oasis:names:tc:xacml:1.3.0:function:any-of">
  <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
  </Apply>
</Apply>
```

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

• urn:oasis:names:tc:xacml:1.3.0:function: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 n+1 arguments, where n is one or greater. The first argument SHALL be a <Function> element that names a Boolean function that takes n arguments of primitive types. Under the remaining n arguments, n-1 parameters SHALL be values of primitive data-types and one SHALL be a bag of a primitive data-type. The expression SHALL be evaluated as if the function named in the <Function> argument were applied to the n-1 non-bag arguments and each element of the bag argument and the results are combined with "urn:oasis:names:tc:xacml:1.0:function:and".

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

```
<Apply FunctionId="urn:oasis:names:tc:xacml:1.3.0:function:all-of">
  <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater-than"/>
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
    <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>
</Apply>
```
This expression is "True" because the first argument (10) is greater than all of the elements of the
`bag` (9, 3, 4 and 2).

- `urn:oasis:names:tc:xacml:1.0:function:any-of-any`

This function applies a Boolean function on each tuple from the cross product on all bags
arguments, and returns "True" if and only if the `predicate` is "True" for at least one inside-function
call.

This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL
be an `<Function>` element that names a Boolean function that takes n arguments. The
remaining arguments are either primitive data types or bags of primitive types. The expression
SHALL be evaluated as if the function named in the `<Function>` argument was applied between
every tuple of the cross product on all bags and the primitive values, 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 at least one
function call on the tuples from the bags and primitive values.

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

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

This expression is "True" because at least one of the elements of the first bag, namely "Ringo", is
equal to at least one of the elements of the second bag.

- `urn:oasis:names:tc:xacml:1.0:function:all-of-any`

This function applies a Boolean function between the elements of two bags. The expression
SHALL be "True" if and only if the supplied `predicate` is "True" between each element of the first
`bag` and any element of the second `bag`.

This function SHALL take three arguments. The first argument SHALL be an `<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
"urn:oasis:names:tc:xacml:1.0:function:any-of" function had been applied to each value of the
first bag and the whole of the second bag using the supplied xacml:Function, and the results
were then combined using "urn:oasis:names:tc:xacml:1.0:function:and".

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

```xml
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-any">
  <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-greater-than"/>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
  </Apply>
</Apply>
```
This expression is "True" because each of the elements of the first bag is greater than at least one of the elements of the second bag.

- urn:oasis:names:tc:xacml:1.0:function: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 supplied predicate is "True" between each element of the second bag and any element of the first bag.

  This function SHALL take three arguments. The first argument SHALL be an <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 "urn:oasis:names:tc:xacml:1.0:function:and" function had been applied to each value of the second bag and the whole of the first bag using the supplied xacml:Function, and the results were then combined using "urn:oasis:names:tc:xacml:1.0:function:and".

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

  This expression is "True" because, for all of the values in the second bag, there is a value in the first bag that is greater.

- urn:oasis:names:tc:xacml:1.0:function: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 supplied predicate is "True" between each and every element of the first bag collectively against all the elements of the second bag.

  This function SHALL take three arguments. The first argument SHALL be an <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 of the second argument and every element of the third argument 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`.

For example, the following expression SHALL evaluate to “True”:

```
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-all">
  <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than"/>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
  </Apply>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
    <AttributeValue DataType="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`.

- `urn:oasis:names:tc:xacml:1.3.0:function:map`

  This function converts a `bag` of values to another `bag` of values.

  This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL be a `<Function>` element naming a function that takes a n arguments of a primitive data-type and returns a value of a primitive data-type. Under the remaining n arguments, n-1 parameters SHALL be values of primitive data-types and one SHALL be a `bag` of a primitive data-type. The expression SHALL be evaluated as if the function named in the `<Function>` argument were applied to the n-1 non-bag arguments and each element of the bag argument and resulting in a `bag` of the converted value. The result SHALL be a `bag` of the primitive data-type that is returned by the function named in the `<dacml:Function>` element.

  For example, the following expression,

  ````
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.3.0: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>
  </Apply>
  ````

  evaluates to a `bag` containing “hello” and “world!”.

### A.3.13 Regular-expression-based functions

These functions operate on various types using regular expressions and evaluate to “http://www.w3.org/2001/XMLSchema#boolean”.

• urn:oasis:names:tc:xacml:1.0:function:string-regexp-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 7.6.2.

• urn:oasis:names:tc:xacml:2.0:function:anyURI-regexp-match

This function decides a regular expression match. It SHALL take two arguments; the first is of
type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
"http://www.w3.org/2001/XMLSchema#anyURI". It 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 URI. The function SHALL convert the second
argument to type "http://www.w3.org/2001/XMLSchema#string" with
urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI, then apply
"urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

• urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match

This function decides a regular expression match. It SHALL take two arguments; the first is of
type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
"urn:oasis:names:tc:xacml:2.0:data-type:ipAddress". It SHALL return an
"http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
expression and the second argument SHALL be an IPv4 or IPv6 address. The function SHALL convert the second
argument to type "http://www.w3.org/2001/XMLSchema#string" with
urn:oasis:names:tc:xacml:3.0:function:string-from-ipAddress, then apply
"urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

• urn:oasis:names:tc:xacml:2.0:function:dnsName-regexp-match

This function decides a regular expression match. It SHALL take two arguments; the first is of
type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
"urn:oasis:names:tc:xacml:2.0:data-type:dnsName". It 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 DNS name. The function SHALL convert the
second argument to type "http://www.w3.org/2001/XMLSchema#string" with
urn:oasis:names:tc:xacml:3.0:function:string-from-dnsName, then apply
"urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

• urn:oasis:names:tc:xacml:2.0:function:rfc822Name-regexp-match

This function decides a regular expression match. It SHALL take two arguments; the first is of
type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
"urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name". It SHALL return an
"http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
expression and the second argument SHALL be an RFC 822 name. The function SHALL convert the
second argument to type "http://www.w3.org/2001/XMLSchema#string" with
urn:oasis:names:tc:xacml:3.0:function:string-from-rfc822Name, then apply
"urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

• urn:oasis:names:tc:xacml:2.0:function:x500Name-regexp-match

This function decides a regular expression match. It SHALL take two arguments; the first is of
type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
"urn:oasis:names:tc:xacml:1.0:data-type:x500Name". It SHALL return an
"http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
expression and the second argument SHALL be an X.500 directory name. The function SHALL convert the second argument to type "http://www.w3.org/2001/XMLSchema#string" with
urn:oasis:names:tc:xacml:3.0:function:string-from-x500Name, then apply
"urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

5003
5004
5005
A.3.14 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.

- urn:oasis:names:tc:xacml:1.0:function: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.
  As an example (non-normative), if the first argument is "O=Medico Corp,C=US" and the second argument is "cn=John Smith,o=Medico Corp, c=US", then the function will return "True".

- urn:oasis:names:tc:xacml:1.0:function: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 a domain-part. The local-part is case-sensitive, while the domain-part (which is usually a DNS name) is not case-sensitive.
  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 address 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 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 second argument of "Anderson@sun.com" or "Baxter@SUN.COM", but not "Anderson@east.sun.com".
  In order to match any 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 "Anderson@east.sun.com" and "anne.anderson@ISRG.EAST.SUN.COM" but not "Anderson@sun.com".

A.3.15 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 isolation of the particular function specified. The context nodes and namespace mappings of the XPath expressions are defined by the XPath data-type, see section B.3. The following functions are defined:

- urn:oasis:names:tc:xacml:3.0:function:xpath-node-count
  This function SHALL take an "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression" as an argument 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 match the given XPath expression. If the <Content> element of the category to which the XPath expression applies to is not present in the request, this function SHALL return a value of zero.

- urn:oasis:names:tc:xacml:3.0:function:xpath-node-equal
This function SHALL take two "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression" arguments and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if any of the XML nodes in the node-set matched by the first argument equals any of the XML nodes in the node-set matched by the second argument. Two nodes are considered equal if they have the same identity. If the <Content> element of the category to which either XPath expression applies to is not present in the request, this function SHALL return a value of "False".

- urn:oasis:names:tc:xacml:3.0:function:xpath-node-match

This function SHALL take two "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression" arguments and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if one of the following two conditions is satisfied: (1) Any of the XML nodes in the node-set matched by the first argument is equal to any of the XML nodes in the node-set matched by the second argument; (2) any node below any of the XML nodes in the node-set matched by the first argument is equal to any of the XML nodes in the node-set matched by the second argument. Two nodes are considered equal if they have the same identity. If the <Content> element of the category to which either XPath expression applies to is not present in the request, this function SHALL return a value of "False".

NOTE: The first condition is equivalent to "xpath-node-equal", and guarantees that "xpath-node-equal" is a special case of "xpath-node-match".

A.3.16 Other functions

- urn:oasis:names:tc:xacml:3.0:function:access-permitted

This function SHALL take an "http://www.w3.org/2001/XMLSchema#anyURI" and an "http://www.w3.org/2001/XMLSchema#string" as arguments. The first argument SHALL be interpreted as an attribute category. The second argument SHALL be interpreted as the XML content of an <Attributes> element with Category equal to the first argument. The function evaluates to an "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL return "True" if and only if the policy evaluation described below returns the value of "Permit".

The following evaluation is described as if the context is actually instantiated, but it is only required that an equivalent result be obtained.

The function SHALL construct a new context, by copying all the information from the current context, omitting any <Attributes> element with Category equal to the first argument. The second function argument SHALL be added to the context as the content of an <Attributes> element with Category equal to the first argument.

The function SHALL invoke a complete policy evaluation using the newly constructed context. This evaluation SHALL be completely isolated from the evaluation which invoked the function, but shall use all current policies and combining algorithms, including any per request policies.

The PDP SHALL detect any loop which may occur if successive evaluations invoke this function by counting the number of total invocations of any instance of this function during any single initial invocation of the PDP. If the total number of invocations exceeds the bound for such invocations, the initial invocation of this function evaluates to Indeterminate with a "urn:oasis:names:tc:xacml:1.0:status:processing-error" status code. Also, see the security considerations in section 9.1.8.

A.3.17 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 the integrity of the XACML evaluation strategy, the result of an extension function 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.

A.4 Functions, data types, attributes and algorithms planned for
deprecation

The following functions, data types and algorithms have been defined by previous versions of XACML
and newer and better alternatives are defined in XACML 3.0. Their use is discouraged for new use and
they are candidates for deprecation in future versions of XACML.

The following xpath based functions have been replaced with equivalent functions which use the new

- urn:oasis:names:tc:xacml:1.0:function:xpath-node-count
  - Replaced with urn:oasis:names:tc:xacml:3.0:function:xpath-node-count
- urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal
  - Replaced with urn:oasis:names:tc:xacml:3.0:function:xpath-node-equal
- urn:oasis:names:tc:xacml:1.0:function:xpath-node-match
  - Replaced with urn:oasis:names:tc:xacml:3.0:function:xpath-node-match

The following URI and string concatenation function has been replaced with a string to URI conversion
function, which allows the use of the general string functions with URI through string conversion.

- urn:oasis:names:tc:xacml:2.0:function:uri-string-concatenate
  - Replaced by urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI

The following identifiers have been replaced with official identifiers defined by W3C.

- http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration
  - Replaced with http://www.w3.org/2001/XMLSchema#dayTimeDuration
- http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration
  - Replaced with http://www.w3.org/2001/XMLSchema#yearMonthDuration

The following functions have been replaced with functions which use the updated dayTimeDuration and
yearMonthDuration data types.

- urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-equal
  - Replaced with urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-equal
- urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-equal
  - Replaced with urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-equal
- urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration
  - Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-add-dayTimeDuration
- urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration
  - Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-add-yearMonthDuration
- urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-dayTimeDuration
  - Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-dayTimeDuration
- urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-yearMonthDuration
  - Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-yearMonthDuration
- urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration
  - Replaced with urn:oasis:names:tc:xacml:3.0:function:date-add-yearMonthDuration
- urn:oasis:names:tc:xacml:1.0:function:date-subtract-yearMonthDuration
  - Replaced with urn:oasis:names:tc:xacml:3.0:function:date-subtract-yearMonthDuration
The following attribute identifiers have been replaced with new identifiers:

- urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address
  - Replaced with urn:oasis:names:tc:xacml:3.0:subject:authn-locality:ip-address
- urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name
  - Replaced with urn:oasis:names:tc:xacml:3.0:subject:authn-locality:dns-name

The following combining algorithms have been replaced with new variants which allow for better handling of "Indeterminate" results:

  - Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides
  - Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides
  - Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-permit-overrides
Appendix B. XACML identifiers (normative)

This section defines standard identifiers for commonly used entities.

B.1 XACML namespaces

XACML is defined using this identifier.

urn:oasis:names:tc:xacml:3.0:core:schema

B.2 Attribute categories

The following attribute category identifiers MUST be used when an XACML 2.0 or earlier policy or request is translated into XACML 3.0.

Attributes previously placed in the Resource, Action, and Environment sections of a request are placed in an attribute category with the following identifiers respectively. It is RECOMMENDED that they are used to list attributes of resources, actions, and the environment respectively when authoring XACML 3.0 policies or requests.

urn:oasis:names:tc:xacml:3.0:attribute-category:resource
urn:oasis:names:tc:xacml:3.0:attribute-category:action
urn:oasis:names:tc:xacml:3.0:attribute-category:environment

Attributes previously placed in the Subject section of a request are placed in an attribute category which is identical of the subject category in XACML 2.0, as defined below. It is RECOMMENDED that they are used to list attributes of subjects when authoring XACML 3.0 policies or requests.

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 in XACML 2.0, this is the default translation value.

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).

urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject

This identifier indicates a system entity through which the access request was passed.

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.

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.

urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine

B.3 Data-types

The following identifiers indicate data-types that are defined in Section A.2.

urn:oasis:names:tc:xacml:1.0:data-type:x500Name.
urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name
urn:oasis:names:tc:xacml:2.0:data-type:ipAddress
urn:oasis:names:tc:xacml:2.0:data-type:dnsName
urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression
The following data-type identifiers are defined by XML Schema [XS].

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 10.3.2 and 10.3.1, respectively.

http://www.w3.org/2001/XMLSchema#dayTimeDuration
http://www.w3.org/2001/XMLSchema#yearMonthDuration

B.4 Subject attributes

These identifiers indicate attributes of a subject. When used, it is RECOMMENDED that they appear within an <Attributes> element of the request context with a subject category (see section B.2).

At most one of each of these attributes is associated with each subject. Each attribute associated with authentication included within a single <Attributes> element relates to the same authentication event.

This identifier indicates the name of the subject.

urn:oasis:names:tc:xacml:1.0:subject:subject-id

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.

This identifier indicates that the location is expressed as an IP address.

urn:oasis:names:tc:xacml:3.0:subject:authn-locality:ip-address

The corresponding attribute SHALL be of data-type "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress ".

This identifier indicates that the location is expressed as a DNS name.

urn:oasis:names:tc:xacml:3.0:subject:authn-locality:dns-name

The corresponding attribute SHALL be of data-type "urn:oasis:names:tc:xacml:2.0:data-type:dnsName ".
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 RFC 2256 SHALL be:

http://www.ietf.org/rfc/rfc2256.txt#userPassword

### B.5 Resource attributes

These identifiers indicate attributes of the resource. When used, it is RECOMMENDED they appear within the <Attributes> element of the request context with Category urn:oasis:names:tc:xacml:3.0:attribute-category:resource.

This attribute identifies the resource to which access is requested.

This attribute identifies the namespace of the top element(s) of the contents of the <Content> element. In the case where the resource content is supplied in the request context and the resource namespaces are defined in the resource, the PEP MAY provide this attribute in the request to indicate the namespaces of the resource content. In this case there SHALL be one value of this attribute for each unique namespace of the top level elements in the <Content> element. The type of the corresponding attribute SHALL be "http://www.w3.org/2001/XMLSchema#anyURI".

### B.6 Action attributes

These identifiers indicate attributes of the action being requested. When used, it is RECOMMENDED they appear within the <Attributes> element of the request context with Category urn:oasis:names:tc:xacml:3.0:attribute-category:action.

This attribute identifies the action for which access is requested.

Where the action is implicit, the value of the action-id attribute SHALL be

This attribute identifies the namespace in which the action-id attribute is defined.

### B.7 Environment attributes

These identifiers indicate attributes of the environment within which the decision request is to be evaluated. When used in the decision request, it is RECOMMENDED they appear in the <Attributes> element of the request context with Category urn:oasis:names:tc:xacml:3.0:attribute-category:environment.

This identifier indicates the current time at the context handler. In practice it is the time at which the request context was created. For this reason, if these identifiers appear in multiple places within a <Policy> or <PolicySet>, then the same value SHALL be assigned to each occurrence in the evaluation procedure, regardless of how much time elapses between the processing of the occurrences.

The corresponding attribute SHALL be of data-type "http://www.w3.org/2001/XMLSchema#time".

The corresponding attribute SHALL be of data-type "http://www.w3.org/2001/XMLSchema#date".

The corresponding attribute SHALL be of data-type "http://www.w3.org/2001/XMLSchema#dateTime".
B.8 Status codes

The following status code values are defined.

- This identifier indicates success.
  - `urn:oasis:names:tc:xacml:1.0:status:ok`

- This identifier indicates that all the attributes necessary to make a policy decision were not available (see Section 5.58).
  - `urn:oasis:names:tc:xacml:1.0:status:missing-attribute`

- This identifier indicates that some attribute value contained a syntax error, such as a letter in a numeric field.
  - `urn:oasis:names:tc:xacml:1.0:status:syntax-error`

- This identifier indicates that an error occurred during policy evaluation. An example would be division by zero.
  - `urn:oasis:names:tc:xacml:1.0:status:processing-error`

B.9 Combining algorithms

- The deny-overrides rule-combining algorithm has the following value for the ruleCombiningAlgId attribute:

- The deny-overrides policy-combining algorithm has the following value for the policyCombiningAlgId attribute:

- The permit-overrides rule-combining algorithm has the following value for the ruleCombiningAlgId attribute:

- The permit-overrides policy-combining algorithm has the following value for the policyCombiningAlgId attribute:

- The first-applicable rule-combining algorithm has the following value for the ruleCombiningAlgId attribute:
  - `urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable`

- The first-applicable policy-combining algorithm has the following value for the policyCombiningAlgId attribute:
  - `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 the policyCombiningAlgId attribute:
  - `urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable`

- The ordereddeny-overrides rule-combining algorithm has the following value for the ruleCombiningAlgId attribute:

- The ordereddeny-overrides policy-combining algorithm has the following value for the policyCombiningAlgId attribute:

- The ordered-permit-overrides rule-combining algorithm has the following value for the ruleCombiningAlgId attribute:
The ordered-permit overrides **policy-combining algorithm** has the following value for the
policyCombiningAlgId attribute:

The deny-unless-permit **rule-combining algorithm** has the following value for the
policyCombiningAlgId attribute:

The permit-unless-deny **rule-combining algorithm** has the following value for the
policyCombiningAlgId attribute:

The legacy deny-overrides **rule-combining algorithm** has the following value for the
ruleCombiningAlgId attribute:

The legacy deny-overrides **policy-combining algorithm** has the following value for the
policyCombiningAlgId attribute:

The legacy permit-overrides **rule-combining algorithm** has the following value for the
ruleCombiningAlgId attribute:

The legacy permit-overrides **policy-combining algorithm** has the following value for the
policyCombiningAlgId attribute:

The legacy ordered-deny-overrides **rule-combining algorithm** has the following value for the
ruleCombiningAlgId attribute:

The legacy ordered-deny-overrides **policy-combining algorithm** has the following value for the
policyCombiningAlgId attribute:

The legacy ordered-permit-overrides **rule-combining algorithm** has the following value for the
ruleCombiningAlgId attribute:

The legacy ordered-permit-overrides **policy-combining algorithm** has the following value for the
policyCombiningAlgId attribute:
Appendix C. Combining algorithms (normative)

This section contains a description of the rule- and policy-combining algorithms specified by XACML.

Pseudo code is normative, descriptions in English are non-normative.

The legacy combining algorithms are defined in previous versions of XACML, and are retained for compatibility reasons. It is RECOMMENDED that the new combining algorithms are used instead of the legacy combining algorithms for new use.

Note that in each case an implementation is conformant as long as it produces the same result as is specified here, regardless of how and in what order the implementation behaves internally.

C.1 Extended Indeterminate values

Some combining algorithms are defined in terms of an extended set of "Indeterminate" values. See section 7.10 for the definition of the Extended Indeterminate values. For these algorithms, the PDP MUST keep track of the extended set of "Indeterminate" values during rule and policy combining.

The output of a combining algorithm which does not track the extended set of "Indeterminate" values MUST be treated as "Indeterminate(DP)" for the value "Indeterminate" by a combining algorithm which tracks the extended set of "Indeterminate" values.

A combining algorithm which does not track the extended set of "Indeterminate" values MUST treat the output of a combining algorithm which tracks the extended set of "Indeterminate" values as an "Indeterminate" for any of the possible values of the extended set of "Indeterminate".

C.2 Deny-overrides

This section defines the “Deny-overrides” rule-combining algorithm of a policy and policy-combining algorithm of a policy set.

This combining algorithm makes use of the extended "Indeterminate". The rule combining algorithm defined here has the following identifier:

urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides

The policy combining algorithm defined here has the following identifier:


The following is a non-normative informative description of this combining algorithm.

The deny overrides combining algorithm is intended for those cases where a deny decision should have priority over a permit decision. This algorithm has the following behavior.

1. If any decision is "Deny", the result is "Deny".
2. Otherwise, if any decision is "Indeterminate(DP)" , the result is "Indeterminate(DP)".
3. Otherwise, if any decision is "Indeterminate(D)" and another decision is "Indeterminate(P) or Permit", the result is "Indeterminate(DP)".
4. Otherwise, if any decision is "Indeterminate(D)" , the result is "Indeterminate(D)".
5. Otherwise, if any decision is "Permit", the result is "Permit".
6. Otherwise, if any decision is "Indeterminate(P)" , the result is "Indeterminate(P)".
7. Otherwise, the result is "NotApplicable".

The following pseudo-code represents the normative specification of this combining algorithm. The algorithm is presented here in a form where the input to it is an array with children (the policies, policy sets or rules) of the policy or policy set. The children may be processed in any order, so the set of obligations or advice provided by this algorithm is not deterministic.
Decision denyOverridesCombiningAlgorithm(Node[] children)
{
    Boolean atLeastOneErrorD = false;
    Boolean atLeastOneErrorP = false;
    Boolean atLeastOneErrorDP = false;
    Boolean atLeastOnePermit = false;
    for( i=0 ; i < lengthOf(children) ; i++ )
    {
        Decision decision = children[i].evaluate();
        if (decision == Deny)
        {
            return Deny;
        }
        if (decision == Permit)
        {
            atLeastOnePermit = true;
            continue;
        }
        if (decision == NotApplicable)
        {
            continue;
        }
        if (decision == Indeterminate(D))
        {
            atLeastOneErrorD = true;
            continue;
        }
        if (decision == Indeterminate(P))
        {
            atLeastOneErrorP = true;
            continue;
        }
        if (decision == Indeterminate(DP))
        {
            atLeastOneErrorDP = true;
            continue;
        }
        if (atLeastOneErrorDP)
        {
            return Indeterminate(DP);
        }
        if (atLeastOneErrorD && (atLeastOneErrorP || atLeastOnePermit))
        {
            return Indeterminate(DP);
        }
        if (atLeastOneErrorD)
        {
            return Indeterminate(D);
        }
        if (atLeastOnePermit)
        {
            return Permit;
        }
        if (atLeastOneErrorP)
        {
            return Indeterminate(P);
        }
        return NotApplicable;
    }

    Obligations and advice shall be combined as described in Section 7.18.
C.3 Ordered-deny-overrides

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 rule combining algorithm defined here has the following identifier:

urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides

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.

The policy combining algorithm defined here has the following identifier:


C.4 Permit-overrides

This section defines the "Permit-overrides" rule-combining algorithm of a policy and policy-combining algorithm of a policy set.

This combining algorithm makes use of the extended "Indeterminate".

The rule combining algorithm defined here has the following identifier:

urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides

The policy combining algorithm defined here has the following identifier:


The following is a non-normative informative description of this combining algorithm.

The permit overrides combining algorithm is intended for those cases where a permit decision should have priority over a deny decision. This algorithm has the following behavior.

1. If any decision is "Permit", the result is "Permit".
2. Otherwise, if any decision is "Indeterminate(DP)", the result is "Indeterminate(DP)".
3. Otherwise, if any decision is "Indeterminate(P)" and another decision is "Indeterminate(D) or Deny", the result is "Indeterminate(DP)".
4. Otherwise, if any decision is "Indeterminate(P)", the result is "Indeterminate(P)".
5. Otherwise, if any decision is "Deny", the result is "Deny".
6. Otherwise, if any decision is "Indeterminate(D)", the result is "Indeterminate(D)".
7. Otherwise, the result is "NotApplicable".

The following pseudo-code represents the normative specification of this combining algorithm. The algorithm is presented here in a form where the input to it is an array with all children (the policies, policy sets or rules) of the policy or policy set. The children may be processed in any order, so the set of obligations or advice provided by this algorithm is not deterministic.

```plaintext
Decision permitOverridesCombiningAlgorithm(Node[] children) {
    Boolean atLeastOneErrorD = false;
    Boolean atLeastOneErrorP = false;
    Boolean atLeastOneErrorDP = false;
    Boolean atLeastOneDeny = false;
```
for (i = 0; i < lengthOf(children); i++)
{
    Decision decision = children[i].evaluate();
    if (decision == Deny)
    {
        atLeastOneDeny = true;
        continue;
    }
    if (decision == Permit)
    {
        return Permit;
    }
    if (decision == NotApplicable)
    {
        continue;
    }
    if (decision == Indeterminate{D})
    {
        atLeastOneErrorD = true;
        continue;
    }
    if (decision == Indeterminate{P})
    {
        atLeastOneErrorP = true;
        continue;
    }
    if (decision == Indeterminate{DP})
    {
        atLeastOneErrorDP = true;
        continue;
    }
    if (atLeastOneErrorDP)
    {
        return Indeterminate{DP};
    }
    if (atLeastOneErrorP && (atLeastOneErrorD || atLeastOneDeny))
    {
        return Indeterminate{DP};
    }
    if (atLeastOneErrorP)
    {
        return Indeterminate{P};
    }
    if (atLeastOneDeny)
    {
        return Deny;
    }
    if (atLeastOneErrorD)
    {
        return Indeterminate{D};
    }
    return NotApplicable;
}

Obligations and advice shall be combined as described in Section 7.18.

C.5 Ordered-permit-overrides

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 `rule combining algorithm` defined here has the following identifier:
urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-permit-overrides

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`.

The `policy combining algorithm` defined here has the following identifier:
urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-permit-overrides

C.6 Deny-unless-permit

This section defines the "Deny-unless-permit" `rule-combining algorithm` of a `policy` or `policy-combining algorithm` of a `policy set`.

The `rule combining algorithm` defined here has the following identifier:
urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit

The `policy combining algorithm` defined here has the following identifier:

The following is a non-normative informative description of this `combining algorithm`.

The "Deny-unless-permit" `combining algorithm` is intended for those cases where a permit decision should have priority over a deny decision, and an "Indeterminate" or "NotApplicable" must never be the result. It is particularly useful at the top level in a `policy` structure to ensure that a `PDP` will always return a definite "Permit" or "Deny" result. This algorithm has the following behavior.

1. If any decision is "Permit", the result is "Permit".
2. Otherwise, the result is "Deny".

The following pseudo-code represents the normative specification of this `combining algorithm`. The algorithm is presented here in a form where the input to it is an array with all the children (the `policies`, `policy sets` or `rules`) of the `policy` or `policy set`. The children may be processed in any order, so the set of obligations or advice provided by this algorithm is not deterministic.

```plaintext
Decision denyUnlessPermitCombiningAlgorithm(Node[] children)
{
  for( i=0 ; i < lengthOf(children) ; i++ )
  {
    if (children[i].evaluate() == Permit)
      return Permit;
  }
  return Deny;
}
```

`Obligations` and `advice` shall be combined as described in Section 7.18.

C.7 Permit-unless-deny

This section defines the "Permit-unless-deny" `rule-combining algorithm` of a `policy` or `policy-combining algorithm` of a `policy set`.

The `rule combining algorithm` defined here has the following identifier:
urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-unless-deny
The policy combining algorithm defined here has the following identifier:

urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-unless-deny

The following is a non-normative informative description of this combining algorithm.

The "Permit-unless-deny" combining algorithm is intended for those cases where a deny decision should have priority over a permit decision, and an "Indeterminate" or "NotApplicable" must never be the result. It is particularly useful at the top level in a policy structure to ensure that a PDP will always return a definite "Permit" or "Deny" result. This algorithm has the following behavior.

1. If any decision is "Deny", the result is "Deny".
2. Otherwise, the result is "Permit".

The following pseudo-code represents the normative specification of this combining algorithm. The algorithm is presented here in a form where the input to it is an array with all the children (the policies, policy sets or rules) of the policy or policy set. The children may be processed in any order, so the set of obligations or advice provided by this algorithm is not deterministic.

```java
Decision permitUnlessDenyCombiningAlgorithm(Node[] children)
{
    for( i=0 ; i < lengthOf(children) ; i++ )
    {
        if (children[i].evaluate() == Deny)
        {
            return Deny;
        }
    }
    return Permit;
}
```

Obligations and advice shall be combined as described in Section 7.18.

C.8 First-applicable

This section defines the "First-applicable" rule-combining algorithm of a policy and policy-combining algorithm of a policy set.

The rule combining algorithm defined here has the following identifier:

urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable

The following is a non-normative informative description of 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 normative specification of this rule-combining algorithm.

```java
Decision firstApplicableEffectRuleCombiningAlgorithm(Rule[] rules)
{
    for( i = 0 ; i < lengthOf(rules) ; i++ )
    {
        Decision decision = evaluate(rules[i]);
        if (decision == Deny)
        {
            return Deny;
        }
    }
    return Permit;
}
```
The **policy combining algorithm** defined here has the following identifier:

```
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable
```

The following is a non-normative informative description of 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 normative specification of this policy-combination algorithm.

```java
return Deny;
}
if (decision == Permit)
{
    return Permit;
}
```}

```
if (decision == NotApplicable)
{
    continue;
}
if (decision == Indeterminate)
{
    return Indeterminate;
}
return NotApplicable;
}
```

```
Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy[] policies)
{
    for( i = 0 ; i < lengthOf(policies) ; i++ )
    {
        Decision decision = evaluate(policies[i]);
        if(decision == Deny)
        {
            return Deny;
        }
        if(decision == Permit)
        {
            return Permit;
        }
        if (decision == NotApplicable)
        {
            continue;
        }
        if (decision == Indeterminate)
        {
            return Indeterminate;
        }
    }
    return NotApplicable;
}
```

**Obligations** and **advice** of the individual **policies** shall be combined as described in Section 7.18.
C.9 Only-one-applicable

This section defines the "Only-one-applicable" policy-combining algorithm of a policy set.

The policy combining algorithm defined here has the following identifier:

urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable

The following is a non-normative informative description of 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 its target, then the result of the policy-combination algorithm SHALL be "NotApplicable". If more than one policy is considered applicable by virtue of its target, then the result of the policy-combination algorithm SHALL be "Indeterminate".

If only one policy is considered applicable by evaluation of its target, 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 normative specification of this policy-combining algorithm.

```java
Decision onlyOneApplicablePolicyPolicyCombiningAlgorithm(Policy[] policies)
{
    Boolean atLeastOne = false;
    Policy selectedPolicy = null;
    ApplicableResult appResult;

    for ( i = 0; i < lengthOf(policies) ; i++ )
    {
        appResult = isApplicable(policies[I]);

        if ( appResult == Indeterminate )
        {
            return Indeterminate;
        }
        if( appResult == Applicable )
        {
            if ( atLeastOne )
            {
                return Indeterminate;
            }
            else
            {
                atLeastOne = true;
                selectedPolicy = policies[i];
            }
        }
        if ( appResult == NotApplicable )
        {
            continue;
        }
        if ( atLeastOne )
        {
            return evaluate(selectedPolicy);
        }
        else
        {
            return NotApplicable;
        }
    }
}
```

Obligations and advice of the individual rules shall be combined as described in Section 7.18.
C.10 Legacy Deny-overrides

This section defines the legacy "Deny-overrides" rule-combining algorithm of a policy and policy-combining algorithm of a policy set.

The rule combining algorithm defined here has the following identifier:

urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

The following is a non-normative informative description of this combining algorithm.

The "Deny–overrides" rule combining algorithm is intended for those cases where a deny decision should have priority over a permit decision. This algorithm has the following behavior.

1. If any rule evaluates to "Deny", the result is "Deny".
2. Otherwise, if any rule having Effect="Deny" evaluates to "Indeterminate", the result is "Indeterminate".
3. Otherwise, if any rule evaluates to "Permit", the result is "Permit".
4. Otherwise, if any rule having Effect="Permit" evaluates to "Indeterminate", the result is "Indeterminate".
5. Otherwise, the result is "NotApplicable".

The following pseudo-code represents the normative specification of this rule-combining algorithm.

```java
Decision denyOverridesRuleCombiningAlgorithm(Rule[] rules)
{
    Boolean atLeastOneError = false;
    Boolean potentialDeny = false;
    Boolean atLeastOnePermit = false;
    for( i=0 ; i < lengthOf(rules) ; i++ )
    {
        Decision decision = evaluate(rules[i]);
        if (decision == Deny)
        {
            return Deny;
        }
        if (decision == Permit)
        {
            atLeastOnePermit = true;
            continue;
        }
        if (decision == NotApplicable)
        {
            continue;
        }
        if (decision == Indeterminate)
        {
            atLeastOneError = true;
            if (effect(rules[i]) == Deny)
            {
                potentialDeny = true;
            }
            continue;
        }
        if (potentialDeny)
        {
            return Indeterminate;
        }
        if (atLeastOnePermit)
        {
```
Obligations and advice of the individual rules shall be combined as described in Section 7.18.

The policy combining algorithm defined here has the following identifier:


The following is a non-normative informative description of this combining algorithm.

The "Deny–overrides" policy combining algorithm is intended for those cases where a deny decision should have priority over a permit decision. This algorithm has the following behavior.

1. If any policy evaluates to "Deny", the result is "Deny".
2. Otherwise, if any policy evaluates to "Indeterminate", the result is "Deny".
3. Otherwise, if any policy evaluates to "Permit", the result is "Permit".
4. Otherwise, the result is "NotApplicable".

The following pseudo-code represents the normative specification of this policy-combining algorithm.

```plaintext
Decision denyOverridesPolicyCombiningAlgorithm(Policy[] policies)
{
    Boolean atLeastOnePermit = false;
    for( i=0 ; i < lengthOf(policies) ; i++ )
    {
        Decision decision = evaluate(policies[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 and advice of the individual policies shall be combined as described in Section 7.18.

C.11 Legacy Ordered-deny-overrides

The following specification defines the legacy "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 rule combining algorithm defined here has the following identifier:


The following specification defines the legacy "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.

The rule combining algorithm defined here has the following identifier:


### C.12 Legacy Permit-overrides

This section defines the legacy “Permit-overrides” rule-combining algorithm of a policy and policy-combining algorithm of a policy set.

The rule combining algorithm defined here has the following identifier:

urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides

The following is a non-normative informative description of this combining algorithm.

The "Permit-overrides" rule combining algorithm is intended for those cases where a permit decision should have priority over a deny decision. This algorithm has the following behavior.

1. If any rule evaluates to "Permit", the result is "Permit".
2. Otherwise, if any rule having Effect="Permit" evaluates to "Indeterminate" the result is "Indeterminate".
3. Otherwise, if any rule evaluates to "Deny", the result is "Deny".
4. Otherwise, if any rule having Effect="Deny" evaluates to "Indeterminate", the result is "Indeterminate".
5. Otherwise, the result is "NotApplicable".

The following pseudo-code represents the normative specification of this rule-combining algorithm.

```plaintext
Decision permitOverridesRuleCombiningAlgorithm(Rule[] rules)
{
    Boolean atLeastOneError = false;
    Boolean potentialPermit = false;
    Boolean atLeastOneDeny = false;
    for( i=0 ; i < lengthOf(rules) ; i++ )
    {
        Decision decision = evaluate(rules[i]);
        if (decision == Deny)
        {
            atLeastOneDeny = true;
            continue;
        }
        if (decision == Permit)
        {
            return Permit;
        }
        if (decision == NotApplicable)
        {
            continue;
        }
    }
    return Permit;
}
```
if (decision == Indeterminate)
{
    atLeastOneError = true;
    if (effect(rules[i]) == Permit)
    {
        potentialPermit = true;
    }
    continue;
}
}
if (potentialPermit)
{
    return Indeterminate;
}
if (atLeastOneDeny)
{
    return Deny;
}
if (atLeastOneError)
{
    return Indeterminate;
}
return NotApplicable;

Obligations and advice of the individual rules shall be combined as described in Section 7.18. The policy combining algorithm defined here has the following identifier: urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides

The following is a non-normative informative description of this combining algorithm.

The "Permit–overrides" policy combining algorithm is intended for those cases where a permit decision should have priority over a deny decision. This algorithm has the following behavior.

1. If any policy evaluates to "Permit", the result is "Permit".
2. Otherwise, if any policy evaluates to "Deny", the result is "Deny".
3. Otherwise, if any policy evaluates to "Indeterminate", the result is "Indeterminate".
4. Otherwise, the result is "NotApplicable".

The following pseudo-code represents the normative specification of this policy-combining algorithm.

```java
Decision permitOverridesPolicyCombiningAlgorithm(Policy[] policies)
{
    Boolean atLeastOneError = false;
    Boolean atLeastOneDeny = false;
    for( i=0 ; i < lengthOf(policies) ; i++ )
    {
        Decision decision = evaluate(policies[i]);
        if (decision == Deny)
        {
            atLeastOneDeny = true;
        }
        continue;
        if (decision == Permit)
        {
            return Permit;
        }
        if (decision == NotApplicable)
        {
            continue;
        }
        if (decision == Indeterminate)
        {
```


```plaintext
{
    atLeastOneError = true;
    continue;
}

if (atLeastOneDeny)
{
    return Deny;
}
if (atLeastOneError)
{
    return Indeterminate;
}
return NotApplicable;
```

**Obligations** and **advice** of the individual **policies** shall be combined as described in Section 7.18.

### C.13 Legacy Ordered-permit-overrides

The following specification defines the legacy "Ordered-permit-overrides" **rule-combining algorithm** of a **policy**.

The behavior of this algorithm is identical to that of the “Permitoverrides” **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 **rule combining algorithm** defined here has the following identifier:

```
```

The following specification defines the legacy "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**.

The **policy combining algorithm** defined here has the following identifier:

```
```
Appendix D. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Anil Saldhana
Anil Tappetla
Anne Anderson
Anthony Nadalin
Bill Parducci
Craig Forster
David Chadwick
David Staggs
Dilli Arumugam
Duane DeCouteau
Erik Rissanen
Gareth Richards
Hal Lockhart
Jan Herrmann
John Tolbert
Ludwig Seitz
Michiharu Kudo
Naomaru Itoi
Paul Tyson
Prateek Mishra
Rich Levinson
Ronald Jacobson
Seth Proctor
Sridhar Muppidi
Tim Moses
Vernon Murdoch
## Appendix E. Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Editor</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>WD 05</td>
<td>10 Oct 2007</td>
<td>Erik Rissanen</td>
<td>Convert to new OASIS template. Fixed typos and errors.</td>
</tr>
<tr>
<td>WD 06</td>
<td>18 May 2008</td>
<td>Erik Rissanen</td>
<td>Added missing MaxDelegationDepth in schema fragments. Added missing urn:oasis:names:tc:xacml:1.0:resource:xpath identifier. Corrected typos on xpaths in the example policies. Removed use of xpointer in the examples. Made the &lt;Content&gt; element the context node of all xpath expressions and introduced categorization of XPaths so they point to a specific &lt;Content&gt; element. Added &lt;Content&gt; element to the policy issuer. Added description of the &lt;PolicyIssuer&gt; element. Updated the schema figure in the introduction to reflect the new AllOf/AnyOf schema. Remove duplicate &lt;CombinerParameters&gt; element in the &lt;Policy&gt; element in the schema. Removed default attributes in the schema. (Version in &lt;Policy(Set)&gt; and MustBePresent in &lt;AttributeDesignator&gt; in &lt;AttributeSelector&gt;) Removed references in section 7.3 to the &lt;Condition&gt; element having a FunctionId attribute. Fixed typos in data type URIs in section A.3.7.</td>
</tr>
<tr>
<td>WD 07</td>
<td>3 Nov 2008</td>
<td>Erik Rissanen</td>
<td>Fixed &quot;….data-types:…&quot; typo in conformace section. Removed XML default attribute for IncludeInResult for element &lt;Attribute&gt;. Also added this attribute in the associated schema file. Removed description of non-existing XML attribute &quot;ResourceId&quot; from the element &lt;Result&gt;. Moved the urn:oasis:names:tc:xacml:3.0:function:access-permitted function into here from the delegation profile.</td>
</tr>
</tbody>
</table>
Updated the daytime and yearmonth duration data types to the W3C defined identifiers.
Added <Description> to <Apply>.
Added XPath versioning to the request.
Added security considerations about denial service and the access-permitted function.
Changed <Target> matching so NoMatch has priority over Indeterminate.
Fixed multiple typos in identifiers.
Lower case incorrect use of “MAY”.
Misc minor typos.
Removed whitespace in example attributes.
Removed an incorrect sentence about higher order functions in the definition of the <Function> element.
Clarified evaluation of empty or missing targets.
Use Unicode codepoint collation for string comparisons.
Support multiple arguments in multiply functions.
Define Indeterminate result for overflow in integer to double conversion.
Simplified descriptions of deny/permit overrides algorithms.
Add ipAddress and dnsName into conformance section.
Don’t refer to IEEE 754 for integer arithmetic.
Rephrase indeterminate result for arithmetic functions.
Fix typos in examples.
Clarify Match evaluation and drop list of example functions which can be used in a Match.
Added behavior for circular policy/variable references.
Fix obligation enforcement so it refers to PEP bias.
Added Version xml attribute to the example policies.
Remove requirement for PDP to check the target-namespace resource attribute.
Added policy identifier list to the response/request.
Added statements about Unicode normalization.
Clarified definitions of string functions.
<table>
<thead>
<tr>
<th>Date</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>WD 08 5 Feb 2009</td>
<td>Added new string functions. Added section on Unicode security issues.</td>
</tr>
<tr>
<td></td>
<td>Updated Unicode normalization section according to suggestion from W3C working group.</td>
</tr>
<tr>
<td></td>
<td>Set union functions now may take more than two arguments.</td>
</tr>
<tr>
<td></td>
<td>Made obligation parameters into runtime expressions.</td>
</tr>
<tr>
<td></td>
<td>Added new combining algorithms.</td>
</tr>
<tr>
<td></td>
<td>Added security consideration about policy id collisions.</td>
</tr>
<tr>
<td></td>
<td>Added the <code>&lt;Advice&gt;</code> feature</td>
</tr>
<tr>
<td></td>
<td>Made obligations mandatory (per the 19th Dec 2008 decision of the TC)</td>
</tr>
<tr>
<td></td>
<td>Made obligations/advice available in rules</td>
</tr>
<tr>
<td>WD 09</td>
<td>Clarified wording about normative/informative in the combining algorithms section.</td>
</tr>
<tr>
<td></td>
<td>Fixed duplicate variable in comb.algs and cleaned up variable names.</td>
</tr>
<tr>
<td></td>
<td>Updated the schema to support the new multiple request scheme.</td>
</tr>
<tr>
<td>WD 10 19 Mar 2009</td>
<td>Fixed schema for <code>&lt;Request&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Fixed typos.</td>
</tr>
<tr>
<td></td>
<td>Added optional Category to <code>AttributeAssignments</code> in obligations/advice.</td>
</tr>
<tr>
<td>WD 11</td>
<td>Cleanups courtesy of John Tolbert.</td>
</tr>
<tr>
<td></td>
<td>Added Issuer XML attribute to <code>&lt;AttributeAssignment&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Fix the XPath expressions in the example policies and requests</td>
</tr>
<tr>
<td></td>
<td>Fix inconsistencies in the conformance tables.</td>
</tr>
<tr>
<td></td>
<td>Editorial cleanups.</td>
</tr>
<tr>
<td>WD 12 16 Nov 2009</td>
<td>(Now working draft after public review of CD 1)</td>
</tr>
<tr>
<td></td>
<td>Fix typos.</td>
</tr>
<tr>
<td></td>
<td>Allow element selection in attribute selector.</td>
</tr>
<tr>
<td></td>
<td>Improve consistency in the use of the terms obligation, advice, and advice/obligation</td>
</tr>
<tr>
<td></td>
<td>expressions and where they can appear.</td>
</tr>
<tr>
<td></td>
<td>Fixed inconsistency in PEP bias between sections 5.1 and 7.2.</td>
</tr>
<tr>
<td></td>
<td>Clarified text in overview of combining algorithms.</td>
</tr>
<tr>
<td></td>
<td>Relaxed restriction on matching in <code>xpath-node-</code></td>
</tr>
</tbody>
</table>
match function.
Remove note about XPath expert review.
Removed obsolete resource:xpath identifier.
Updated reference to XML spec.
Defined error behavior for string-substring and uri-substring functions.
Reversed the order of the arguments for the following functions: string-starts-with, uri-starts-with, string-ends-with, uri-ends-with, string-contains and uri-contains
Renamed functions:
- uri-starts-with to anyURI-starts-with
- uri-ends-with to anyURI-ends-with
- uri-contains to anyURI-contains
- uri-substring to anyURI-substring
Removed redundant occurrence indicators from RequestType.
Don't use "...:os" namespace in examples since this is still just "...:wd-12".
Added missing MustBePresent and Version XML attributes in example policies.
Added missing ReturnPolicyIdList and IncludeInResult XML attributes in example requests.
Clarified error behavior in obligation/advice expressions.
Allow bags in attribute assignment expressions.
Use the new daytimeduration and yearmonthduration identifiers consistently.

<table>
<thead>
<tr>
<th>WD 13</th>
<th>14 Dec 2009</th>
<th>Erik Rissanen</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Fix small inconsistency in number of arguments to the multiply function.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generalize higher order bag functions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Add ContextSelectorId to attribute selector.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use &lt;Policy(Set)IdReference&gt; in &lt;PolicyIdList&gt;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fix typos and formatting issues.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Make the conformance section clearly reference the functional requirements in the spec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conformance tests are no longer hosted by Sun.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WD 14</th>
<th>17 Dec 2009</th>
<th>Erik Rissanen</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Update acknowledgments</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WD 15</th>
<th></th>
<th>Erik Rissanen</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Replace DecisionCombiningAlgorithm with a simple Boolean for CombinedDecision.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Restrict &lt;Content&gt; to a single child element</td>
</tr>
</tbody>
</table>
and update the `<AttributeSelector>` and XPathExpression data type accordingly.

| WD 16 | 12 Jan 2010 | Erik Rissanen | Updated cross references  
Fix typos and minor inconsistencies.  
Simplify schema of `<PolicyIdentifierList>`  
Refactor some of the text to make it easier to understand.  
Update acknowledgments |
|-------|-------------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WD 17 | 8 Mar 2010  | Erik Rissanen | Updated cross references.  
Fixed OASIS style issues. |
| WD 18 | 23 Jun 2010 | Erik Rissanen | Fixed typos in examples.  
Fixed typos in schema fragments. |
| WD 19 | 14 April 2011 | Erik Rissanen | Updated function identifiers for new duration functions. Listed old identifiers as planned for deprecation.  
Added example for the X500Name-match function.  
Removed the (broken) Haskel definitions of the higher order functions.  
Clarified behavior of extended indeterminate in context of legacy combining algorithms or an Indeterminate target.  
Removed `<Condition>` from the expression substitution group.  
Specified argument order for subtract, divide and mod functions.  
Specified datatype to string conversion form to those functions which depend on it.  
Specified Indeterminate value for functions which convert strings to another datatype if the string is not a valid lexigraphical representation of the datatype.  
Removed higher order functions for ip address and dns name. |
| WD 20 | 24 May 2011 | Erik Rissanen | Fixed typo between “first” and “second” arguments in rfc822Name-match function.  
Removed duplicate word “string” in a couple of places.  
Improved and reorganized the text about extended indeterminate processing and Rule/Policy/PolicySet evaluation.  
Explicitly stated that an implementation is conformant regardless of its internal workings as longs as the external result is the same as in this specification.  
Changed requirement on Indeterminate behavior at the top of section A.3 which |
conflicted with Boolean function definitions.

| WD 21 | 28 Jun 2011 | Erik Rissanen | Redefined combining algorithms so they explicitly evaluate their children in the pseudocode. Changed wording in 7.12 and 7.13 to clarify that the combining algorithm applies to the children only, not the target. Removed wording in attribute category definitions about the attribute categories appearing multiple times since bags of bags are not supported. Fixed many small typos. Clarified wording about combiner parameters. |
| WD 23 | 19 Mar 2012 | Erik Rissanen | Refomat references to OASIS specs. Define how XACML identifiers are matched. Do not highlight “actions” with the glossary term meaning in section 2.12. Fix minor typos. Make a reference to the full list of combining algorithms from the introduction. Clarified behavior of the context handler. Renamed higher order functions which were generalized in an earlier working draft. Add missing line in schema fragment for <AttributeDesignator> Removed reference to reuse of rules in section 2.2. There is no mechanism in XACML itself to re-use rules, though of course a tool could create copies as a form of “re-use”. |