



---

# Attribute-Centric Authorization Language (ACAL) Version 1.0

## Committee Specification Draft 01

18 February 2026

### This version

- <https://docs.oasis-open.org/xacml/acal/acal/core/v1.0/csd01/acal-core-v1.0-csd01.html> (Authoritative)
- <https://docs.oasis-open.org/xacml/acal/acal/core/v1.0/csd01/acal-core-v1.0-csd01.pdf>
- <https://docs.oasis-open.org/xacml/acal/acal/core/v1.0/csd01/acal-core-v1.0-csd01.md>

### Previous version

N/A

### Latest version

- <https://docs.oasis-open.org/xacml/acal/acal/core/v1.0/acal-core-v1.0.html> (Authoritative)
- <https://docs.oasis-open.org/xacml/acal/acal/core/v1.0/acal-core-v1.0.pdf>
- <https://docs.oasis-open.org/xacml/acal/acal/core/v1.0/acal-core-v1.0.md>

### Technical Committee

OASIS eXtensible Access Control Markup Language (XACML) TC

### Chairs

- Bill Parducci ([bill@parducci.net](mailto:bill@parducci.net)), Individual

### Secretaries

- Bill Parducci ([bill@parducci.net](mailto:bill@parducci.net)), Individual

### Editors

- Steven Legg ([steven.legg@viewds.com](mailto:steven.legg@viewds.com)), ViewDS Identity Solutions
- Cyril Dangerville ([cyril.dangerville@thalesgroup.com](mailto:cyril.dangerville@thalesgroup.com)), THALES

### Abstract

This specification defines Version 1.0 of the Attribute-Centric Authorization Language's core model. This model is essentially an XML-agnostic evolution of XACML v3.0 that provides a common foundation for other concrete representation formats to come (e.g. JSON).

### Citation Format

When referencing this document, the following citation format should be used:

**[ACAL-Core-1.0]** *Attribute-Centric Authorization Language (ACAL) Version 1.0*. Edited by Steven Legg and Cyril Dangerville. 18 February 2026. OASIS Committee Specification Draft 01. <https://docs.oasis-open.org/xacml/acal/acal/core/v1.0/csd01/acal-core-v1.0-csd01.html> . Latest version: <https://docs.oasis-open.org/xacml/acal/acal/core/v1.0/acal-core-v1.0.html>

## Related Work

This document replaces or supersedes:

- *eXtensible Access Control Markup Language (XACML) Version 3.0 Plus Errata 01.* Edited by Erik Rissanen. OASIS Standard incorporating Approved Errata. <https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html>.

This document is related to:

- *JACAL: The JSON representation of the Attribute-Centric Authorization Language (ACAL) Version 1.0.*
- *XACML: The XML representation of the Attribute-Centric Authorization Language (ACAL) Version 4.0.*
- *YACAL: The YAML representation of the Attribute-Centric Authorization Language (ACAL) Version 1.0.*

## License, Document Status, and Notices

Copyright © OASIS Open 2026. All Rights Reserved. For license and copyright information, and complete status, please see Annex A which contains the License, Document Status and Notices.

---

## Table of Contents

- 1 Scope
- 2 Definitions and Acronyms
  - 2.1 Definitions
    - 2.1.1 Terms Defined Elsewhere
    - 2.1.2 Terms Defined in this Document
    - 2.1.3 Related terms
  - 2.2 Abbreviations and Acronyms
- 3 Document Conventions
  - 3.1 Key Words
  - 3.2 Typographical Conventions
- 4 Introduction (non-normative)
  - 4.1 Requirements
  - 4.2 Rule and Policy Combining
  - 4.3 Combining Algorithms
  - 4.4 Multiple Subjects
  - 4.5 Policies Based on Subject and Resource Attributes
  - 4.6 Multi-Valued Attributes
  - 4.7 Policies Based on Resource Contents
  - 4.8 Operators
  - 4.9 Policy Distribution
  - 4.10 Policy Indexing
  - 4.11 Abstraction Layer
  - 4.12 Actions Performed in Conjunction with Enforcement
  - 4.13 Supplemental Information About a Decision
  - 4.14 Changes From the Previous Version
- 5 Models (non-normative)
  - 5.1 Data-Flow Model
  - 5.2 ACAL Context
  - 5.3 Policy Language Model
    - 5.3.1 Rule
      - 5.3.1.1 Condition
      - 5.3.1.2 Effect
      - 5.3.1.4 Notice Expressions
    - 5.3.2 Policy
      - 5.3.2.1 Policy Target
      - 5.3.2.2 Combining Algorithm
      - 5.3.2.3 Notice Expressions
- 6 Examples (non-normative)
  - 6.1 Example One
    - 6.1.1 Example Policy

- 6.1.2 Example Decision Request
  - 6.1.3 Example Decision Response
- 6.2 Example Two
  - 6.2.1 Example Plain-Language Rules
  - 6.2.2 Example Short Identifier Set
  - 6.2.3 Example Decision Request
  - 6.2.4 Example ACAL Rule Instances
    - 6.2.4.1 Rule 1
    - 6.2.4.2 Rule 2
    - 6.2.4.3 Rule 3
    - 6.2.4.4 Rule 4
    - 6.2.4.5 Example Policy with Nested Policies
- 7 Structures
  - 7.1 Introduction
  - 7.1.1 Object Type
    - 7.1.1.1 Object constraints
      - 7.1.1.1.1 Property-level constraints
        - 7.1.1.1.1.1 UML native constraints
        - 7.1.1.1.1.2 OCL constraints
      - 7.1.1.1.2 Object-level constraints
  - 7.1.2 Simple Types
    - 7.1.2.1 Standard UML primitive types (String, Boolean, Integer, Real)
    - 7.1.2.2 ACAL-defined UML stereotype(s) used in new primitive types
    - 7.1.2.3 ACAL-defined simple types
      - 7.1.2.3.1 URI
      - 7.1.2.3.2 NonNegativeInteger
      - 7.1.2.3.3 Double
      - 7.1.2.3.4 VersionType
      - 7.1.2.3.5 VersionMatchType
      - 7.1.2.3.6 ShortIdNameType
      - 7.1.2.3.7 ShortIdValueType
      - 7.1.2.3.8 IdentifierType
      - 7.1.2.3.9 LocalIdentifierType
      - 7.1.2.3.10 AttributeSelectorPathType
      - 7.1.2.3.11 Name
      - 7.1.2.3.12 EffectType
      - 7.1.2.3.13 DecisionType
      - 7.1.2.3.14 MediaType (optional)
      - 7.1.2.3.15 ContentEncodingType (optional)
  - 7.1.3 Relationship to Concrete Representations
  - 7.2 ShortIdSetType
  - 7.3 ShortIdType
  - 7.4 PolicyType
  - 7.5 PolicyDefaultsType (optional)
  - 7.6 ParameterType
  - 7.7 BooleanExpressionType
  - 7.8 IdReferenceType
  - 7.9 ExactMatchIdReferenceType (optional)
  - 7.10 PatternMatchIdReferenceType
  - 7.11 PolicyReferenceType
  - 7.12 RuleType
  - 7.13 VariableDefinitionType
  - 7.13b SharedVariableDefinitionType
  - 7.14 ExpressionType
  - 7.15 ApplyType
  - 7.16 FunctionType
  - 7.17 NamedAttributeDesignatorType
  - 7.18 AttributeDesignatorType
  - 7.19 EntityAttributeDesignatorType
  - 7.20 BaseAttributeSelectorType (optional)
  - 7.21 AttributeSelectorType (optional)

- 7.22 EntityAttributeSelectorType (optional)
- 7.23 ValueType
- 7.24 VariableReferenceType
- 7.24b SharedVariableReferenceType
- 7.25 QuantifiedExpressionType
  - 7.25.1 ForAny Expression
  - 7.25.2 ForAll Expression
  - 7.25.3 Map Expression
  - 7.25.4 Select Expression
- 7.26 NoticeType
- 7.27 AttributeType
- 7.28 AttributeAssignmentType
- 7.29 NoticeExpressionType
- 7.30 AttributeAssignmentExpressionType
- 7.31 RequestType
- 7.32 RequestDefaultsType (optional)
- 7.33 RequestEntityType
- 7.34 ContentType (optional)
- 7.35 RequestAttributeType
- 7.36 ResponseType
- 7.37 ResultType
- 7.38 MultiRequestsType (optional)
- 7.39 RequestReferenceType (optional)
- 7.40 RequestEntityReferenceType (optional)
- 7.41 StatusType
- 7.42 StatusCodeType
- 7.43 StatusDetailType (optional)
- 7.44 MissingAttributeDetailType
- 7.45 ResultEntityType
- 7.46 EntityType
- 7.47 BundleType
- 8 Functional Requirements
  - 8.1 Unicode Issues
    - 8.1.1 Normalization
    - 8.1.2 Version of Unicode
  - 8.2 Policy Enforcement Point
    - 8.2.1 Base PEP
    - 8.2.2 Deny-Biased PEP
    - 8.2.3 Permit-Biased PEP
  - 8.3 Identifier Evaluation
    - 8.3.1 Identifier Examples (non-normative)
    - 8.3.2 Short Identifier Set Usage (non-normative)
  - 8.4 Attribute Evaluation
    - 8.4.1 Structured Attributes
    - 8.4.2 Attribute Bags
    - 8.4.3 Multivalued Attributes
    - 8.4.4 Attribute Matching
    - 8.4.5 Attribute Retrieval
    - 8.4.6 Environment Attributes
    - 8.4.7 Selector Evaluation
  - 8.5 Expression Evaluation
  - 8.6 Arithmetic Evaluation
  - 8.7 Target Evaluation
  - 8.8 VariableReference Evaluation
  - 8.9 Condition Evaluation
  - 8.10 Extended Indeterminate
  - 8.11 Rule Evaluation
  - 8.12 Policy Evaluation
  - 8.13 PolicyReference Evaluation
  - 8.14 Hierarchical Resources
  - 8.15 Authorization Decision

- 8.16 Notices
- 8.17 Exception Handling
  - 8.17.1 Unsupported Functionality
  - 8.17.2 Syntax and Type Errors
  - 8.17.3 Missing Attributes
- 8.18 Identifier Equality
- 8.19 Short Identifiers in Responses
- 9 ACAL Extensibility Points (non-normative)
  - 9.1 Extensible Properties
  - 9.2 Structured Attributes
- 10 Safety, Security, and Data Protection Considerations
  - 10.1 Threat Model
    - 10.1.1 Unauthorized Disclosure
    - 10.1.2 Message Replay
    - 10.1.3 Message Insertion
    - 10.1.4 Message Deletion
    - 10.1.5 Message Modification
    - 10.1.6 NotApplicable Results
    - 10.1.7 Negative Rules
    - 10.1.8 Denial of Service
  - 10.2 Safeguards
    - 10.2.1 Authentication
    - 10.2.2 Policy Administration
    - 10.2.3 Confidentiality
      - 10.2.3.1 Communication Confidentiality
      - 10.2.3.2 Statement Level Confidentiality
    - 10.2.4 Policy Integrity
    - 10.2.5 Policy Identifiers
    - 10.2.6 Trust Model
    - 10.2.7 Privacy
  - 10.3 Unicode Security Issues
  - 10.4 Identifier Equality
- 11 Conformance
  - 11.1 Introduction
  - 11.2 Conformance Tables
    - 11.2.1 Object Types
    - 11.2.2 Identifier Prefixes
    - 11.2.3 Algorithms
    - 11.2.4 Status Codes
    - 11.2.5 Environment Attributes
    - 11.2.6 Attributes and Categories
    - 11.2.7 Data Types
    - 11.2.8 Functions
- Annex A License, Document Status and Notices
  - A.1 Document Status
  - A.2 License and Notices
- Annex B References
  - B.1 Normative References
  - B.2 Informative References
- Annex C Data Types and Functions
  - C.1 Introduction
  - C.2 Data Types
    - C.2.1 X.500 Directory Name
    - C.2.2 RFC 822 Name
    - C.2.3 IP Address
    - C.2.4 DNS Name
    - C.2.6 Entity
  - C.3 Functions
    - C.3.1 Equality Predicates
    - C.3.2 Arithmetic Functions
    - C.3.3 String Conversion Functions

- C.3.4 Numeric Data Type Conversion Functions
  - C.3.5 Logical Functions
  - C.3.6 Numeric Comparison Functions
  - C.3.7 Date and Time Arithmetic Functions
  - C.3.8 Non-numeric Comparison Functions
  - C.3.9 String Functions
  - C.3.10 Bag Functions
  - C.3.11 Set Functions
  - C.3.12 Higher-order Bag Functions
  - C.3.13 Regular-Expression-Based Functions
  - C.3.14 Aggregate Functions
  - C.3.15 Special Match Functions
  - C.3.16 Other Functions
  - C.3.18 Extension Functions and Data Types
  - Annex D ACAL Identifiers
    - D.1 ACAL Namespaces
    - D.2 Attribute Categories
    - D.3 Data Types
    - D.4 Subject Attributes
    - D.5 Resource Attributes
    - D.6 Action Attributes
    - D.7 Environment Attributes
    - D.8 Status Codes
    - D.9 Combining Algorithms
    - D.10 Content Types
    - D.11 Content Encodings
  - Annex E Combining Algorithms
    - E.1 Extended Indeterminate Values
    - E.2 Deny Overrides
    - E.3 Ordered Deny Overrides
    - E.4 Permit Overrides
    - E.5 Ordered Permit Overrides
    - E.6 Deny Unless Permit
    - E.7 Permit Unless Deny
    - E.8 First Applicable
  - Annex F How to generate HTML and PDF Versions
  - Appendix 1 Acknowledgments
    - Leadership
    - Special Thanks
    - Participants
  - Appendix 2 Changes From Previous Version
    - Revision History
- 

## 1 Scope

This specification defines a common language and processing model for:

1. Attribute-Based Access Control policies;
2. Access Control Decision Request and Response information exchanged between an Access Control Enforcement entity and an Access Control Decision entity.

This common language intends to be technology-agnostic enough to address enterprise-wide access control requirements, i.e. for any kind of enterprise resource that requires protection against unauthorized access.

Concrete representations (data formats) are to be provided as separate specifications and therefore out of scope of this document.

---

## 2 Definitions and Acronyms

### 2.1 Definitions

#### 2.1.1 Terms Defined Elsewhere

This document uses the following terms defined elsewhere:

##### Bag (a.k.a. *multiset*)

[NISTIR8318] An unordered collection of values, in which there may be duplicate values.

##### Type Unification

[Hancock] 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].

#### 2.1.2 Terms Defined in this Document

This document defines the following terms:

##### Access

Performing an *action*.

##### Access control

Controlling *access* in accordance with a *policy*.

##### Action

An operation on a *resource*.

##### Advice

A *notice* providing supplementary information to the *PEP* about a *decision* from the *PDP*.

##### Applicable policy

The set of *policies* that governs *access* for a specific *decision request*.

##### Attribute

Characteristic of a *subject*, *resource*, *action* or *environment* that may be referenced in a *predicate* (see also – *named attribute*).

##### Attribute issuer

Source of a *named attribute*, e.g. a particular PIP or the PEP itself.

##### 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 list of *notices*.

##### Combining algorithm

The procedure for combining the *decision* and *notices* from multiple *policies* and *rules*.

##### 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* or *policy*.

**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 8.18.

**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.

**Notice**

An additional information item provided to the **PEP** alongside a *decision* from the **PDP**. A *notice* is either an *obligation* or *advice*. *Notices* are potentially generated by *rules* and *policies* during their evaluation by the **PDP**.

**Obligation**

A *notice* specifying an operation that should be performed by the **PEP** in conjunction with the enforcement of an *authorization decision*.

**Policy**

A set of *rules*, other *policies*, an identifier for the *combining algorithm* and (optionally) a list of *notice* expressions. May be a component of another *policy*.

**Policy issuer**

A set of *attributes* describing the source of a *policy*.

**Predicate**

A statement about *attributes* whose truth can be evaluated.

**Resource**

Data, service or system component that a subject requests access to.

**Rule**

An *effect*, a *condition* and (optionally) a list of *notice* expressions. A component of a *policy*.

**Short identifier**

A binding of a simple alias name to a URI or a part thereof.

**Subject**

An actor that requests access to a resource and whose *attributes* may be referenced by a *predicate*.

**Target**

An element of an XACML *policy* which matches specified values of *resource*, *subject*, *environment*, *action*, or other custom *attributes* against those provided in the request *context* as a part of the process of determining whether the *policy* is applicable to the current decision.

### 2.1.3 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.

Access requestors and initiators are covered by the term *subject*.

## 2.2 Abbreviations and Acronyms

This document uses the following abbreviations and acronyms:

**Iff** Term used in logic and mathematic fields, that means *if and only if*.

**OCL (Object Constraint Language)**

[OCL] is an international OMG and ISO standard specifying a formal language to describe expressions on UML models, UML constraints in particular. These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model.

**PAP (Policy Administration Point)**

The system entity that creates a *policy*.

**PDP (Policy Decision Point)**

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].

**PEP (Policy Enforcement Point)**

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].

**PIP (Policy Information Point)**

The system entity that acts as a source of *attribute* values.

**UML (Unified Modelling Language)**

[UML] is an international OMG (Object Management Group) and ISO standard defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed software systems.

---

## 3 Document Conventions

### 3.1 Key Words

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

### 3.2 Typographical Conventions

None.

---

## 4 Introduction (non-normative)

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

The security policy of a large enterprise has many elements and many points of enforcement. Elements of policy may be managed by the Information Systems department, by Human Resources, by the Legal department and by the Finance department. And the policy may be enforced by the extranet, mail, WAN, and remote-access systems; platforms which inherently implement a permissive security policy. The current practice is to manage the configuration of each point of enforcement independently in order to implement the security policy as

accurately as possible. Consequently, it is an expensive and unreliable proposition to modify the security policy. 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.

This specification defines the concepts and processing models for such a common language but leaves the concrete syntactic representation to separate specifications. ACAL offers a choice of syntaxes and implementations are free to choose any one or more of these syntaxes to support.

## 4.1 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 that applies to a particular decision request.
- To provide a method for flexible definition of the procedure by which rules and policies are combined.
- To provide a method for dealing with multiple subjects acting in different capacities.
- To provide a method for basing an authorization decision on attributes of the subject and resource.
- To provide a method for dealing with multi-valued attributes.
- To provide a method for basing an authorization decision on the contents of an information resource.
- To provide a set of logical and mathematical operators on attributes of the subject, resource and environment.
- To provide a method for handling a distributed set of policy components, while abstracting the method for locating, retrieving and authenticating the policy components.
- To provide a method for rapidly identifying the policy that applies to a given action, based upon the values of attributes of the subjects, resource and action.
- To provide an abstraction-layer that insulates the policy-writer from the details of the application environment.
- To provide a method for specifying a set of actions that must be performed in conjunction with policy enforcement.

The motivation behind ACAL is to express these well-established ideas in the field of access control policy. The ACAL solutions for each of these requirements are discussed in the following sections.

## 4.2 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.

A rule 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 ACAL PAP, where it may form the basic unit of management.

A policy contains a set of rules and/or other policies 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. It is also 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.

## 4.3 Combining Algorithms

ACAL defines a number of combining algorithms that each define a procedure for arriving at an authorization decision given the individual results of evaluation of a set of rules and/or policies. Some examples of standard combining algorithms are (see Annex E for a full list of standard combining algorithms):

- **deny-overrides** (ordered and unordered),
- **permit-overrides** (ordered and unordered) and
- **first-applicable**.

In the case of the **deny-overrides** algorithm, if a single rule or policy is encountered that evaluates to **Deny**, then, regardless of the evaluation result of the other rules or policies in the applicable policy, the combined result is **Deny**. (See also the similar **permit-unless-deny** algorithm in Annex E.)

Likewise, in the case of the **permit-overrides** algorithm, if a single **Permit** result is encountered, then the combined result is **Permit**. (See also the similar **deny-unless-permit** algorithm in Annex E.)

In the case of the **first-applicable** combining algorithm, the combined result is the same as the result of the first rule or policy in the list of rules and policies that is applicable to the decision request.

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

## 4.4 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, ACAL 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.

## 4.5 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]. ACAL provides facilities to support this approach. Attributes of subjects contained in the request context may be identified by an attribute designator. Alternatively, an attribute selector may contain a Path expression (e.g. Path/JSONPath) over the **ContentType** object of the subject to identify a particular subject attribute value by its location in the context (see Section 4.11 for an explanation of context).

ACAL 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. ACAL provides facilities to support this approach. Attributes of the resource may be identified by an attribute designator. Alternatively, an attribute selector may contain a Path expression (e.g. XPath/JSONPath) over the **ContentType** object of the resource to identify a particular resource attribute value by its location in the context.

## 4.6 Multi-Valued Attributes

The most common techniques for communicating attributes (LDAP, XPath, JSONPath, SAML, etc.) support multiple values per attribute. Therefore, when an ACAL 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 ACAL rule is satisfied if any one of the attribute values meets the criteria expressed in the rule.

ACAL 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 Annex C.3).

## 4.7 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.

ACAL provides facilities for doing this when the information resource can be represented as a structured document like XML or JSON. The `AttributeSelectorType` object may contain a content-specific Path expression (e.g. XPath) over the `ContentType` object 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 structured document like XML or JSON, specified attributes of the resource can be referenced, as described in Section 4.5.

## 4.8 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.

ACAL 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 `ApplyType` objects. An `ApplyType` object has a property called `FunctionId` that identifies the function to be applied to the contents of the object, i.e., the arguments of the function. 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 ACAL method of representing functions borrows from MathML [MathML]. (For XPath functions, see the XPath Profile of ACAL.)

## 4.9 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. ACAL policy statements may be distributed in any one of a number of ways. But, ACAL 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 that the policy is applicable to the decision request that it is processing.

Policies may be attached to the information resources to which they apply, as described by Perritt [Perritt93]. Alternatively, policies 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.

## 4.10 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 policy target in ACAL.

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 of the retrieved policy statements as defined by the ACAL specification.
2. Alternatively, the PDP may be loaded with all available policies and evaluate their targets in the context of a particular decision request, in order to identify the policies that are applicable to that request.

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

## 4.11 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 ACAL PDP. This canonical form is called the ACAL context. Its abstract syntax is defined here in UML and concrete syntaxes and representation formats in separate profiles of ACAL (XML, JSON, etc.).

Naturally, ACAL-conformant PEPs may issue requests and receive responses in the form of an ACAL 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 ACAL 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.

## 4.12 Actions Performed in Conjunction with Enforcement

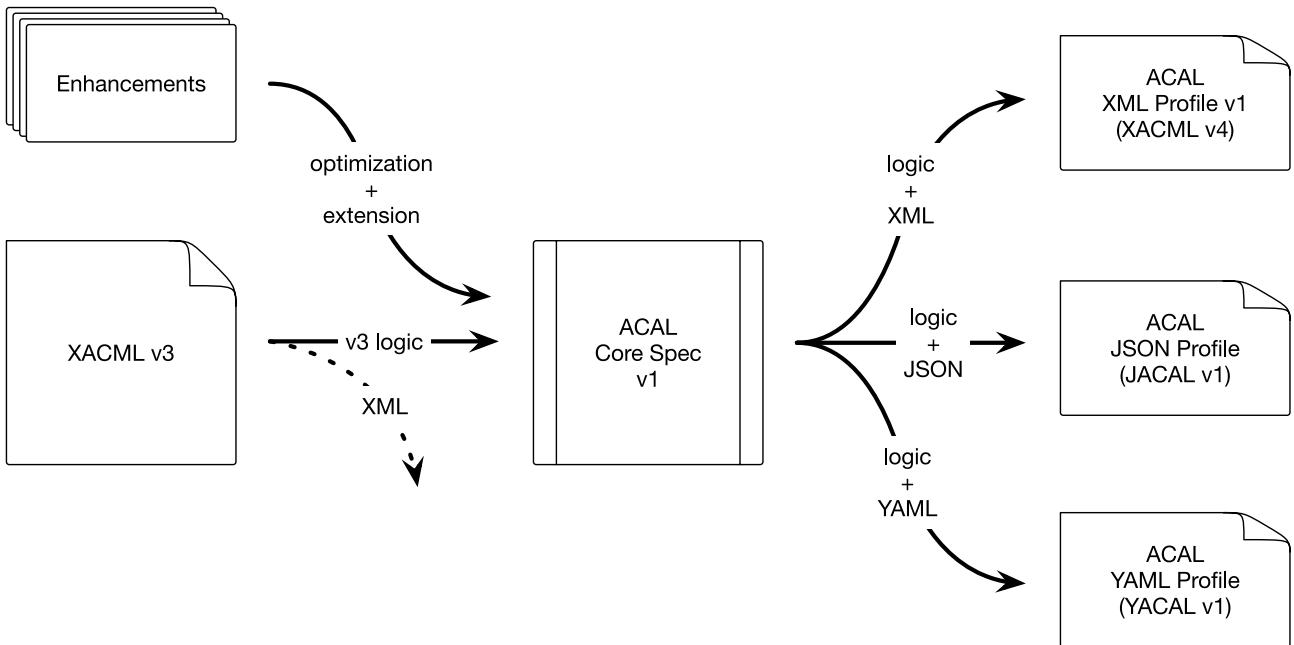
In many applications, policies specify actions that MUST be performed, either instead of, or in addition to, actions that MAY be performed. This idea was described by Sloman [Sloman94]. ACAL provides facilities to specify actions that MUST be performed in conjunction with policy evaluation through obligation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

## 4.13 Supplemental Information About a Decision

In some applications it is helpful to specify supplemental information about a decision. ACAL provides facilities to specify supplemental information about a decision through advice notices. Such advice may be safely ignored by the PEP.

## 4.14 Changes From the Previous Version

As illustrated in the **Figure 4-1** below, compared to XACML v3, this ACAL specification is a generalization of XACML model (syntax and processing model), aiming to be XML-agnostic, therefore laying the ground for other (non-XML) concrete representations to be added. In addition, ACAL benefits from extra enhancements including model simplifications and optimizations as well as new features. For purposes of simplifying this specification, all XPath features - originally from XACML - are kept in a separate ACAL Profile.



**Fig. 4-1.** Evolution from XACML v3 to ACAL

The detailed list of changes from the previous version and any revision history can be found in Appendix 2.

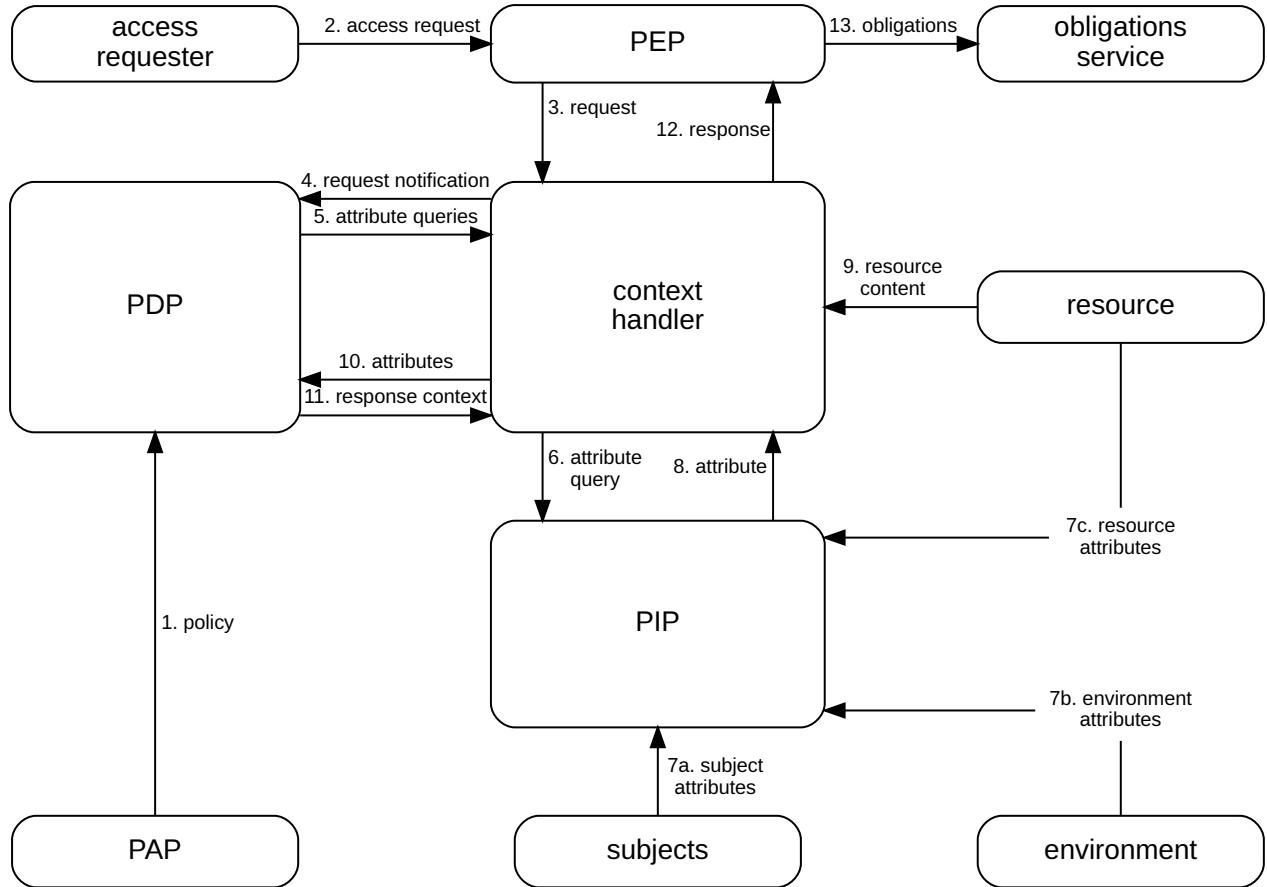
---

## 5 Models (non-normative)

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

### 5.1 Data-Flow Model

The major actors in the ACAL domain are shown in the data-flow diagram of **Figure 5-1**.



**Fig. 5-1.** Data-Flow Diagram

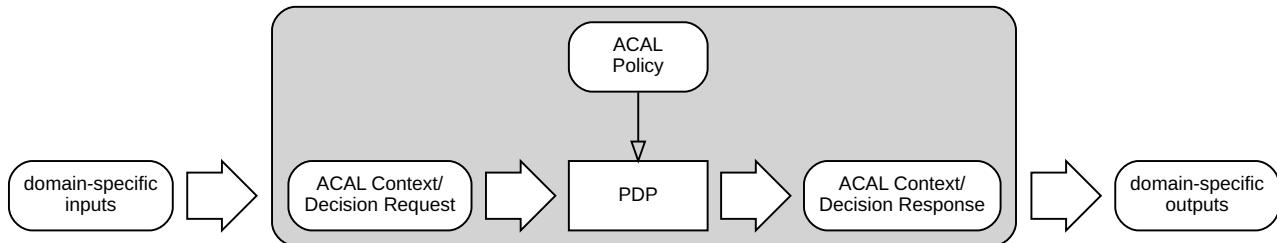
Note: some of the data-flows shown in the diagram may be facilitated by a repository. For instance, the communications between the context handler and the PIP or the communications between the PDP and the PAP may be facilitated by a repository. The ACAL specification is not intended to place restrictions 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 make them available to the PDP. These policies represent the complete policy for a specified target.
2. The access requester sends a request for access to the PEP.
3. The PEP sends the request for access to the context handler in its native request format, optionally including attributes of the subjects, resource, action, environment and other categories.
4. The context handler constructs an ACAL 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 obligation notices.
14. (Not shown) If access is permitted, then the PEP permits access to the resource; otherwise, it denies access.

## 5.2 ACAL Context

ACAL is intended to be suitable for a variety of application environments. The core language is insulated from the application environment by the ACAL context, as shown in **Figure 5-2**, in which the scope of the ACAL specification is indicated by the shaded area. In the Figure 5-2, the ACAL context represents the inputs and outputs of the PDP decision. An ACAL Policy references attributes in the ACAL context via content-specific path expressions (e.g. XPath for XML content, JSONPath for JSON content) over the `ContentType` objects of the context, or *named* 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 ACAL context. How this is achieved is outside the scope of the ACAL specification. In some cases, such as SAML, this conversion may be accomplished in an automated way through the use of an XSLT transformation.



**Figure 5-2.** ACAL Context

Note: The PDP is not required to operate directly on the ACAL 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 Annex D.2 for suggested attribute categories.

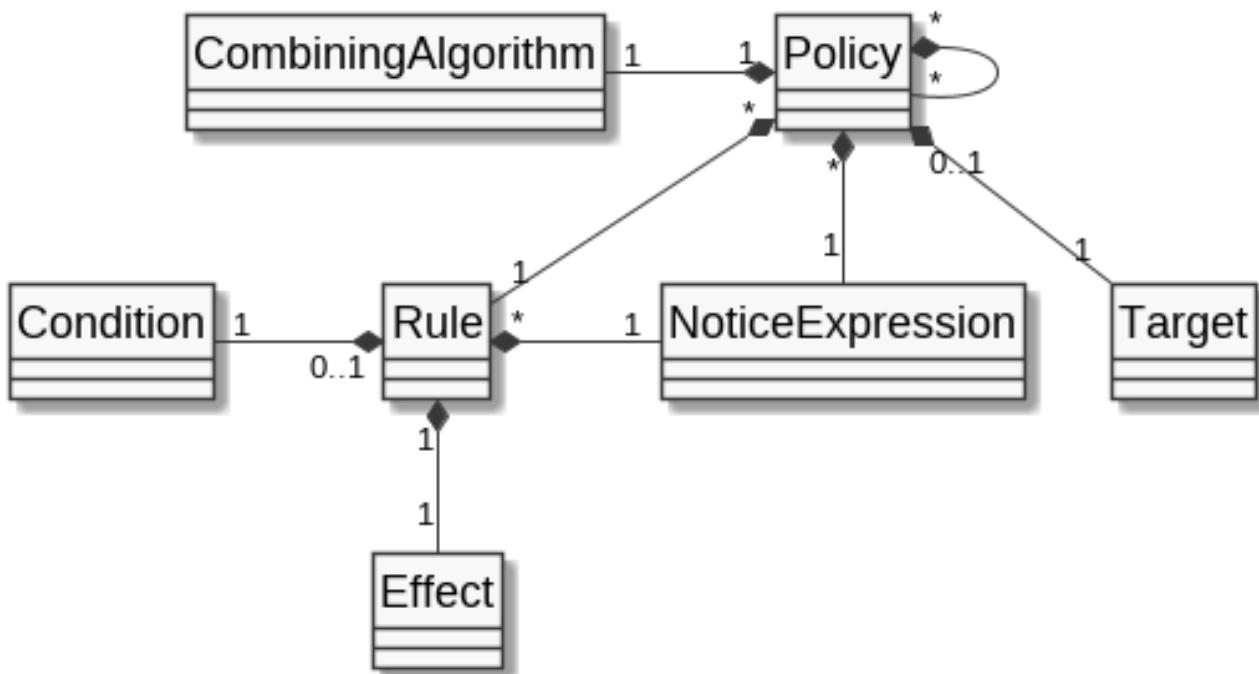
See Section 8.4.5 for a more detailed discussion of the request context.

## 5.3 Policy Language Model

The policy language model is shown in **Figure 5-3**. The main components of the model are:

- Rule; and
- Policy.

These are described in the following sub-sections.



### Figure 5-3. Policy Language Model

#### 5.3.1 Rule

A rule is the most elementary unit of policy. It may exist in isolation only within one of the major actors of the ACAL 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:

- an effect,
- a condition, and
- notice expressions.

These are discussed in the following sub-sections.

**5.3.1.1 Condition** The condition defines the set of requests to which the rule applies, expressed as a logical expression on attributes in the request. If a rule is intended to apply to all requests to which its parent policy is applicable, the condition may be omitted.

**Hierarchical Logic** When evaluating conditions against structured name-forms (such as directory names, file paths, or nested object keys), the matching logic MUST distinguish between a discrete node and a subtree.

- **Subjects:** Non-leaf subject names generally identify the set of subjects subordinate to that node.
- **Resources:** A resource node may represent either a discrete entity or a container for a subtree of entities.

The interpretation of these structures is influenced by the match function selected and the hierarchy's specific delimiter (e.g., /, \, or .). The **Hierarchical Resource Profile** provides the specific functions and logic required to handle these variations across different data representations.

**Evaluation** The PDP verifies that the matches defined by the condition are satisfied by the attributes in the request context. For the formal processing model, see **Section 9, "Condition Evaluation"**.

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

**5.3.1.4 Notice Expressions** Notice expressions may be added by the writer of the rule.

When a PDP evaluates a rule containing notice expressions, it evaluates the notice expressions into notices and returns certain of those notices to the PEP in the response context. Section 8.16 explains which notices are to be returned.

In contrast to obligation notices, advice notices may be safely ignored by the PEP.

#### 5.3.2 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 combining algorithm identifier;
- a set of rules and policies and
- notice expressions.

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

**5.3.2.1 Policy Target** An ACAL policy contains a target that specifies the set of requests to which it applies. The target of a policy may be:

1. Either declared by the writer of the policy. In this case, any component rules in the policy that have the same condition as the target may omit the condition. Such rules inherit the target of the policy in which they are contained.
2. Or not, in which case it may be calculated from the targets and conditions of the policies and rules (respectively) that it contains. A system entity that calculates a target in this way is not defined by ACAL, but there are two logical methods that might be used:

1. First method: the target of the outer policy (the "outer component") is calculated as the union of all the targets of the referenced policies and the conditions of the referenced rules (the "inner components").
2. Second method: the target of the outer component is calculated as the intersection of all the targets and conditions of the inner components. The results of evaluation in each case will be very different: in the first case, the target of the outer component makes it applicable to any decision request that matches the target or condition of at least one inner component; in the second case, the target of the outer component makes it applicable only to decision requests that match the target or condition of every inner component.

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

See Annex E for definitions of the standard combining algorithms.

**5.3.2.3 Notice Expressions** The writer of a policy may add notice expressions to the policy, in addition to those contained in the component rules and policies.

When a PDP evaluates a policy containing notice expressions, it evaluates the notice expressions into notices and returns certain of those notices to the PEP in the response context. Section 8.16 explains which notices are to be returned. In contrast to obligation notices, advice notices may be safely ignored by the PEP.

---

## 6 Examples (non-normative)

This section contains two examples of the use of ACAL for illustrative purposes. The first example is a relatively simple one to illustrate the use of rules, conditions, matching functions and subject attributes. The second example additionally illustrates the use of the combining algorithm and notices.

### 6.1 Example One

#### 6.1.1 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` domain is allowed to perform any action on any resource.*

The following XACML policy, respectively in XML and JSON, contains a rule instance expressing this access control policy (the numbers in square brackets on the left-hand side are for referencing purposes and are not part of the policy in either representation):

```
[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <Policy
[03]   xmlns="urn:oasis:names:tc:xacml:4.0:core:schema"
[04]   PolicyId="urn:oasis:names:tc:acal:1.0:example:SimplePolicy1"
[05]   Version="1.0"
[06]   CombiningAlgId="deny-overrides">
[07]   <ShortIdSetReference>urn:oasis:names:tc:acal:1.0:core:identifiers</ShortIdSetReference>
[08]   <Description>Medi Corp access control policy.</Description>
[09]   <Rule
[10]     Id="Rule1"
[11]     Effect="Permit">
[12]     <Description>Any subject with an e-mail name in the med.example.com domain can perform any action on any resource.</Description>
[13]     <Condition>
[14]       <Apply
[15]         FunctionId="any-of">
[16]           <Function Id="rfc822Name-match"/>
[17]           <AttributeDesignator
[18]             Category="access-subject"
[19]             AttributeId="subject-id"
[20]             DataType="rfc822Name"/>
[21]           <Value DataType="string">med.example.com</Value>
[22]         </Apply>
[23]       </Condition>
[24]     </Rule>
[25]   </Policy>
```

```

[35]  </Rule>
[37] </Policy>

[02] {
[04]   "PolicyId": "urn:oasis:names:tc:acal:1.0:example:SimplePolicy1",
[05]   "Version": "1.0",
[06]   "CombiningAlgId": "deny-overrides",
[07]   "ShortIdSetReference": ["urn:oasis:names:tc:acal:1.0:core:identifiers"],
[08]   "Description": "Medi Corp access control policy.",
[09]   "CombinerInput": [
[10]     "Rule": {
[11]       "Id": "Rule1",
[12]       "Effect": "Permit",
[13]       "Description": "Any subject with an e-mail name in the med.example.com domain can perform any action on any resource."
[14]       "Condition": {
[15]         "Apply": {
[16]           "FunctionId": "any-of",
[17]           "Expression": [
[18]             "Function": {
[19]               "Id": "rfc822Name-match"
[20]             }
[21]           },
[22]           "AttributeDesignator": {
[23]             "Category": "access-subject",
[24]             "AttributeId": "subject-id",
[25]             "DataType": "rfc822Name"
[26]           }
[27]         },
[28]         "Value": {
[29]           "DataType": "string",
[30]           "Value": "med.example.com"
[31]         }
[32]       }
[33]     }
[34]   }
[35] }
[36] ]
[37] }

```

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

[02] The beginning of the ACAL Policy.

[03] An XML namespace declaration.

[04] An identifier for this policy. Different policy instances with the same identifier must have different version numbers.

[05] The version of this instance of the policy. The combination of identifier and version has to be unique for a given policy instance so that there is no ambiguity if one policy is referenced from another policy.

[07] A reference to a short identifier set imported into the policy. The names of the short identifiers in the set are available to use in this policy.

[06] The algorithm that will be used to combine the results of the various rules and policies that may be contained in the policy. In this case the algorithm is nominated with the short identifier name **deny-overrides** which evaluates to **urn:oasis:names:tc:acal:1.0:combining-algorithm:deny-overrides** using the imported short identifier set. This combining algorithm specifies that, if any rule evaluates to **Deny**, then the policy must return **Deny**; otherwise, if any rule evaluates to **Permit**, then the policy must return **Permit**. The 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.

[08] An optional text description of the policy.

This policy does not specify a target so the policy is applicable to any decision request.

[10] The beginning of the one and only rule in this simple policy.

[11] An identifier for the rule. Each rule in a policy must have a distinct identifier.

[12] The result the rule will produce if its condition is satisfied. 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 the rule's condition is not satisfied, then the rule returns a result of **NotApplicable**. If an error occurs when evaluating the condition, then the rule returns a result of

**Indeterminate.** As mentioned above, the combining algorithm for the policy specifies how various rule results are combined into a single policy result.

[13] An optional text description of the rule.

[14] - [34] The condition of the rule. The condition determines the decision requests to which this rule applies.

[15] - [33] The expression that defines the condition to be satisfied, in this case a function is applied.

[16] The **FunctionId** property specifies the function to be applied. In this case, the function is nominated with the short identifier name **any-of**, which evaluates to **urn:oasis:names:tc:acal:1.0:function:any-of** using the imported short identifier set. This function compares an attribute value to each of the attribute values in a bag according to a matching function that is specified in the first argument. The bag of values can be either the second or third argument. In this case it is the second argument. The function evaluates to **true** if any value in the bag matches the third argument according to the matching function.

[18] - [20] The first argument: the matching function. The **Id** property specifies that the matching function is **urn:oasis:names:tc:acal:1.0:function:rfc822Name-match** using the short identifier name **rfc822Name-match**. This function returns **true** if an email address as its first argument belongs to a domain nominated by its second argument.

[22] - [26] The second argument: the bag of values. It is an attribute designator that selects a bag of **urn:oasis:names:tc:acal:1.0:data-type:rfc822Name** values from the **urn:oasis:names:tc:acal:1.0:subject:subject** attribute (short identifier name **subject-id**) in the **urn:oasis:names:tc:acal:1.0:subject-category:access-subject** category.

[28] - [31] The third argument: an attribute value. It is the email address domain to be matched, specifically, the character string **med.example.com**. The **DataType** property indicates that the data type of the value is **urn:oasis:names:tc:acal:1.0:data-type:string** using the short identifier name **string**.

[35] The end of the rule.

[37] The end of the policy. As mentioned above, this policy has only one rule, but more complex policies may have any number of rules.

### 6.1.2 Example Decision Request

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.

The following request, respectively in XML and JSON, expresses this decision request (the numbers in square brackets on the left-hand side are for referencing purposes and are not part of the request in either representation):

```
[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <Request xmlns="urn:oasis:names:tc:xacml:4.0:core:schema">
[03]   <ShortIdSetReference>urn:oasis:names:tc:acal:1.0:core:identifiers</ShortIdSetReference>
[04]   <RequestEntity
[05]     Category="access-subject">
[06]       <RequestAttribute
[07]         AttributeId="subject-id" DataType="rfc822Name">
[08]           <Value>bs@simpsons.com</Value>
[09]       </RequestAttribute>
[10]   </RequestEntity>
[11]   <RequestEntity
[12]     Category="resource">
[13]       <RequestAttribute
[14]         AttributeId="resource-id" DataType="anyURI">
[15]           <Value>file://example/med/record/patient/BartSimpson</Value>
[16]       </RequestAttribute>
[17]   </RequestEntity>
[18]   <RequestEntity
[19]     Category="action">
[20]       <RequestAttribute
[21]         AttributeId="action-id" DataType="string">
[22]           <Value>read</Value>
[23]       </RequestAttribute>
[24]   </RequestEntity>
[25] </Request>
[26] {
[27]   "ShortIdSetReference": ["urn:oasis:names:tc:acal:1.0:core:identifiers"],
```

```

[04] "RequestEntity": [
[05]   "Category": "access-subject",
[06]   "RequestAttribute": [
[07]     "AttributeId": "subject-id",
[08]     "DataType": "rfc822Name",
[09]     "Value": [
[10]       "bs@simpsons.com"
[11]     ]
[12]   ],
[13] },
[14] {
[15]   "Category": "resource",
[16]   "RequestAttribute": [
[17]     "AttributeId": "resource-id",
[18]     "DataType": "anyURI",
[19]     "Value": [
[20]       "file://example/med/record/patient/BartSimpson"
[21]     ]
[22]   ],
[23] },
[24] {
[25]   "Category": "action",
[26]   "RequestAttribute": [
[27]     "AttributeId": "action-id",
[28]     "DataType": "string",
[29]     "Value": [
[30]       "read"
[31]     ]
[32]   ],
[33] }
[34]

```

[01] A standard XML document tag.

[03] A reference to a short identifier set imported into the request. The names of the short identifiers in the set are available to use in this request. This short identifier set is defined by ACAL.

[04] - [33] The attributes describing the request separated into three categories.

[04] - [13] The first category of attributes.

[05] The **Category** property nominates the category using the short identifier name **access-subject**, which evaluates to **urn:oasis:names:tc:acal:1.0:subject-category:access-subject** using the imported short identifier set. The access subject category contains attributes of the entity making the access request. This category has only one attribute: the subject's identity, expressed as an e-mail name, is **bs@simpsons.com**.

[14] - [23] The second category of attributes is identified as **urn:oasis:names:tc:acal:1.0:attribute-category:resource** using the short identifier name **resource**. The resource category contains attributes of the resource to which access is being requested. This category has only one attribute: Bart Simpson's medical record, identified by its file URI, which is **file://medico/record/patient/BartSimpson**.

[24] - [33] The third category of attributes is identified as **urn:oasis:names:tc:acal:1.0:attribute-category:action** using the short identifier name **action**. The action category contains attributes that describe the action that the subject wishes to take on the resource. In this case the action is **read**.

[34] The end of the request.

A more complex decision request 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 **RequestEntity** objects. Examples of such attributes are attributes describing the environment or some application specific category of attributes.

The PDP processing this decision request locates the policy in its policy repository. It compares the attributes in the request with the policy target. Since the example policy's target is empty, the policy is applicable to this request.

The PDP now compares the attributes in the request with the condition of the one rule in this policy. The requesting **subject-id** attribute does not match **med.example.com**.

### 6.1.3 Example Response

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

```

[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <Response xmlns="urn:oasis:names:tc:xacml:4.0:core:schema">
[03]   <Result Decision="NotApplicable"/>
[06] </Response>

[02] {
[03]   "Result": [
[04]     "Decision": "NotApplicable"
[05]   ]
[06] }

```

[01] A standard XML document tag.

[02] The beginning of the response.

[03] - [05] The **Result** property contains the result of evaluating the decision request against the policy. In this case, the decision is **NotApplicable**. Therefore, a deny-biased PEP will deny access. A policy can return **Permit**, **Deny**, **NotApplicable** or **Indeterminate**.

[06] The end of the response.

## 6.2 Example Two

### 6.2.1 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.

### 6.2.2 Example Short Identifier Set

Policy writers are able to define sets of short identifiers to provide simple alias names to use in place of URIs. A set with the identifier **urn:oasis:names:tc:acal:1.0:core:identifiers** is defined by ACAL for the various identifiers assigned by this specification. However, a deployment will usually have need for additional identifiers, especially for locally-defined attributes, so it is usually desirable to define a set of additional short identifiers to use in the deployment, that may import the first set.

The following short-identifier set, in both the XML and JSON representations, defines short identifiers for the additional attributes in this example and also imports the standardized set.

```

<ShortIdSet xmlns="urn:oasis:names:tc:xacml:4.0:core:schema"
  Id="urn:oasis:names:tc:acal:1.0:example:identifiers">

  <!-- Include the short identifiers for standard URIs. -->
  <ShortIdSetReference>urn:oasis:names:tc:acal:1.0:core:identifiers</ShortIdSetReference>

  <!-- These are the short identifiers specific to the deployment. -->
  <!-- Attributes -->
  <ShortId Name="patient-number" Value="urn:oasis:names:tc:acal:1.0:example:attribute:patient-number"/>
  <ShortId Name="collection" Value="urn:oasis:names:tc:acal:1.0:example:attribute:collection"/>
  <ShortId Name="patient-date-of-birth" Value="urn:oasis:names:tc:acal:1.0:example:attribute:patient-date-of-birth"/>
  <ShortId Name="parent-guardian-id" Value="urn:oasis:names:tc:acal:1.0:example:attribute:parent-guardian-id"/>
  <ShortId Name="physician-id" Value="urn:oasis:names:tc:acal:1.0:example:attribute:physician-id"/>
  <ShortId Name="role" Value="urn:oasis:names:tc:acal:1.0:example:attribute:role"/>
  <ShortId Name="patient-contact" Value="urn:oasis:names:tc:acal:1.0:example:attribute:patient-contact"/>

</ShortIdSet>

```

```
{
  "Id": "urn:oasis:names:tc:acal:1.0:example:identifiers",
  "ShortIdSetReference": ["urn:oasis:names:tc:acal:1.0:core:identifiers"],
  "ShortId": [
    { "Name": "patient-number", "Value": "urn:oasis:names:tc:acal:1.0:example:attribute:patient-number" },
    { "Name": "collection", "Value": "urn:oasis:names:tc:acal:1.0:example:attribute:collection" },
    { "Name": "patient-date-of-birth", "Value": "urn:oasis:names:tc:acal:1.0:example:attribute:patient-date-of-birth" },
    { "Name": "parent-guardian-id", "Value": "urn:oasis:names:tc:acal:1.0:example:attribute:parent-guardian-id" },
    { "Name": "physician-id", "Value": "urn:oasis:names:tc:acal:1.0:example:attribute:physician-id" },
    { "Name": "role", "Value": "urn:oasis:names:tc:acal:1.0:example:attribute:role" },
    { "Name": "patient-contact", "Value": "urn:oasis:names:tc:acal:1.0:example:attribute:patient-contact" }
  ]
}
```

### 6.2.3 Example Decision Request

The following example illustrates a decision request to which the example rules may be applicable. It represents a request by the physician Julius Hibbert, in both the XML and JSON representations, to read the patient record of Bartholomew Simpson:

```
[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <Request xmlns="urn:oasis:names:tc:xacml:4.0:core:schema">
[03]   <ShortIdSetReference>urn:oasis:names:tc:acal:1.0:example:identifiers</ShortIdSetReference>
[04]   <RequestEntity
[05]     Category="access-subject">
[06]     <RequestAttribute
[07]       AttributeId="subject-id"
[08]       Issuer="med.example.com" DataType="rfc822Name">
[09]         <Value>Julius.Hibbert@med.example.com</Value>
[13]       </RequestAttribute>
[14]     <RequestAttribute
[15]       AttributeId="role"
[16]       Issuer="med.example.com" DataType="string">
[17]         <Value>physician</Value>
[21]       </RequestAttribute>
[22]     <RequestAttribute
[23]       AttributeId="physician-id"
[24]       Issuer="med.example.com" DataType="string">
[25]         <Value>jh1234</Value>
[29]       </RequestAttribute>
[30]     </RequestEntity>
[31]     <RequestEntity
[32]       Category="resource">
[33]       <RequestAttribute
[34]         AttributeId="patient-date-of-birth"
[35]         Issuer="med.example.com" DataType="date">
[36]           <Value>1992-03-21</Value>
[40]         </RequestAttribute>
[41]       <RequestAttribute
[42]         AttributeId="patient-number"
[43]         Issuer="med.example.com" DataType="integer">
[44]           <Value>555555</Value>
[48]         </RequestAttribute>
[49]       <RequestAttribute
[50]         AttributeId="patient-contact"
[51]         Issuer="med.example.com" DataType="rfc822Name">
[52]           <Value>b.simpson@example.com</Value>
[56]         </RequestAttribute>
[57]       </RequestEntity>
[58]     <RequestEntity
[59]       Category="action">
[60]       <RequestAttribute
[61]         AttributeId="action-id" DataType="string">
[62]           <Value>read</Value>
[66]         </RequestAttribute>
[67]       </RequestEntity>
[68]     <RequestEntity
[69]       Category="environment">
[70]       <RequestAttribute
[71]         AttributeId="current-date" DataType="date">
[72]           <Value>2010-01-11</Value>
[76]         </RequestAttribute>
[77]       </RequestEntity>
[78]     </Request>
[02] {
```

```

[03] "ShortIdSetReference": ["urn:oasis:names:tc:acal:1.0:example:identifiers"],
[04] "RequestEntity": [
[05]   "Category": "access-subject",
[06]   "RequestAttribute": [
[07]     "AttributeId": "subject-id",
[08]     "Issuer": "med.example.com",
[09]     "DataType": "rfc822Name",
[10]     "Value": [
[11]       "Julius.Hibbert@med.example.com"
[12]     ]
[13]   },
[14]   {
[15]     "AttributeId": "role",
[16]     "Issuer": "med.example.com",
[17]     "DataType": "string",
[18]     "Value": [
[19]       "physician"
[20]     ]
[21]   },
[22]   {
[23]     "AttributeId": "physician-id",
[24]     "Issuer": "med.example.com",
[25]     "DataType": "string",
[26]     "Value": [
[27]       "jh1234"
[28]     ]
[29]   }
[30] },
[31] {
[32]   "Category": "resource",
[33]   "RequestAttribute": [
[34]     "AttributeId": "patient-date-of-birth",
[35]     "Issuer": "med.example.com",
[36]     "DataType": "date",
[37]     "Value": [
[38]       "1992-03-21"
[39]     ]
[40]   },
[41]   {
[42]     "AttributeId": "patient-number",
[43]     "Issuer": "med.example.com",
[44]     "DataType": "integer",
[45]     "Value": [
[46]       555555
[47]     ]
[48]   },
[49]   {
[50]     "AttributeId": "patient-contact",
[51]     "Issuer": "med.example.com",
[52]     "DataType": "rfc822Name",
[53]     "Value": [
[54]       "b.simpson@example.com"
[55]     ]
[56]   }
[57] },
[58] {
[59]   "Category": "action",
[60]   "RequestAttribute": [
[61]     "AttributeId": "action-id",
[62]     "DataType": "string",
[63]     "Value": [
[64]       "read"
[65]     ]
[66]   }
[67] },
[68] {
[69]   "Category": "environment",
[70]   "RequestAttribute": [
[71]     "AttributeId": "current-date",
[72]     "DataType": "date",
[73]     "Value": [
[74]       "2010-01-11"
[75]     ]
[76]   }
[77] }
[78] }

```

[01] A standard XML document tag.

[03] A reference to a short identifier set imported into the request. The names of the short identifiers in the set are available to use in this request. The referenced short identifier set is customized for this example.

[04] - [30] Access subject attributes are placed in the `urn:oasis:names:tc:acal:1.0:subject-category:access-subject`

attribute category of the decision request. Each attribute consists of the attribute meta-data and the attribute value.

[06] - [13] The subject's `urn:oasis:names:tc:acal:1.0:subject:subject-id` attribute denoting the identity for which the request was issued.

[14] - [21] The subject's `urn:oasis:names:tc:acal:1.0:example:attribute:role` attribute.

[22] - [29] The subject's `urn:oasis:names:tc:acal:1.0:example:attribute:physician-id` attribute.

[31] - [57] Resource attributes representing the patient's medical record are placed in the `urn:oasis:names:tc:acal:1.0:attribute` attribute category of the decision request. Each attribute consists of attribute meta-data and an attribute value.

[33] - [40] The patient's `urn:oasis:names:tc:acal:1.0:example:attribute:patient-date-of-birth` attribute.

[41] - [48] The patient's `urn:oasis:names:tc:acal:1.0:example:attribute:patient-number` attribute.

[49] - [57] The patient's `urn:oasis:names:tc:acal:1.0:example:attribute:patient-contact` attribute.

[58] - [67] Action attributes are placed in the `urn:oasis:names:tc:acal:1.0:attribute-category:action` attribute category of the decision request.

[60] - [66] The `urn:oasis:names:tc:acal:1.0:action:action-id` attribute indicates that the requested action is `read`.

## 6.2.4 Example ACAL Rule Instances

**6.2.4.1 Rule 1** Rule 1 illustrates a policy with a simple rule containing a condition. It also illustrates the use of a variable definition to define an expression that may be used throughout the policy. The following ACAL policy, represented in XML and JSON, contains a rule instance expressing Rule 1 (the numbers in square brackets on the left-hand side are for referencing purposes and are not part of the policy in either representation):

```

[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <Policy
[03]   xmlns="urn:oasis:names:tc:xacml:4.0:core:schema"
[04]   PolicyId="urn:oasis:names:tc:acal:1.0:example:policyid:1"
[05]   Version="1.0"
[06]   CombiningAlgId="deny-overrides">
[07]   <ShortIdSetReference>urn:oasis:names:tc:acal:1.0:example:identifiers</ShortIdSetReference>
[08]   <VariableDefinition VariableId="patient_number_match">
[09]     <Apply FunctionId="string-equal">
[10]       <Apply FunctionId="string-one-and-only">
[11]         <AttributeDesignator
[12]           Category="access-subject"
[13]           AttributeId="patient-number"
[14]           DataType="string"/>
[15]     </Apply>
[16]     <Apply FunctionId="string-one-and-only">
[17]       <AttributeDesignator
[18]         Category="resource"
[19]         AttributeId="patient-number"
[20]         DataType="string"/>
[21]     </Apply>
[22]   </VariableDefinition>
[23]   <Target>
[24]     <Apply FunctionId="anyURI-is-in">
[25]       <Value DataType="anyURI">http://www.med.example.com/springfield-hospital</Value>
[26]       <AttributeDesignator
[27]         Category="resource"
[28]         AttributeId="collection"
[29]         DataType="anyURI"/>
[30]     </Apply>
[31]   </Target>
[32]   <Rule Id="Rule1" Effect="Permit">
[33]     <Description>A person may read any medical record for which he or she is the designated patient.</Description>
[34]     <Condition>
[35]       <Apply FunctionId="and">
[36]         <Apply FunctionId="any-of">
[37]           <Function Id="string-equal"/>
[38]           <Value>read</Value>
[39]         <AttributeDesignator
[40]           Category="action"
[41]           AttributeId="action-id"
[42]         </AttributeDesignator>
[43]       </Apply>
[44]     </Condition>
[45]   </Rule>
[46] </Policy>
```

```

[80]         DataType="string"/>
[83]     </Apply>
[85]     <VariableReference VariableId="patient_number_match"/>
[89]   </Apply>
[90] </Condition>
[91] </Rule>
[93] </Policy>

[02] {
[04]   "PolicyId": "urn:oasis:names:tc:acal:1.0:example:policyid:1",
[05]   "Version": "1.0",
[06]   "CombiningAlgId": "deny-overrides",
[07]   "ShortIdSetReference": ["urn:oasis:names:tc:acal:1.0:example:identifiers"],
[08]   "VariableDefinition": [
[09]     "VariableId": "patient_number_match",
[10]     "Expression": [
[11]       "Apply": [
[12]         "FunctionId": "string-equal",
[13]         "Expression": [
[14]           "Apply": [
[15]             "FunctionId": "string-one-and-only",
[16]             "Expression": [
[17]               "AttributeDesignator": [
[18]                 "Category": "access-subject",
[19]                 "AttributeId": "patient-number",
[20]                 "DataType": "string"
[21]               ]
[22]             ]
[23]           ]
[24]         ],
[25]         "Apply": [
[26]           "FunctionId": "string-one-and-only",
[27]           "Expression": [
[28]             "AttributeDesignator": [
[29]               "Category": "resource",
[30]               "AttributeId": "patient-number",
[31]               "DataType": "string"
[32]             ]
[33]           ]
[34]         ]
[35]       ]
[36]     ]
[37]   ],
[38] ],
[39]   "Target": [
[40]     "Apply": [
[41]       "FunctionId": "anyURI-is-in",
[42]       "Expression": [
[43]         "Value": [
[44]           "DataType": "anyURI",
[45]           "Value": "http://www.med.example.com/springfield-hospital"
[46]         ]
[47]       ],
[48]       "AttributeDesignator": [
[49]         "Category": "resource",
[50]         "AttributeId": "collection",
[51]         "DataType": "anyURI"
[52]       ]
[53]     ]
[54]   ],
[55] ],
[56]   "CombinerInput": [
[57]     "Rule": [
[58]       "Id": "Rule1",
[59]       "Effect": "Permit",
[60]       "Description": "A person may read any medical record for which he or she is the designated patient",
[61]       "Condition": [
[62]         "Apply": [
[63]           "FunctionId": "and",
[64]           "Expression": [
[65]             "Apply": [
[66]               "FunctionId": "any-of",
[67]               "Expression": [
[68]                 "Function": [
[69]                   "Id": "string-equal"
[70]                 ]
[71]               ],
[72]             ]
[73]           ]
[74]         ]
[75]       ]
[76]     ]
[77]   ]
[78] ]
[79] }

```

```

[72]         "Value": {
[73]             "DataType": "string",
[74]             "Value": "read"
[75]         }
[76]     },
[77]     "AttributeDesignator": {
[78]         "Category": "action",
[79]         "AttributeId": "action-id",
[80]         "DataType": "string"
[81]     }
[82] }
[83]
[84] },
[85]     "VariableReference": {
[86]         "VariableId": "patient_number_match"
[87]     }
[88] ]
[89]
[90] }
[91] }
[92] ]
[93] }

```

[07] A reference to a short identifier set imported into the policy. The names of the short identifiers in the set are available to use in this policy.

[08] - [38] A variable definition. It defines an expression that evaluates the truth of the statement: the patient-number subject attribute is equal to the patient-number attribute in the resource.

[11] - [12] The `FunctionId` property names the function to be used for comparison. In this case, the function is nominated with the short identifier name `string-equal`, which evaluates to `urn:oasis:names:tc:acal:1.0:function:string-equals` using the imported short identifier set. This function takes two arguments of type `urn:oasis:names:tc:acal:1.0:data-type:string`.

[17] - [21] An attribute designator that selects a bag of values for the patient-number subject attribute in the request context.

[18] The `Category` property names the category from which the desired attribute will be obtained. In this case, the category is nominated with the short identifier name `access-subject`, which evaluates to `urn:oasis:names:tc:acal:1.0:subject-category:access-subject` using the imported short identifier set.

[19] The `AttributeId` property nominates the patient-number attribute with the short identifier name `patient-number`, which evaluates to `urn:oasis:names:tc:acal:1.0:example:attribute:patient-number` using the imported short identifier set.

[20] The `DataType` property specifies the data type of the attribute. In this case, the data type is nominated with the short identifier name `string`, which evaluates to `urn:oasis:names:tc:acal:1.0:data-type:string` using the imported short identifier set.

[14] - [15] The attribute designator returns a bag of attribute values but the `urn:oasis:names:tc:acal:1.0:function:string-equals` function takes arguments that are single attribute values. The `FunctionId` property here names a function to convert the bag of values to a single value. The function is nominated with the short identifier name `string-one-and-only`, which evaluates to `urn:oasis:names:tc:acal:1.0:function:string-one-and-only` using the imported short identifier set. This function generates an `Indeterminate` result if its argument is not a bag with exactly one value.

[28] - [32] An attribute designator that selects a bag of values for the patient-number resource attribute in the request context. This attribute designator differs from the previous one in that it selects the attribute from the resource category, `urn:oasis:names:tc:acal:1.0:attribute-category:resource`, nominated with the short identifier name `resource`.

[25] - [26] The result of the second attribute designator must also be converted to a single value for comparison.

[39] - [55] A target, which must evaluate to `true` for the policy to be applicable. This target restricts the applicability of the policy to decision requests for access to medical records belonging to Springfield Hospital.

[40] - [54] The expression for the target, which applies a function.

[41] The `FunctionId` property specifies the `urn:oasis:names:tc:acal:1.0:function:anyURI-is-in` function using the short identifier name `anyURI-is-in`. This function takes an attribute value as its first argument and a bag of values as its second argument. The function evaluates to `true` if the first argument is equal to any value in the bag.

[43] - [46] The first argument: an attribute value. It is a literal URI to be matched, specifically `http://www.med.example.com/springfield-hospital`. The `DataType` property indicates that the data type of the value is `urn:oasis:names:tc:acal:1.0:data-type:anyURI` using the short identifier name `anyURI`.

[48] - [52] The second argument: the bag of values. It is an attribute designator that selects a bag of `urn:oasis:names:tc:acal:1.0:data-type:anyURI` values from the `urn:oasis:names:tc:acal:1.0:example:attribute` attribute (short identifier name `collection`) in the `urn:oasis:names:tc:acal:1.0:attribute-category:resource` category.

[57] - [59] The beginning of a rule definition. The `Id` property assigns a string identifier for the rule. The `Effect` property specifies the value the rule emits when it evaluates to `true`.

[60] A free-form description of the rule.

[61] - [90] A condition, which must evaluate to `true` for the rule to be applicable.

[62] - [89] The expression for the condition.

[62] - [63] The expression for the condition is a conjunction of two terms. The `FunctionId` property specifies a conjunction using the `urn:oasis:names:tc:acal:1.0:function:and` function nominated with the short identifier name `and`.

[65] - [83] The first term is a function.

[65] - [66] The `FunctionId` property specifies the `urn:oasis:names:tc:acal:1.0:function:any-of` function using the short identifier name `any-of`. This function compares an attribute value to each of the attribute values in a bag according to a matching function that is specified in the first argument. The bag of values can be either the second or third argument. In this case it is the third argument. The function evaluates to `true` if any value in the bag matches the second argument according to the matching function.

[68] - [70] The first argument: the matching function. The `Id` property specifies that the matching function is `urn:oasis:names:tc:acal:1.0:function:string-equal` using the short identifier name `string-equal`.

[72] - [75] The second argument: an attribute value. It is a literal string to be matched, specifically `read`. The `DataType` property indicates that the data type of the value is `urn:oasis:names:tc:acal:1.0:data-type:string` using the short identifier name `string`.

[77] - [81] The third argument: the bag of values. It is an attribute designator that selects a bag of `urn:oasis:names:tc:acal:1.0:data-type:string` values from the `urn:oasis:names:tc:acal:1.0:action:action-id` attribute (short identifier name `action-id`) in the `urn:oasis:names:tc:acal:1.0:attribute-category:action` category.

[85] - [87] The second term is a reference to a variable definition defined elsewhere in the policy.

**6.2.4.2 Rule 2** Rule 2 illustrates the use of a mathematical function, i.e., `urn:oasis:names:tc:acal:1.0:function:date-add-yearMonthDuration`, to calculate the date of the patient's sixteenth birthday. The following ACAL policy, represented in XML and JSON, contains a rule instance expressing Rule 2 (the numbers in square brackets on the left-hand side are for referencing purposes and are not part of the policy in either representation):

```
[001] <?xml version="1.0" encoding="UTF-8"?>
[002] <Policy
[003]   xmlns="urn:oasis:names:tc:xacml:4.0:core:schema"
[004]   PolicyId="urn:oasis:names:tc:acal:1.0:example:policyid:2"
[005]   Version="1.0"
[006]   CombiningAlgId="deny-overrides"
[007]   <ShortIdSetReference>urn:oasis:names:tc:acal:1.0:example:identifiers</ShortIdSetReference>
[008]   <VariableDefinition VariableId="patient-under-16">
[011]     <Apply FunctionId="date-less-than-or-equal">
[014]       <Apply FunctionId="date-one-and-only">
[017]         <AttributeDesignator
[018]           Category="environment"
[019]           AttributeId="current-date"
[020]           DataType="date"/>
[023]     </Apply>
[025]     <Apply FunctionId="date-add-yearMonthDuration">
[028]       <Apply FunctionId="date-one-and-only">
[031]         <AttributeDesignator
[032]           Category="resource"
[033]           AttributeId="patient-date-of-birth"
[034]           DataType="date"/>
[037]     </Apply>
[039]     <Value DataType="yearMonthDuration">P16Y</Value>
```

```

[044]      </Apply>
[046]    </Apply>
[048]  </VariableDefinition>
[049]  <Target>
[050]    <Apply FunctionId="anyURI-is-in">
[053]      <Value DataType="anyURI">http://www.med.example.com/springfield-hospital</Value>
[058]      <AttributeDesignator
[059]        Category="resource"
[060]        AttributeId="collection"
[061]        DataType="anyURI"/>
[064]    </Apply>
[065]  </Target>
[067] <Rule Id="Rule2" Effect="Permit">
[070]   <Description>A person may read any medical record in the http://www.med.example.com/records.xsd namespace for which he has been granted read access</Description>
[071]   <Condition>
[072]     <Apply FunctionId="and">
[075]       <Apply FunctionId="string-is-in">
[078]         <Value DataType="string">read</Value>
[082]         <AttributeDesignator
[083]           Category="action"
[084]           AttributeId="action-id"
[085]           DataType="string"/>
[088]     </Apply>
[090]     <Apply FunctionId="rfc822Name-equal">
[093]       <Apply FunctionId="rfc822Name-one-and-only">
[096]         <AttributeDesignator
[097]           Category="access-subject"
[098]           AttributeId="subject-id"
[099]           DataType="rfc822Name"/>
[102]       </Apply>
[104]       <Apply FunctionId="rfc822Name-one-and-only">
[107]         <AttributeDesignator
[108]           Category="resource"
[109]           AttributeId="parent-guardian-id"
[110]           DataType="rfc822Name"/>
[113]         </Apply>
[115]       </Apply>
[117]       <VariableReference VariableId="patient-under-16"/>
[121]     </Apply>
[122]   </Condition>
[123] </Rule>
[125] </Policy>

[002] {
[004]   "PolicyId": "urn:oasis:names:tc:acal:1.0:example:policyid:2",
[005]   "Version": "1.0",
[006]   "CombiningAlgId": "deny-overrides",
[007]   "ShortIdSetReference": ["urn:oasis:names:tc:acal:1.0:example:identifiers"],
[008]   "VariableDefinition": [
[009]     {"VariableId": "patient-under-16",
[010]      "Expression": {
[011]        "Apply": {
[012]          "FunctionId": "date-less-than-or-equal",
[013]          "Expression": [
[014]            "Apply": {
[015]              "FunctionId": "date-one-and-only",
[016]              "Expression": [
[017]                "AttributeDesignator": {
[018]                  "Category": "environment",
[019]                  "AttributeId": "current-date",
[020]                  "DataType": "date"
[021]                }
[022]              ]
[023]            }
[024]          },
[025]          "Apply": {
[026]            "FunctionId": "date-add-yearMonthDuration",
[027]            "Expression": [
[028]              "Apply": {
[029]                "FunctionId": "date-one-and-only",
[030]                "Expression": [
[031]                  "AttributeDesignator": {
[032]                    "Category": "resource",
[033]                    "AttributeId": "patient-date-of-birth",
[034]                    "DataType": "date"
[035]                  }
[036]                ]
[037]              ]
[038]            ]
[039]          }
[040]        }
[041]      }
[042]    }
[043]  ]
[044] }

```

```

[037]         }
[038]     },{
[039]         "Value": {
[040]             "DataType": "yearMonthDuration",
[041]             "Value": "P16Y"
[042]         }
[043]     }]
[044]   }
[045] }
[046] }
[047] }
[048] ],
[049] "Target": {
[050]   "Apply": {
[051]     "FunctionId": "anyURI-is-in",
[052]     "Expression": [
[053]       "Value": {
[054]           "DataType": "anyURI",
[055]           "Value": "http://www.med.example.com/springfield-hospital"
[056]       }
[057]     ],
[058]     "AttributeDesignator": {
[059]       "Category": "resource",
[060]       "AttributeId": "collection",
[061]       "DataType": "anyURI"
[062]     }
[063]   ]
[064] }
[065] },
[066] "CombinerInput": [
[067]   "Rule": {
[068]     "Id": "Rule2",
[069]     "Effect": "Permit",
[070]     "Description": "A person may read any medical record in the http://www.med.example.com/records.xsd namespace for whi
[071]     "Condition": {
[072]       "Apply": {
[073]         "FunctionId": "and",
[074]         "Expression": [
[075]           "Apply": {
[076]             "FunctionId": "string-is-in",
[077]             "Expression": [
[078]               "Value": {
[079]                 "DataType": "string",
[080]                 "Value": "read"
[081]               },
[082]               "AttributeDesignator": {
[083]                 "Category": "action",
[084]                 "AttributeId": "action-id",
[085]                 "DataType": "string"
[086]               }
[087]             ]
[088]           }
[089]         ],
[090]         "Apply": {
[091]           "FunctionId": "rfc822Name-equal",
[092]           "Expression": [
[093]             "Apply": {
[094]               "FunctionId": "rfc822Name-one-and-only",
[095]               "Expression": [
[096]                 "AttributeDesignator": {
[097]                   "Category": "access-subject",
[098]                   "AttributeId": "subject-id",
[099]                   "DataType": "rfc822Name"
[100]                 }
[101]               ]
[102]             }
[103]           ],
[104]           "Apply": {
[105]             "FunctionId": "rfc822Name-one-and-only",
[106]             "Expression": [
[107]               "AttributeDesignator": {
[108]                 "Category": "resource",
[109]                 "AttributeId": "parent-guardian-id",
[110]                 "DataType": "rfc822Name"
[111]               }
[112]             ]
[113]           ]
[114]         ]
[115]       ]
[116]     }
[117]   }
[118] }
[119] }

```

```

[113]         }
[114]     }]
[115]   }
[116] }, {
[117]   "VariableReference": {
[118]     "VariableId": "patient-under-16"
[119]   }
[120] }
[121]   }
[122] }
[123] }
[124] }
[125] }

```

[008] - [048] A variable definition containing 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 years to the patient's date of birth.

[011] - [012] The `urn:oasis:names:tc:acal:1.0:function:date-less-than-or-equal` function (short identifier name `date-less-than-or-equal`) is used to compare the two date arguments.

[014] - [023] The first date argument uses the `urn:oasis:names:tc:acal:1.0:function:date-one-and-only` function (short identifier name `date-one-and-only`) to ensure that the bag of values selected by its argument contains exactly one value of type `urn:oasis:names:tc:acal:1.0:data-type:date` (short identifier name `date`).

[017] - [21] The current date is evaluated with an attribute designator selecting the `urn:oasis:names:tc:acal:1.0:environment` environment attribute (short identifier name `current-date`).

[025] - [044] The second date argument uses the `urn:oasis:names:tc:acal:1.0:function:date-add-yearMonthDuration` function (short identifier name `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 `urn:oasis:names:tc:acal:1.0:data-type:date` and the second is of type `urn:oasis:names:tc:acal:1.0:data-type:yearMonthDuration` (short identifier name `yearMonthDuration`).

[031] - [035] An attribute designator that selects a bag of `urn:oasis:names:tc:acal:1.0:data-type:date` values from the `urn:oasis:names:tc:acal:1.0:example:attribute:patient-date-of-birth` attribute (short identifier name `patient-date-of-birth`) in the `urn:oasis:names:tc:acal:1.0:attribute-category:resource` category.

[039] - [042] A literal `urn:oasis:names:tc:acal:1.0:function:date-add-yearMonthDuration` value that specifies a duration of 16 years.

[049] - [065] The target restricts the applicability of the policy to decision requests for access to medical records belonging to Springfield Hospital.

[067] - [123] The rule that assesses the decision request.

[071] - [122] The rule's condition, which must evaluate to `true` for the rule to be applicable. This condition evaluates, in part, the truth of the statement: the requestor is the designated parent or guardian and the patient is under 16 years of age.

[072] - [121] The expression for the condition is the conjunction of three terms using the `urn:oasis:names:tc:acal:1.0:function` function.

[075] - [088] The first term is satisfied if the requested access is `read`.

[090] - [115] The second term tests whether the subject is the designated parent or guardian of the patient according to the resource. The function is `urn:oasis:names:tc:acal:1.0:function:rfc822Name-equal` and it takes two arguments of type `urn:oasis:names:tc:acal:1.0:data-type:rfc822Name`.

[093] - [102] Since `urn:oasis:names:tc:acal:1.0:function:rfc822Name-equal` takes arguments of type `urn:oasis:names:tc:acal:1.0:data-type:rfc822Name`, the `urn:oasis:names:tc:acal:1.0:function:rfc822Name-one` function is used to ensure that the subject attribute `urn:oasis:names:tc:acal:1.0:subject:subject-id` in the decision request contains exactly one value.

[096] - [100] The value of the subject attribute `urn:oasis:names:tc:acal:1.0:subject:subject-id` is selected from the decision request using an attribute designator.

[104] - [113] As above, the `urn:oasis:names:tc:acal:1.0:function:rfc822Name-one-and-only` function is used to ensure that the bag of values selected by its argument contains exactly one value of type

urn:oasis:names:tc:acal:1.0:data-type:rfc822Name.

[107] - [111] The value of the resource attribute urn:oasis:names:tc:acal:1.0:example:attribute:parent-guardian-id (short identifier name parent-guardian-id) is selected from the decision request using an attribute designator.

[117] - [119] The third term is a reference to the variable definition that tests whether the patient is under 16 years of age.

#### 6.2.4.3 Rule 3

Rule 3 illustrates the use of a notice expression.

```
[001] <?xml version="1.0" encoding="UTF-8"?>
[002] <Policy
[003]   xmlns="urn:oasis:names:tc:xacml:4.0:core:schema"
[004]   PolicyId="urn:oasis:names:tc:acal:1.0:example:policyid:3"
[005]   Version="1.0"
[006]   CombiningAlgId="deny-overrides"
[007]   <ShortIdSetReference>urn:oasis:names:tc:acal:1.0:example:identifiers</ShortIdSetReference>
[008]   <Description>Policy for any medical record in the Springfield Hospital collection.</Description>
[009]   <Target>
[010]     <Apply FunctionId="anyURI-is-in">
[011]       <Value DataType="anyURI">http://www.med.example.com/springfield-hospital</Value>
[012]       <AttributeDesignator
[013]         Category="resource"
[014]         AttributeId="collection"
[015]         DataType="anyURI"/>
[016]     </Apply>
[017]   </Target>
[018]   <Rule Id="Rule3" Effect="Permit">
[019]     <Description>A physician may write any medical element in a record for which he or she is the designated primary care provider</Description>
[020]     <Condition>
[021]       <Apply FunctionId="and">
[022]         <Apply FunctionId="string-is-in">
[023]           <Value DataType="string">physician</Value>
[024]           <AttributeDesignator
[025]             Category="access-subject"
[026]             AttributeId="role"
[027]             DataType="string"/>
[028]         </Apply>
[029]         <Apply FunctionId="string-is-in">
[030]           <Value DataType="string">write</Value>
[031]           <AttributeDesignator
[032]             Category="action"
[033]             AttributeId="action-id"
[034]             DataType="string"/>
[035]         </Apply>
[036]         <Apply FunctionId="string-equal">
[037]           <Apply FunctionId="string-one-and-only">
[038]             <AttributeDesignator
[039]               Category="access-subject"
[040]               AttributeId="physician-id"
[041]               DataType="string"/>
[042]             </Apply>
[043]             <Apply FunctionId="string-one-and-only">
[044]               <AttributeDesignator
[045]                 Category="resource"
[046]                 AttributeId="primary-care-physician"
[047]                 DataType="string"/>
[048]               </Apply>
[049]             </Apply>
[050]           </Apply>
[051]         </Apply>
[052]       </Condition>
[053]     </Rule>
[054]     <NoticeExpression
[055]       IsObligation="true"
[056]       Id="email"
[057]       AppliesTo="Permit">
[058]         <AttributeAssignmentExpression
[059]           <AttributeId="mailto">
[060]             <AttributeDesignator
[061]               Category="resource"
[062]               AttributeId="patient-contact"
[063]               DataType="rfc822Name"/>
[064]           </AttributeAssignmentExpression>
[065]           <AttributeAssignmentExpression
[066]             AttributeId="text">
[067]             <Value DataType="string">Your medical record has been accessed by: </Value>
[068]           </AttributeAssignmentExpression>
[069]         </NoticeExpression>
[070]       <AttributeAssignmentExpression
[071]         AttributeId="text">
[072]           <Value DataType="string">Accessed by: </Value>
[073]         </AttributeAssignmentExpression>
[074]       </AttributeAssignmentExpression>
[075]     </AttributeAssignmentExpression>
[076]   </AttributeAssignmentExpression>
[077] </AttributeAssignmentExpression>
[078] </AttributeAssignmentExpression>
[079] </AttributeAssignmentExpression>
[080] </AttributeAssignmentExpression>
[081] </AttributeAssignmentExpression>
[082] </AttributeAssignmentExpression>
[083] </AttributeAssignmentExpression>
[084] </AttributeAssignmentExpression>
[085] </AttributeAssignmentExpression>
[086] </AttributeAssignmentExpression>
[087] </AttributeAssignmentExpression>
[088] </AttributeAssignmentExpression>
[089] </AttributeAssignmentExpression>
[090] </AttributeAssignmentExpression>
[091] </AttributeAssignmentExpression>
[092] </AttributeAssignmentExpression>
[093] </AttributeAssignmentExpression>
[094] </AttributeAssignmentExpression>
[095] </AttributeAssignmentExpression>
[096] </AttributeAssignmentExpression>
[097] </AttributeAssignmentExpression>
[098] </AttributeAssignmentExpression>
[099] </AttributeAssignmentExpression>
[100] </AttributeAssignmentExpression>
[101] </AttributeAssignmentExpression>
[102] </AttributeAssignmentExpression>
[103] </AttributeAssignmentExpression>
[104] </AttributeAssignmentExpression>
[105] </AttributeAssignmentExpression>
[106] </AttributeAssignmentExpression>
[107] </AttributeAssignmentExpression>
[108] </AttributeAssignmentExpression>
[109] </AttributeAssignmentExpression>
[110] </AttributeAssignmentExpression>
[111] </AttributeAssignmentExpression>
[112] </AttributeAssignmentExpression>
[113] </AttributeAssignmentExpression>
[114] </AttributeAssignmentExpression>
[115] </AttributeAssignmentExpression>
```

```

[120]      </AttributeAssignmentExpression>
[121]      <AttributeAssignmentExpression
[122]          AttributeId="text">
[124]          <AttributeDesignator
[125]              Category="access-subject"
[126]              AttributeId="subject-id"
[127]              DataType="string"/>
[130]      </AttributeAssignmentExpression>
[131]  </NoticeExpression>
[132] </Policy>

[002] {
[004]     "PolicyId": "urn:oasis:names:tc:acal:1.0:example:policyid:3",
[005]     "Version": "1.0",
[006]     "CombiningAlgId": "deny-overrides",
[007]     "ShortIdSetReference": ["urn:oasis:names:tc:acal:1.0:example:identifiers"],
[008]     "Description": "Policy for any medical record in the Springfield Hospital collection.",
[009]     "Target": {
[010]         "Apply": {
[011]             "FunctionId": "anyURI-is-in",
[012]             "Expression": [
[013]                 "Value": {
[014]                     "DataType": "anyURI",
[015]                     "Value": "http://www.med.example.com/springfield-hospital"
[016]                 }
[017]             ],
[018]             "AttributeDesignator": {
[019]                 "Category": "resource",
[020]                 "AttributeId": "collection",
[021]                 "DataType": "anyURI"
[022]             }
[023]         }
[024]     },
[025]     "CombinerInput": [
[026]         "Rule": {
[027]             "Id": "Rule3",
[028]             "Effect": "Permit",
[029]             "Description": "A physician may write any medical element in a record for which he or she is the designated primary care provider",
[030]             "Condition": {
[031]                 "Apply": {
[032]                     "FunctionId": "and",
[033]                     "Expression": [
[034]                         "Apply": {
[035]                             "FunctionId": "string-is-in",
[036]                             "Expression": [
[037]                                 "Value": {
[038]                                     "DataType": "string",
[039]                                     "Value": "physician"
[040]                                 }
[041]                             ],
[042]                         ],
[043]                         "AttributeDesignator": {
[044]                             "Category": "access-subject",
[045]                             "AttributeId": "role",
[046]                             "DataType": "string"
[047]                         }
[048]                     ],
[049]                 }
[050]             },
[051]             "Apply": {
[052]                 "FunctionId": "string-is-in",
[053]                 "Expression": [
[054]                     "Value": {
[055]                         "DataType": "string",
[056]                         "Value": "write"
[057]                     }
[058]                 ],
[059]                 "AttributeDesignator": {
[060]                     "Category": "action",
[061]                     "AttributeId": "action-id",
[062]                     "DataType": "string"
[063]                 }
[064]             ],
[065]         }
[066]     },
[067]     "Apply": {
[068]         "FunctionId": "string-equal",

```

```

[069]     "Expression": [
[070]       "Apply": {
[071]         "FunctionId": "string-one-and-only",
[072]         "Expression": [
[073]           "AttributeDesignator": {
[074]             "Category": "access-subject",
[075]             "AttributeId": "physician-id",
[076]             "DataType": "string"
[077]           }
[078]         }
[079]       }
[080]     },
[081]     "Apply": {
[082]       "FunctionId": "string-one-and-only",
[083]       "Expression": [
[084]         "AttributeDesignator": {
[085]           "Category": "resource",
[086]           "AttributeId": "primary-care-physician",
[087]           "DataType": "string"
[088]         }
[089]       }
[090]     }
[091]   }
[092] }
[093] }
[094] }
[095] }
[096] }
[097] ],
[098] "NoticeExpression": [
[099]   "IsObligation": true,
[100]   "Id": "email",
[101]   "AppliesTo": "Permit",
[102]   "AttributeAssignmentExpression": [
[103]     "AttributeId": "mailto",
[104]     "Expression": {
[105]       "AttributeDesignator": {
[106]         "Category": "resource",
[107]         "AttributeId": "patient-contact",
[108]         "DataType": "rfc822Name"
[109]       }
[110]     }
[111]   },
[112]   "AttributeId": "text",
[113]   "Expression": {
[114]     "Value": {
[115]       "Value": {
[116]         "Value": {
[117]           "Value": "Your medical record has been accessed by: "
[118]         }
[119]       }
[120]     }
[121]   },
[122]   "AttributeId": "text",
[123]   "Expression": {
[124]     "AttributeDesignator": {
[125]       "Category": "access-subject",
[126]       "AttributeId": "subject-id",
[127]       "DataType": "string"
[128]     }
[129]   }
[130] }
[131] ]
[132] }

```

[009] - [025] The target restricts the applicability of the policy to decision requests for access to medical records belonging to Springfield Hospital.

[027] - [096] The rule that assesses the decision request.

[031] - [095] The rule's condition, which must evaluate to `true` for the rule to be applicable.

[032] - [094] The expression for the condition is the conjunction of three terms using the `urn:oasis:names:tc:acal:1.0:function` function.

[035] - [049] The first term is satisfied if the subject's `urn:oasis:names:tc:acal:1.0:example:attribute:role` attribute contains the string value `physician`.

[051] - [065] The second term is satisfied if the requested access is **write**.

[067] - [092] The third term is satisfied if the accessing physician is the patient's primary physician, i.e., if the **urn:oasis:names:tc:acal:1.0:example:attribute:physician-id** attribute of the access subject is equal to the **urn:oasis:names:tc:acal:1.0:example:attribute:primary-care-physician** attribute of the resource.

[098] - [131] A notice expression.

[099] The **IsObligation** property indicates that this notice is an obligation rather than advice. 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 policy contains a single notice expression, which will be evaluated into an obligation notice when the policy is evaluated.

[100] The **Id** property indicates the kind of obligation. The PEP associates the identifier with particular processing requirements. In this case, the obligation requires the PEP to send an email.

[101] The **AppliesTo** property specifies the authorization decision value for which the obligation notice derived from the notice expression must be fulfilled. In this case, the obligation must be fulfilled when access is permitted.

[102] - [130] Additional information may be provided in a notice by evaluating attribute assignment expressions.

[102] - [111] The first attribute assignment indicates the recipient email address, which is obtained from the **urn:oasis:names:tc:acal:1.0:example:attribute:patient-contact** resource attribute using an attribute designator.

[112] - [120] The second attribute assignment contains literal text for the email body.

[121] - [130] The third attribute assignment provides additional text for the email body, specifically the identity of the physician, which is obtained from the **urn:oasis:names:tc:acal:1.0:subject:subject-id** subject attribute using an attribute designator.

#### 6.2.4.4 Rule 4

Rule 4 illustrates the use of the **Deny** effect value.

```
[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <Policy
[03]   xmlns="urn:oasis:names:tc:xacml:4.0:core:schema"
[04]   PolicyId="urn:oasis:names:tc:acal:1.0:example:policyid:4"
[05]   Version="1.0"
[06]   CombiningAlgId="deny-overrides"
[07]   <ShortIdSetReference>urn:oasis:names:tc:acal:1.0:example:identifiers</ShortIdSetReference>
[08]   <Rule
[09]     Id="Rule4"
[10]     Effect="Deny"
[11]     <Description>An Administrator shall not be permitted to read or write medical elements of a patient record in the Springfield Hospital system</Description>
[12]     <Condition>
[13]       <Apply FunctionId="and">
[14]         <Apply FunctionId="string-is-in">
[15]           <Value DataType="string">administrator</Value>
[16]           <AttributeDesignator
[17]             Category="access-subject"
[18]             AttributeId="role"
[19]             DataType="string"/>
[20]         </Apply>
[21]         <Apply FunctionId="anyURI-is-in">
[22]           <Value DataType="anyURI">http://www.med.example.com/springfield-hospital</Value>
[23]           <AttributeDesignator
[24]             Category="resource"
[25]             AttributeId="collection"
[26]             DataType="anyURI"/>
[27]         </Apply>
[28]       </Apply FunctionId="or">
[29]       <Apply FunctionId="string-is-in">
[30]         <Value DataType="string">read</Value>
[31]         <AttributeDesignator
[32]           Category="action"
[33]           AttributeId="action-id"
[34]           DataType="string"/>
[35]       </Apply>
[36]       <Apply FunctionId="string-is-in">
[37]         <Value DataType="string">write</Value>
[38]         <AttributeDesignator
[39]           Category="action"
[40]           AttributeId="action-id"
[41]           DataType="string"/>
[42]       </Apply>
[43]     </Condition>
[44]   </Rule>
[45] </Policy>
```

```

[82]         </Apply>
[84]     </Apply>
[86]     </Apply>
[87]   </Condition>
[88] </Rule>
[90] </Policy>

[02] {
[04]   "PolicyId": "urn:oasis:names:tc:acal:1.0:example:policyid:4",
[05]   "Version": "1.0",
[06]   "CombiningAlgId": "deny-overrides",
[07]   "ShortIdSetReference": ["urn:oasis:names:tc:acal:1.0:example:identifiers"],
[08]   "CombinerInput": [
[09]     "Rule": {
[10]       "Id": "Rule4",
[11]       "Effect": "Deny",
[12]       "Description": "An Administrator shall not be permitted to read or write medical elements of a patient record in the system",
[13]       "Condition": {
[14]         "Apply": {
[15]           "FunctionId": "and",
[16]           "Expression": [
[17]             "Apply": {
[18]               "FunctionId": "string-is-in",
[19]               "Expression": [
[20]                 "Value": {
[21]                   "DataType": "string",
[22]                   "Value": "administrator"
[23]                 }
[24]               ],
[25]               "AttributeDesignator": {
[26]                 "Category": "access-subject",
[27]                 "AttributeId": "role",
[28]                 "DataType": "string"
[29]               }
[30]             ]
[31]           }
[32]         },
[33]         "Apply": {
[34]           "FunctionId": "anyURI-is-in",
[35]           "Expression": [
[36]             "Value": {
[37]               "DataType": "anyURI",
[38]               "Value": "http://www.med.example.com/springfield-hospital"
[39]             }
[40]           ],
[41]           "AttributeDesignator": {
[42]             "Category": "resource",
[43]             "AttributeId": "collection",
[44]             "DataType": "anyURI"
[45]           }
[46]         ]
[47]       },
[48]     },
[49]     "Apply": {
[50]       "FunctionId": "or",
[51]       "Expression": [
[52]         "Apply": {
[53]           "FunctionId": "string-is-in",
[54]           "Expression": [
[55]             "Value": {
[56]               "DataType": "string",
[57]               "Value": "read"
[58]             }
[59]           ],
[60]           "AttributeDesignator": {
[61]             "Category": "action",
[62]             "AttributeId": "action-id",
[63]             "DataType": "string"
[64]           }
[65]         ]
[66]       }
[67]     },
[68]     "Apply": {
[69]       "FunctionId": "string-is-in",
[70]       "Expression": [
[71]         "Value": {
[72]           "DataType": "string",

```

```

[73]           "Value": "write"
[74]       }
[75]     },
[76]   },
[77]   "AttributeDesignator": {
[78]     "Category": "action",
[79]     "AttributeId": "action-id",
[80]     "DataType": "string"
[81]   }
[82] }
[83] }
[84] }
[85] }
[86] }
[87] }
[88] }
[89] }
[90] }

```

[09] - [72] The rule that assesses the decision request.

[11] The rule effect is **Deny**. If the rule's condition is **true** then the rule evaluates to **Deny**.

[13] - [71] The rule's condition, which must evaluate to **true** for the rule to be applicable.

[14] - [70] The expression for the condition is the conjunction of three terms using the **urn:oasis:names:tc:acal:1.0:function** function.

[17] - [31] The first term is satisfied if the subject's **urn:oasis:names:tc:acal:1.0:example:attribute:role** attribute contains the string value **administrator**.

[33] - [47] The second term checks that the request is to access a medical record belonging to Springfield Hospital.

[49] - [84] The third term is a disjunction of two terms using the **urn:oasis:names:tc:acal:1.0:function:or** function (short identifier name **or**).

[52] - [66] The first term of the disjunction is satisfied if the requested access is **read**.

[68] - [82] The second term of the disjunction is satisfied if the requested access is **write**.

If an administrator attempts to read or write a patient record, then the condition is satisfied and the rule evaluates to **Deny**. According to the policy's **urn:oasis:names:tc:acal:1.0:combining-algorithm:deny-overrides** combining algorithm, if any of the policy's rules (there is only one here) evaluates to **Deny** then the policy evaluates to **Deny** and the administrator is prevented from accessing the patient record.

**6.2.4.5 Example Policy with Nested Policies** This section uses the examples of the previous sections to illustrate the process of combining policies. The policy governing access to medical elements of a record is formed from each of the four rules described in Section 6.2.4. In plain language, the combined rule is:

- Either the requestor is the patient (Rule 1); or
- the requestor is the parent or guardian and the patient is under 16 (Rule 2); or
- the requestor is the primary care physician and a notification is sent to the patient (Rule 3); and
- the requestor is not an administrator (Rule 4).

The following policy illustrates the combined policies. Rule 1, Rule 2 and Rule 3 are duplicated in a new nested policy. The policy containing Rule 4 is included by reference.

```

[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <Policy
[03]   xmlns="urn:oasis:names:tc:xacml:4.0:core:schema"
[04]   PolicyId="urn:oasis:names:tc:acal:1.0:example:policyid:5"
[05]   Version="1.0"
[06]   CombiningAlgId="deny-overrides"
[07]   <ShortIdSetReference>urn:oasis:names:tc:acal:1.0:example:identifiers</ShortIdSetReference>
[08]   <Description>Collected policies.</Description>
[09]   <Policy
[10]     PolicyId="urn:oasis:names:tc:acal:1.0:example:policyid:6"
[11]     Version="1.0"
[12]     CombiningAlgId="deny-overrides"
[13]     <Target>
[14]       <Apply FunctionId="anyURI-is-in">
[15]         <Value DataType="anyURI">http://www.med.example.com/springfield-hospital</Value>

```

```

[23]      <AttributeDesignator
[24]          Category="resource"
[25]          AttributeId="collection"
[26]          DataType="anyURI"/>
[27]      
```

[28]

```

[29]  
```

[30]

```

[31]  
```

[32]

```

[33]  
```

[34]

```

[35]  
```

[36]

```

[37]  
```

[38]

```

[39]  
```

[40]

```

[41]  
```

[42]

```

[43]  
```

[44]

```

[45]  
```

[46]

```

[47]  
```

[48]

```

[49]  
```

[50]

```

[51]  
```

[52]

```

[53]  
```

[54]

```

[01]  
```

[02] {

```

[03]  "PolicyId": "urn:oasis:names:tc:acal:1.0:example:policyid:5",
[04]  "Version": "1.0",
[05]  "CombiningAlgId": "deny-overrides",
[06]  "ShortIdSetReference": ["urn:oasis:names:tc:acal:1.0:example:identifiers"],
[07]  "Description": "Collected policies.",
[08]  "CombinerInput": [
[09]    {
[10]      "Policy": {
[11]          "PolicyId": "urn:oasis:names:tc:acal:1.0:example:policyid:6",
[12]          "Version": "1.0",
[13]          "CombiningAlgId": "deny-overrides",
[14]          "Target": {
[15]              "Apply": {
[16]                  "FunctionId": "anyURI-is-in",
[17]                  "Expression": [
[18]                      {
[19]                          "Value": {
[20]                              "DataType": "anyURI",
[21]                              "Value": "http://www.med.example.com/springfield-hospital"
[22]                          }
[23]                      },
[24]                      {
[25]                          "AttributeDesignator": {
[26]                              "Category": "resource",
[27]                              "AttributeId": "collection",
[28]                              "(DataType": "anyURI"
[29]                          )
[30]                      },
[31]                      "CombinerInput": [
[32]                          {
[33]                              "Rule": {
[34]                                  "RuleId": "Rule1",
[35]                                  "Effect": "Permit",
[36]                                  ...
[37]                              }
[38]                          },
[39]                          {
[40]                              "Rule": {
[41]                                  "RuleId": "Rule2",
[42]                                  "Effect": "Permit",
[43]                                  ...
[44]                              }
[45]                          },
[46]                          {
[47]                              ...
[48]                          }
[49]                      ]
[50]                  },
[51]                  "PolicyReference": {
[52]                      "Id": "urn:oasis:names:tc:acal:1.0:example:policyid:4"
[53]                  }
[54]              }
[55]          }
[56]      }
[57]  
```

```
[55]  }]
[56] }
```

[06] The nested policies are combined according to the `urn:oasis:names:tc:acal:1.0:combining-algorithm:deny-overrides` combining algorithm. This combining algorithm specifies that, if any nested policy evaluates to `Deny`, then the outer policy must return `Deny`; otherwise, if any policy evaluates to `Permit`, then the outer policy must return `Permit`.

[10] - [50] A nested policy that combines Rule 1, Rule 2 and Rule 3. Most of the content of the rules is omitted for clarity.

[13] The rules in the nested policy are also combined according to the `urn:oasis:names:tc:acal:1.0:combining-algorithm:deny-overrides` combining algorithm.

[14] - [30] The target restricts the applicability of the nested policy to decision requests for access to medical records belonging to Springfield Hospital.

[32] - [36] A copy of Rule 1.

[38] - [42] A copy of Rule 2.

[44] - [48] A copy of Rule 3.

[50] The end of the nested policy.

[52] - [54] The policy containing Rule 4 is included by reference.

---

## 7 Structures

### 7.1 Introduction

The structures in ACAL are described here in abstract terms, both in textual form and [UML] class diagrams. The concrete representations of these structures are defined for a variety of syntaxes each in a separate profile.

#### 7.1.1 Object Type

The ACAL structures are objects that each conform to a specific object type. Objects of the same kind conform to a specific, named object type, which describes a set of properties that an object of the type is permitted to have and for each property whether it is required or optional.

A property has a unique name (unique among the object's properties) and a value, where the value conforms to a specific type, either a simple type, an object type, or a sequence of such values.

**7.1.1.1 Object constraints** ACAL object types may have extra object constraints expressed in form of UML constraints (section 7.6 of [UML]) in the UML models of each ACAL type. These constraints may apply either to a single property of the object - so-called *property-level constraints* - or more globally on multiple properties of the object at once, i.e. *object-level constraints*.

As a reminder, a property is said *multi-valued* (or *multivalued*) if the upper bound of its multiplicity (UML) is (strictly) greater than one, else it is said *single-valued*.

**7.1.1.1.1 Property-level constraints** Property-level constraints are defined at the level of a specific property as an annotation between curly braces that is part of the UML property's description in the class diagram. In ACAL model, we use the following kinds of property-level constraints:

##### 7.1.1.1.1 UML native constraints

###### Order constraint:

For multi-valued properties only, the UML standard `ordered` (opposite `unordered`) annotation is used to express that the order of values should be preserved in the created collection.

###### Simple uniqueness constraint:

For multi-valued properties of primitive type only, the UML standard `unique` (opposite `nonunique`) annotation is used to express that each value of the property MUST be unique. Combined with the previous order constraint, this determines the type of collection to be used (cf. Table 7.1 of [UML]). For

example, the following constraint (`{unordered, unique}`) indicates that the order does not matter and each string value SHALL be unique, therefore a Set may be used as collection for the property *P*'s values:

```
+ P: String [*] {unordered, unique}
```

For uniqueness constraints on object-typed properties, see the next section.

#### 7.1.1.1.2 OCL constraints

More advanced constraints are expressed in [OCL] in the form `{OCL expression}` as specified by section 8.3.4.2 (and 8.3.5.2 for the example) of [UML]. Let refer to *prop* as the property to which the constraint is applied. In each OCL expression below, *prop* is used as the OCL context. Therefore, the `self` keyword in particular refers to *prop* itself.

##### Mandatory-property-based uniqueness constraint:

For multi-valued properties of complex (object) type, the previous simple uniqueness constraint is not enough to define what SHALL be unique about each value object. In that case, assuming each value of *prop* has a mandatory property *itemProp* of primitive type, we use the following OCL expression to specify that the property *itemProp* of each item SHALL be unique (in other words, *itemProp* is used as the unique key):

```
self->isUnique(itemProp)
```

##### Optional-property-based uniqueness constraint:

This is a variant of the previous constraint where the property used as unique key is optional, and therefore the OCL expression only checks that the non-null values of *itemProp* are unique and ignore the rest where *itemProp* is null/undefined:

```
self->select(itemProp <> null)->isUnique(itemProp)
```

##### Multi-property-based uniqueness constraint:

This is a variant of the first constraint in this section where the unique key consists of multiple properties (*itemProp1*, *itemProp2*, ...*itemPropN*) of each value of *prop*, all those properties have primitive type and at least *itemProp1* is a mandatory property. In this case, we use the following OCL expression to check that each value (i.e. each key based on those properties, i.e. it is a *compound key*) is unique:

```
self->isUnique(Sequence{itemProp1, itemProp2, ..., itemPropN})
```

##### Multivalued-property-based uniqueness constraint:

This is a variant where the unique key is based on a property *itemProp* which is itself multi-valued, and has its own unique key based on a single property *itemPropKey*. In this case, the key for each *item* in *prop*'s values that is checked for uniqueness is the sequence (if ordered) or set (if unordered) of all *itemPropKey* values, i.e. the following OCL expression:

```
self->isUnique(itemProp->collect(itemPropKey))
```

##### Value type uniqueness constraint:

This is a variant where the constraint only checks that the type of each value of *prop* is unique:

```
self->isUnique(oclType())
```

**7.1.1.2 Object-level constraints** ACAL object-level constraints are expressed in UML as a note attached by a dashed line to the UML class, as specified in [UML] Constraints section, and in OCL language. For such constraint, ACAL only uses OCL expressions of the form *X or Y* where *X* and *Y* are each one of the following kinds of simple expressions:

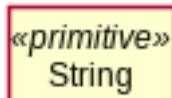
1. *prop*  $\neq$  null: true iff the single-valued property *prop* of the object is not null (defined).
2. *prop* = null: true iff the single-valued property *prop* of the object is null (undefined).
3. *prop*->NotEmpty(): true iff the multivalued property *prop* has at least one value (the collection of values is not empty).

## 7.1.2 Simple Types

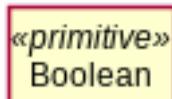
In UML, the simple types may be *primitive types* or *enumerations* (aka *enum types*). ACAL uses both UML standard primitive types and newly defined primitive and enumeration types, described in the next subsections.

**7.1.2.1 Standard UML primitive types (String, Boolean, Integer, Real)** Here are the simple types based on UML standard primitive types.

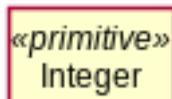
- **String:** A sequence of characters that may be constrained to a particular pattern. Defined in UML by UML standard primitive type **String**:



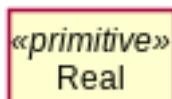
- **Boolean:** Either **true** or **false**. Defined in UML by UML standard primitive type **Boolean**



- **Integer:** An integer number that may be constrained to a particular range. Defined in UML by UML standard primitive type **Integer**:

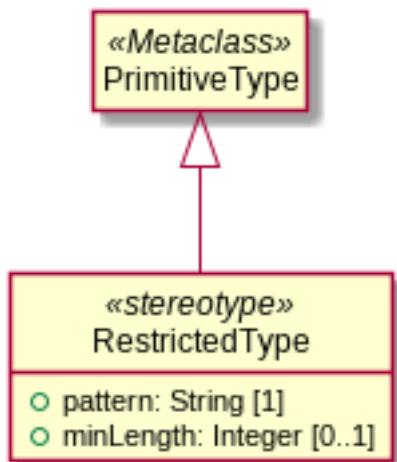


- **Real:** real number in UML, typically represented using a floating point standard such as ISO/IEC/IEEE 60559:2011 (whose content is identical to the predecessor IEEE 754 standard).



This primitive type is not directly used in the current ACAL model and provided for information purposes only. The subtype **Double** is used instead.

**7.1.2.2 ACAL-defined UML stereotype(s) used in new primitive types** **RestrictedString** is defined as a UML stereotype for String-derived primitive types defined in the next subsection and restricted by a pattern (regular expression) and an optional minimum length (positive integer), both specified as the stereotype's properties **pattern** and **minLength** respectively:



### 7.1.2.3 ACAL-defined simple types

**7.1.2.3.1 URI** A URI is a sequence of characters representing a Uniform Resource Identifier according to RFC 3986. Defined in UML as follows:

**«primitive»**  
**URI**

ACAL representation formats (in ACAL profiles) usually have a native equivalent for this type, in which case they should use that, e.g. XML schema `xs:anyURI`, JSON schema `{ "type": "string", "format": "uri" }`.

**7.1.2.3.2 NonNegativeInteger** A `NonNegativeInteger` is a positive `Integer`, equal to or greater than zero. Defined in UML as follows:

**«primitive»**  
**NonNegativeInteger**

ACAL representation formats (in ACAL profiles) may have a native equivalent for this type, in which case they should use that, e.g. XML schema `xs:nonNegativeInteger`, JSON schema `{ "type": "integer", "minimum": 0 }`.

**7.1.2.3.3 Double** A `Double` is a double-precision 64-bit IEEE 754 floating point represented in UML as follows:

**«primitive»**  
**Double**

In UML, it may be defined as a subtype of UML standard primitive type `Real`, restricted to 64-bit length. However, each ACAL representation format should map this to their closest native equivalent if there is any, e.g. `xs:double` in XML.

**7.1.2.3.4 VersionType** A `VersionType` value specifies the version number of a policy or shared variable, and is defined in UML as follows:

**«primitive»**  
**«restrictedString»**  
**VersionType**

**«restrictedString»**  
`pattern="^([0|[1-9]\d*)(\.(0|[1-9]\d*)){0,3}$"`

The version number is expressed as a sequence of decimal numbers, each separated by a period (.), which is compatible with Semantic Versioning.

**7.1.2.3.5 VersionMatchType** Properties of this type SHALL contain a restricted regular expression matching a version number (see Section 7.1.2.3.3). The expression SHALL match versions of a referenced policy that are acceptable for inclusion in the referencing policy.

UML definition:

«primitive»
«restrictedString»
VersionMatchType

«restrictedString»
pattern="^(0 [1-9]\d*\ *)(\.(0 [1-9]\d*\ *\ +)){0,3}\$"

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.+.

**7.1.2.3.6 ShortIdNameType** ShortIdNameType is the primitive type for Short Identifier names (see Section 7.3 ).

UML definition:

«primitive»
«restrictedString»
ShortIdNameType

«restrictedString»
pattern="^([A-Za-z][0-9A-Za-z]*(-[0-9A-Za-z]+)*\$"

**7.1.2.3.7 ShortIdValueType** ShortIdValueType is the primitive type for Short Identifier values (see Section 7.3 ). Such value is an alternating sequence of URI characters and \*{ShortId}\*s (short identifier names enclosed in curly braces). Reminder: the curly brace is not a valid URI character.

UML definition:

«primitive»
«restrictedString»
ShortIdValueType

«restrictedString»
pattern="^(!#;=?-\{\}_a-z)*(\{([A-Za-z][0-9A-Za-z]*(-[0-9A-Za-z]+)*\}(!#;=?-\{\}_a-z)*\$"
minLength=1

The pattern is: ^uc\*(\{s\}uc\*)\*\$ where:

- s is the pattern for a ShortId name, copied from the ShortIdNameType definition (which excludes curly braces).
- uc is the pattern for a valid URI character based on RFC 3986 (cf. the ABNF definition of 'URI' in Appendix A), which excludes curly braces: [!#-;=?-\{\}\_a-z~] The minLength=1 restriction prevents the empty string, i.e. there is at least one uc or {s}.

**7.1.2.3.8 IdentifierType** A value of the IdentifierType simple type refers to a specific attribute category, attribute, data type, function, notice, status code, combining algorithm.

UML definition (class diagram):

«primitive» «restrictedString» IdentifierType
«restrictedString» pattern='^["{}"]*([A-Za-z][0-9A-Za-z]*(-[0-9A-Za-z]+)*["{}"]*)*\$'

A value of this simple type is either:

- an absolute URI [RFC2396],
- the name of a short identifier or
- a character string with one or more short identifier names enclosed by curly brackets (i.e., { and }; U+007B and U+007D) optionally preceded, followed and/or separated by other characters allowed in a URI.

A short identifier name appearing in the second and third cases MUST be the name of a member of a short identifier set referenced by the containing policy, request or response.

Note that the three cases can be distinguished from each other syntactically in a valid, correctly-formatted value. If the value contains curly brackets, then the third case must apply since the curly bracket characters are not legal characters for an absolute URI or a short identifier name; otherwise, if the value matches the pattern for a short identifier name, then the second case applies since an absolute URI begins with a scheme name and a colon character (i.e., :; U+003A) and the colon character is not a legal character for a short identifier name; otherwise, the value is an absolute URI.

The conversion of a value of the IdentifierType simple type into an absolute URI is detailed in Section 8.3.

**7.1.2.3.9 LocalIdentifierType** LocalIdentifierType values are local identifiers, i.e. identifiers that are unique only within a limited scope. Examples of local identifiers are Request-local identifiers (e.g. the Id properties of RequestEntity and RequestEntityReference), Policy-local identifiers (e.g. the VariableId of a policy's variable or the ParameterName of a policy parameter or the RuleId of a policy's rule), Rule-local identifiers (e.g. the VariableId of a rule's variable), etc.

-->

UML definition (class diagram):

«primitive» «restrictedString» LocalIdentifierType
«restrictedString» pattern='^[_*[A-Za-z][A-Za-z_0-9]*([-_*[A-Za-z_0-9])*\$_'

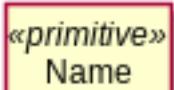
**7.1.2.3.10 AttributeSelectorPathType** The AttributeSelectorPathType defines valid values for the Path property of BaseAttributeSelectorType objects (non-empty strings without leading/trailing whitespace).

UML definition (class diagram):

«primitive» «restrictedString» AttributeSelectorPathType
«restrictedString» pattern='^\S(.*\S)?\$'

**7.1.2.3.11 Name** The **Name** type is the same as XSD standard **Name** datatype.

UML definition:

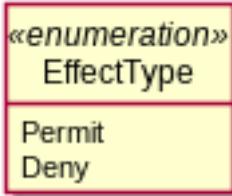


Although an XML representation may use the standard XSD **Name** datatype as is for this type, other ACAL representation formats need an alternative representation. One suggestion is to use the string type restricted by a pattern (regular expression) matching the XSD **Name**'s definition, i.e. matching XSD pattern `\i\c*`. Non-XML representation formats do not usually use the XSD regex (regular expression) flavor but other flavors such as ECMA 262 (JavaScript) flavor in the case of JSON, therefore the pattern - the character classes `\i` and `\c` in particular - must be translated to the corresponding regex flavor. The following equivalence table may be used as a convenience to build the full pattern in non-XML ACAL representation formats:

XSD character class	Javascript regex for full Unicode support
<code>\i</code>	<code>[:A-Z_a-z\u00C0-\u00D6\u00D8-\u00F6\u00F8-\u02FF\u0370-\u037D\u037F-\u1FFF\u200C-\u200E]</code>
<code>\c</code>	<code>[-.0-9:A-Z_a-z\u00B7\u00C0-\u00D6\u00D8-\u00F6\u00F8-\u037D\u037F-\u1FFF\u200C-\u200E]</code>

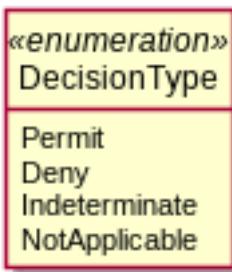
**7.1.2.3.12 EffectType** The **EffectType** simple type defines the values allowed for the **Effect** property of a **RuleType** object and for the **AppliesTo** property of the **NoticeExpressionType** objects; either **Permit** or **Deny**.

UML definition:



**7.1.2.3.13 DecisionType** A **DecisionType** value indicates the result of policy evaluation.

UML definition:



The values of **DecisionType** 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.

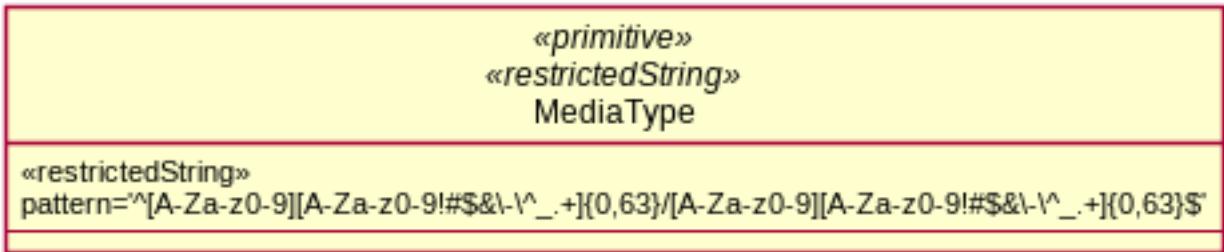
**NotApplicable**: the PDP does not have any policy that applies to this decision request.

**7.1.2.3.14 MediaType (optional)** *Support for this type is optional, required only if ContentType objects MUST be supported.*

**MediaType** is the primitive type for **ContentType** objects' **MediaType** property (see Section 7.34 ).

The regular expression below follows the ABNF syntax from section 4.2 of [RFC6838] (with the recommendation that <type-name> and <subtype-name> SHOULD be limited to 64 characters).

UML definition:

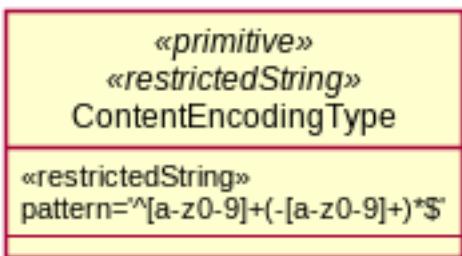


This is the generic type definition. However, the exact possible values SHALL be defined explicitly by the `AttributeSelector` Profile(s) of ACAL, e.g. the XPath and/or JSONPath Profiles, depending on the Content Type(s) they support. In any case, the standard identifiers specified in Annex D.10 SHOULD be used whenever applicable, or else a media type registered at IANA according to [RFC6838].

#### 7.1.2.3.15 ContentEncodingType (optional) *Support for this type is optional, required only if ContentType objects MUST be supported.*

`ContentEncodingType` is the primitive type for `ContentType` objects' `Encoding` property (see Section 7.34 ).

UML definition:



This is the generic type definition. However, the exact possible values SHALL be defined explicitly by the Representation Profile(s) of ACAL (XACML, JACAL, etc.) for each supported Content type. In any case, the standard identifiers specified in Annex D.11 SHOULD be used whenever applicable, or else one of the standard *Transfer Encodings* names registered at IANA according to [RFC4289]. Or else, if a custom content encoding is really needed, as suggested by [RFC2045] Section 6.3, define a new one prefixed with `x-`, e.g. `x-my-new-encoding` (and matching the pattern above).

### 7.1.3 Relationship to Concrete Representations

ACAL model mappings to a concrete representation are defined in the relevant ACAL Profile for that representation format, e.g. in the XML Profile of ACAL (XACML) for the XML representation.

Examples of such mappings:

- In the JSON representation, an object maps to a JSON object and a property of that object maps to a JSON member (a name/value pair). An object type is formally defined by a JSON subschema.
- In the XML representation, a property corresponds to an XML element or XML attribute, an object corresponds to element content and an object type is formally defined by an XML Schema complex type.

## 7.2 ShortIdSetType

A `ShortIdSetType` object defines a set of short identifiers. It contains its own short identifiers as well as references to other short identifier sets, which are imported (recursively) into this set.

UML definition (class diagram):

**«*dataType*»**  
**ShortIdSetType**

- **Id:** URI [1]
- **ShortIdSetReference:** URI [\*] {unordered, unique}
- **ShortId:** ShortIdType [\*] {ordered, unique} {{OCL} self->isUnique(Name)}

A ShortIdSetType object contains the following properties:

**Id [Required]**

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

**ShortIdSetReference [Any Number]**

A sequence of URI values referencing other short identifier sets. The short identifiers of the referenced sets are included in this set. This applies recursively to the sets referenced by a referenced set. This set SHALL NOT reference itself and SHALL NOT reference a set that directly or indirectly references this set. This set SHALL NOT directly or indirectly reference any other set more than once.

**ShortId [Any Number]**

A sequence of ShortIdType objects, each defining a short identifier in this set.

A predefined set of short identifiers for the URIs defined in this specification is provided as a convenience, in both XML (`acal-core-xml-v4.0-identifiers.xml`) and JSON (`acal-core-xml-v4.0-identifiers.json`) representations. Use of this set is OPTIONAL.

## 7.3 ShortIdType

A ShortIdType object defines a single short identifier. Attribute categories, attributes, data types, functions, combining algorithms and other artifacts in ACAL are ultimately identified by URIs. A short identifier provides a simple alias name to use when composing and displaying policies and protocol messages instead of using the full URI.

UML definition (class diagram):

**«*dataType*»**  
**ShortIdType**

- **Name:** ShortIdNameType [1]
- **Value:** ShortIdValueType [1]

A ShortIdType object contains the following properties:

**Name [Required]**

A simple alias name that stands for a URI or a part thereof.

**Value [Required]**

The character string used to replace the simple alias name when a value of IdentifierType is evaluated to produce a complete URI; see Section 8.3.

The value of the Value property SHALL be either one of the following:

- a character string with only characters allowed in a URI [RFC3986] (curly brackets are not allowed) or
- a character string with one or more references to other short identifiers in the form of the short identifier name enclosed in curly brackets (i.e., { and }; U+007B and U+007D) optionally preceded, followed and/or separated by other characters allowed in a URI (this excludes curly brackets, which means that the curly bracket characters are not permitted in the value other than to enclose a short identifier name). In addition, the following restrictions apply:
  - the value SHALL NOT reference this short identifier (i.e., shall not contain any {s} where s matches the value of the Name property),

- the value SHALL NOT reference a short identifier that directly or indirectly references this short identifier and
- the referenced short identifiers MUST each either precede this short identifier in the same short identifier set or be defined in one of the short identifier sets referenced by the containing short identifier set.

The name of the short identifier MUST NOT be the same as the name of any other short identifier in the same short identifier set or the same as the name of any other short identifier defined in one of the short identifier sets referenced by the containing short identifier set.

## 7.4 PolicyType

A **PolicyType** object describes a policy: an aggregation of rules and other policies. Policies MAY be included in an enclosing **PolicyType** object either directly using the **Policy** property or indirectly using the **PolicyReference** property.

A **PolicyType** object may be evaluated, in which case the evaluation procedure defined in Section 8.12 SHALL be used.

The main components of this object type are the **CombiningAlgId**, **Target**, **Policy**, **Rule**, **VariableDefinition** and **NoticeExpression** properties.

If a **PolicyType** object contains references to other policies in the form of URLs, then these references MAY be resolvable.

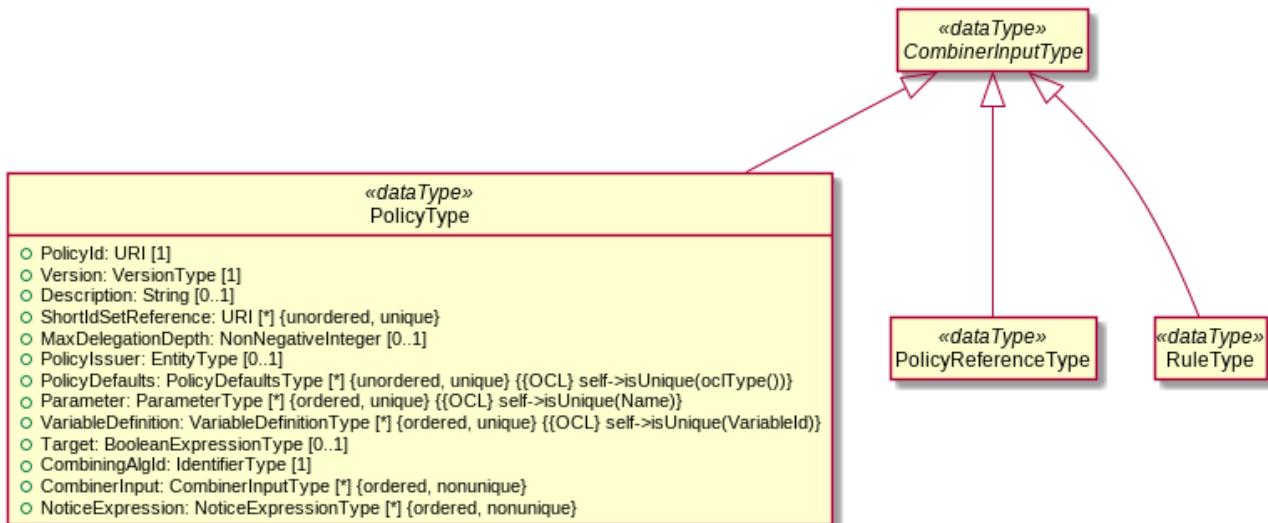
The results of evaluating the policies and rules included in a **PolicyType** object MUST be combined using the algorithm identified by the **CombiningAlgId** property.

A **PolicyType** object MAY contain a **PolicyIssuer** property. The interpretation of the **PolicyIssuer** property is explained in the separate administrative policy profile [XACMLAdmin].

The **Target** property defines the applicability of the **PolicyType** object to a set of decision requests. If the **Target** property within the **PolicyType** object matches the request context, then the **PolicyType** object may be used by the PDP in making its authorization decision. See Section 8.12.

Any **NoticeExpressionType** objects may be evaluated into notices by the PDP. Any resulting obligation notice (with **isObligation="true"**) MUST be fulfilled by the PEP in conjunction with the authorization decision. If the PEP does not understand or cannot fulfill any of the obligation notices, then it MUST act according to the PEP bias. See Section 8.2 and Section 8.16. Any resulting advice notice MAY be safely ignored by the PEP.

UML definition (class diagram):



A **PolicyType** object contains the following properties:

### **PolicyId** [Required]

A URI value specifying an identifier for the policy. 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]**

A VersionType value specifying the version number of the Policy.

**CombiningAlgId [Required]**

An IdentifierType value identifying the combining algorithm by which the PolicyType and RuleType objects MUST be combined. Standard combining algorithms are listed in Annex E. Standard combining algorithm identifiers are listed in Annex D.9.

**MaxDelegationDepth [Optional]**

If present, an Integer that limits the depth of delegation which is authorized by this policy. See the delegation profile [XACMLAdmin].

**ShortIdSetReference [Any Number]**

A sequence of URI values referencing short identifier sets. Each value MUST be unique in the sequence. The short identifiers used by the policy MUST be ones defined in the referenced sets or in any further sets referenced by the referenced sets (recursively). The policy SHALL NOT directly or indirectly reference any short identifier set more than once.

**Description [Optional]**

A free-form String description of the policy.

**PolicyIssuer [Optional]**

An EntityType object containing attributes of the issuer of the policy. See the *EntityType* section for more details on the structure, and see the *Administration and Delegation profile* [XACMLAdmin] for a particular use of this object. A PDP which does not implement this profile MUST report an error or return an Indeterminate result if it encounters this object.

**PolicyDefaults [Any Number]**

sequence of PolicyDefaultsType objects containing each a set of default values specific to a particular ACAL Profile, applicable to the policy (e.g. ACAL XPath Profile's default XPath version). In particular, each object SHALL have a different concrete type (per ACAL profile). The scope of the PolicyDefaults property SHALL be the enclosing policy. The use of PolicyDefaults property is specified by particular ACAL Profiles (e.g. XPath Profile).

**Parameter [Any Number]**

A sequence of ParameterType objects declaring the parameters of the policy if the policy is parameterized. Each object's Name MUST be unique in this sequence.

**VariableDefinition [Any Number]**

A sequence of VariableDefinitionType objects, each defining an expression that can be referenced from anywhere in a contained policy or rule where an expression can appear. Each object's VariableId must be unique in this sequence and in the list of VariableDefinitions in scope, i.e. already defined in an enclosing policy (parent or ancestor).

**Target [Optional]**

A BooleanExpressionType object that determines the set of decision requests to which the parent policy is applicable. If this property is omitted from the PolicyType object, then the policy applies to all decision requests. Evaluation of the Target property is described in Section 8.7. The Target expression MUST NOT be a literal boolean value (ValueType object).

**NoticeExpression [Any Number]**

A sequence of NoticeExpressionType objects to be evaluated into notices by the PDP. See Section 7.29. See Section 8.16 for a description of how the notices to be returned by the PDP are determined.

**CombinerInput [Any Number]**

An ordered sequence of CombinerInputType objects, which are combined according to the combining algorithm specified by the CombiningAlgId property.

Each CombinerInputType object contains exactly one of the following properties:

**Policy**

A PolicyType object defining a nested policy that is included in this policy. A policy whose target matches the decision request MUST be evaluated. A policy whose target does not match the decision request SHALL be ignored.

**PolicyReference**

A PolicyReferenceType object referencing a separate policy that is included in this policy. If the Id property of the PolicyReferenceType object is a URL, then it MAY be resolvable. A policy whose target

matches the decision request MUST be considered. A policy whose target does not match the decision request SHALL be ignored.

#### Rule

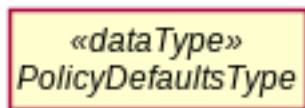
A RuleType object defining a nested rule that is included in this policy. A rule whose condition matches the decision request MUST be considered. A rule whose condition does not match the decision request SHALL be ignored.

## 7.5 PolicyDefaultsType (optional)

*Supporting this part is optional. It is required only for supporting ACAL Profiles that define extensions (subtypes) of PolicyDefaultsType (e.g. XPath Profile).*

PolicyDefaultsType is an abstract object type for default values that apply to the parent PolicyType object. Concrete subtypes of PolicyDefaultsType are defined in separate ACAL Profiles, e.g. XPath Profile.

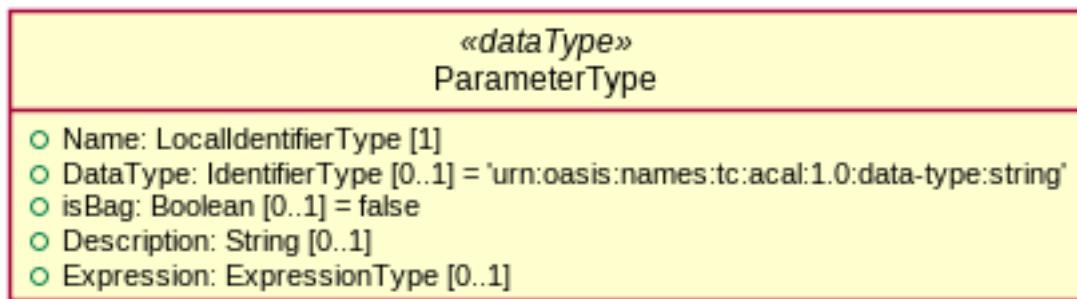
UML definition (class diagram):



## 7.6 ParameterType

A ParameterType object declares a single parameter for a parameterized policy or shared variable. Parameters specify arguments that may be passed to either a Policy from a PolicyReference, or a Shared Variable from a SharedVariableReference. This enables other Policies (respectively Expressions) to pass dynamic (Expression-based) values to another Policy (respectively SharedVariable) during the evaluation, according to its Parameter definitions (ParameterType objects), in the same way as calling a function according to the function signature.

UML definition (class diagram):



A ParameterType object contains the following properties:

#### Name [Required]

A LocalIdentifierType value to name the parameter. The value of the parameter MAY be referenced within an expression contained within the policy using a VariableReferenceType object with its VariableId property set to this name.

#### DataType [Optional, Default urn:oasis:names:tc:acal:1.0:data-type:string]

An IdentifierType value specifying the data type of the parameter. If this property is omitted, then it is treated as being set to urn:oasis:names:tc:acal:1.0:data-type:string.

#### IsBag [Optional, Default false]

A Boolean value that specifies whether the parameter takes a single value (set to **false**) or a bag of values (set to **true**). If this property is omitted, then it is treated as being set to **false**.

#### Description [Optional]

A free-form String description of the parameter.

#### Expression [Optional]

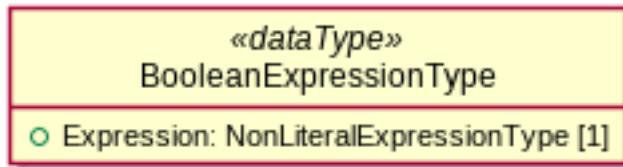
An expression that evaluates to a default value (if Isbag is **false**) or bag of values (if IsBag is true) for the parameter. This default value is used only if no argument is provided as input for this parameter (from

a `PolicyReference` or `SharedVariableReference`). If the Expression value is a `ValueType` object, i.e. a literal value, the latter SHALL not redefine/override the `DataType` identifier already defined by the above `DataType` property, and SHOULD not have any `DataType` identifier property set at all, if possible.

## 7.7 BooleanExpressionType

A `BooleanExpressionType` object contains one ACAL expression, with the restriction that the expression's return data type MUST be `urn:oasis:names:tc:acal:1.0:data-type:boolean`.

UML definition (class diagram):



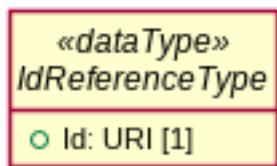
The `Target` and `Condition` properties are of this type.

Expression evaluation is described in Section 8.5.

## 7.8 IdReferenceType

`IdReferenceType` is an abstract type used for referencing policies by their policy identifier (URI). This has two subtypes for matching also a specific policy version: `ExactMatchIdReferenceType`, `PatternMatchIdReferenceType`.

UML definition (class diagram):



The `IdReferenceType` object type contains the following property:

### Id [Required]

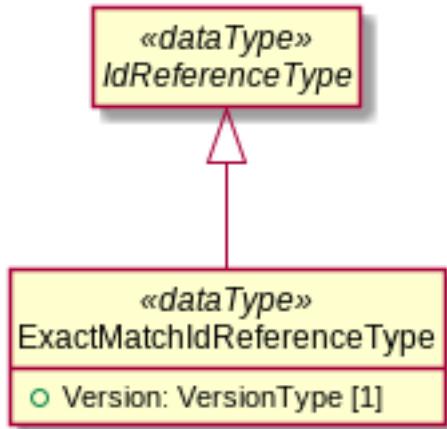
A URI being the `PolicyId` of the referenced policy. If the URI is a URL, then it MAY be resolvable to the policy. However, the mechanism for resolving a policy reference to the corresponding policy is outside the scope of this specification.

## 7.9 ExactMatchIdReferenceType (optional)

*Support for this object type is optional, required only if the `ReturnPolicyIdList` property in the Request is supported.*

An `ExactMatchIdReferenceType` object is a type of policy reference that matches both the identifier and a specific (fixed) version of a policy. In a `ApplicablePolicyReference`, it is used to identify a policy that has been applicable to a request.

UML definition (class diagram):



The `ExactMatchIdReferenceType` object type extends the `IdReferenceType` object type (Section 7.8) with the following property:

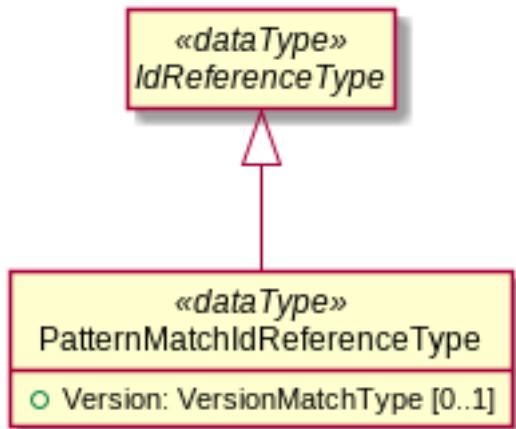
#### **Version [Required]**

A `VersionType` value indicating the version of a policy that was applicable to the request. See `PolicyReferenceType` (Section 7.11).

## 7.10 PatternMatchIdReferenceType

A `PatternMatchIdReferenceType` object is a type of policy reference that matches both the identifier and one or more versions of a policy using version patterns.

UML definition (class diagram):



The `PatternMatchIdReferenceType` object type extends the `IdReferenceType` object type with the following properties:

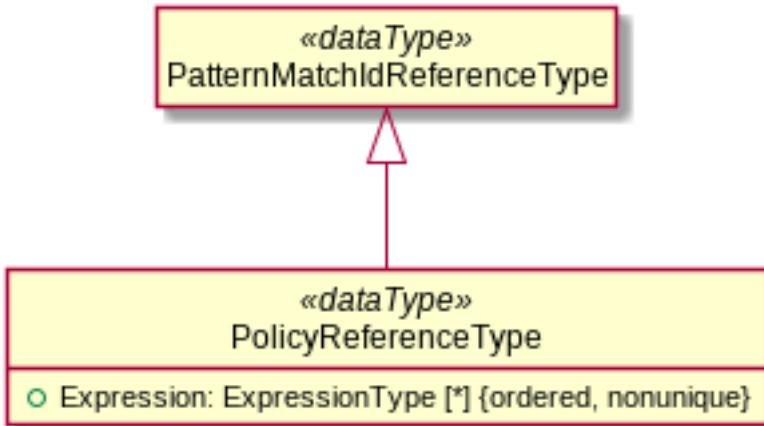
#### **Version [Optional]**

Specifies a matching expression for selecting an acceptable version of the referenced policy. The matching operation is defined by `VersionMatchType` (Section 7.1.2.3.4). If this property is present, then the selected version of the policy MUST match the expression. If the property is absent, then any version of the policy is acceptable. In the case that more than one version matches, then the most recent one (latest version) SHOULD be used.

## 7.11 PolicyReferenceType

A `PolicyReferenceType` object is used to reference a policy by identifier and version.

UML definition (class diagram):



The **PolicyReferenceType** object type extends the **PatternMatchIdReferenceType** object type with the following properties:

#### Expression [Any Number]

Arguments for a parameterized policy, in the same order as the **Parameter** values in the referenced policy, and each argument number  $n$  MUST match the definition of the **Parameter** number  $n$  (**DataType**, **isBag**). The number of arguments  $N$  may be less than the number of declared **Parameters** if and only if the **Parameters** number  $N$  (starting at zero) and above have defined default values, in which case they are used as remaining arguments.

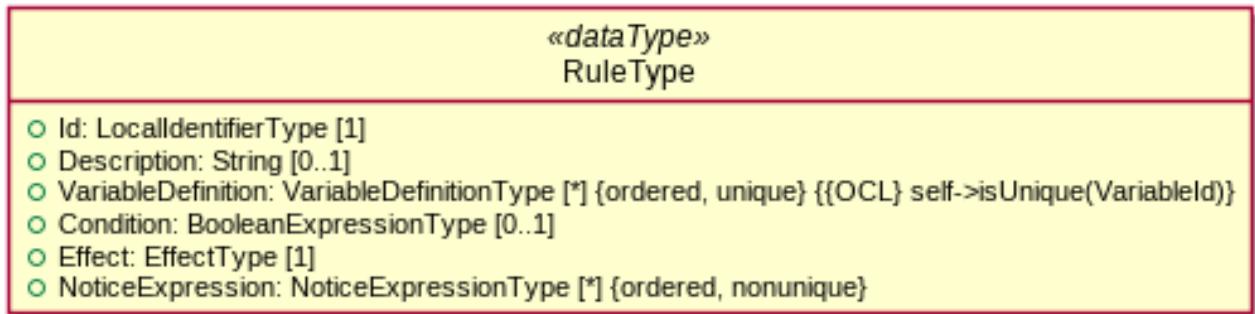
**DataType inference rule:** if an Expression has an (optional) **DataType** property (statically-typed Expression), i.e. its type is **ValueType**, **NamedAttributeDesignatorType**, **BaseAttributeSelectorType** or any subtype thereof, then the Expression's **DataType** property SHOULD be omitted in this case as it is already defined by the corresponding **Parameter**'s definition in the referenced **PolicyType** object. Else it SHALL match that **Parameter**'s **DataType**.

## 7.12 RuleType

A **RuleType** object defines an individual rule in a policy. The main components of an object of this type are the **Condition**, **NoticeExpression** and **Effect** properties.

A **RuleType** object may be evaluated, in which case the evaluation procedure defined in Section 8.11 SHALL be used.

UML definition (class diagram):



A **RuleType** object contains the following properties:

#### Id [Required]

A **LocalIdentifierType** value to identify this rule. It is the responsibility of the PAP to ensure that no two rules in the same policy have the same identifier.

#### Effect [Required]

An **EffectType** value specifying the effect of the rule when its condition is satisfied; either **Permit** or **Deny**.

#### Description [Optional]

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

#### **VariableDefinition [Any Number]**

A sequence of **VariableDefinitionType** objects, each defining a variable - with a value expression - that can be referenced from anywhere in the rule where an Expression can appear. Each object's **VariableId** MUST be unique in this sequence and in the set of **VariableDefinitions** in scope, i.e. MUST NOT override any other defined variable in an enclosing policy (parent or ancestor).

#### **Condition [Optional]**

A **BooleanExpressionType** object holding an expression that MUST be satisfied for the rule to be assigned its **Effect** value. Evaluation of the **Condition** property is described in Section 8.9. The Condition expression MUST NOT be a literal boolean value (**ValueType** object).

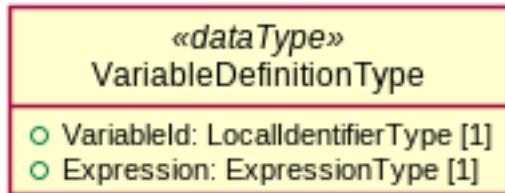
#### **NoticeExpression [Any Number]**

A sequence of **NoticeExpressionType** objects, each defining a notice expression potentially evaluated into a notice by the PDP. See **NoticeExpressionType**. Each object's **Id** MUST be unique in this sequence. See Section 8.16 for a description of how the notices to be returned by the PDP shall be determined. See Section 8.2 about enforcement of obligation notices.

### **7.13 VariableDefinitionType**

A **VariableDefinitionType** object is used to define a value or a bag of values that can be referenced by zero or more **VariableReferenceType** objects. **VariableDefinitionType** objects appear in the **VariableDefinition** property of a **PolicyType** or **RuleType** object. This is the parent object. The scope within which the variable definition can be referenced is any expression within the parent object, including any nested **PolicyType** or **RuleType** objects.

UML definition (class diagram):



A **VariableDefinitionType** object has the following properties:

#### **VariableId [Required]**

A **LocalIdentifierType** value to identify the variable definition. The value of the property MUST NOT be the same as the **VariableId** property of another **VariableDefinitionType** object in the same **VariableDefinition** property or in the **VariableDefinition** property of a **PolicyType** object enclosing the parent object.

#### **Expression [Required]**

An **ExpressionType** object. The value of the variable definition is the result of evaluating the **ExpressionType** object. The expression MAY contain **VariableReferenceType** objects referring to other **VariableDefinitionType** objects provided those objects are in the same **VariableDefinition** property or in the **VariableDefinition** property (list) of a **PolicyType** object enclosing the parent object. The expression SHALL NOT contain **VariableReferenceType** objects that refer to this **VariableDefinitionType** object or that refer to **VariableDefinitionType** objects that directly or indirectly refer to this **VariableDefinitionType** object.

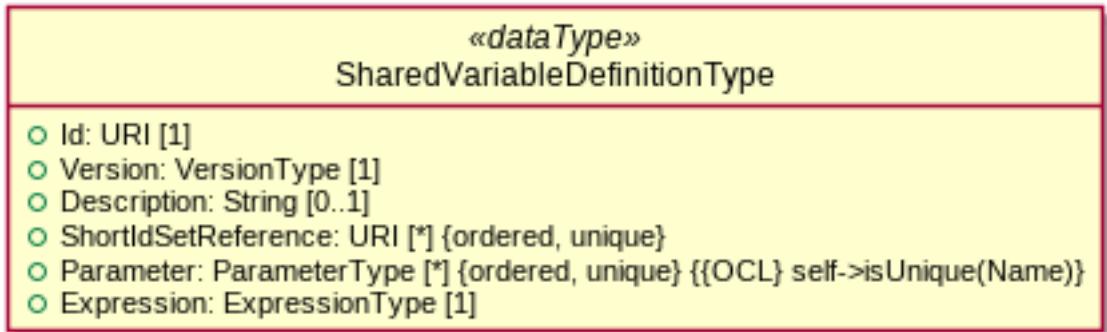
If the **Expression** value is a **ValueType** object (literal value), it SHALL be handled as String (with **Data** type identifier `urn:oasis:names:tc:acal:1.0:data-type:string`) by default. Note that the way the **Data** type identifier is defined depends on the concrete type (subtype of **ValueType**) of that value. For example, **ValueType** subtypes stereotyped `<<fixedDatatype>>` have the **Data** type identifier globally defined and fixed as a Stereotype property (therefore implicitly predefined for every instance), whereas the **LiteralRestrictedStringType** have a UML attribute **Data** type that can be set to a different value (identifier) for each instance.

### **7.13b SharedVariableDefinitionType**

A **SharedVariableDefinitionType** object is used to define a value or a bag of values that can be referenced by zero or more **SharedVariableReferenceType** objects. **SharedVariableDefinitionType** objects appear

in the `SharedVariableDefinition` property of a `BundleType` object, i.e., they exist independently of any policy. The scope within which the shared variable definition can be referenced is any expression nested within the `BundleType` object. This includes policies, any other `SharedVariableDefinitionType` objects and the arguments of the `BundleType` object's `PolicyReference` property.

UML definition (class diagram):



#### **Id [Required]**

A URI value specifying an identifier for the shared variable. It is the responsibility of the PAP to ensure that no two shared variables visible to the PDP have the same identifier. This MAY be achieved by following a predefined URN or URI scheme. If the identifier is in the form of a URL, then it MAY be resolvable.

#### **Version [Required]**

A `VersionType` value specifying the version number of the shared variable.

#### **Description [Optional]**

A free-form `String` description of the shared variable.

#### **ShortIdSetReference [Any Number]**

A sequence of URI values referencing short identifier sets. The short identifiers used by the shared variable MUST be ones defined in the referenced sets or in any further sets referenced by the referenced sets (recursively). The shared variable SHALL NOT directly or indirectly reference any short identifier set more than once.

#### **Parameter [Any Number]**

A sequence of `ParameterType` objects declaring the parameters of the shared variable if the variable is parameterized. Each object's `Name` MUST be unique in this sequence.

#### **Expression [Required]**

An `ExpressionType` object. The value of the shared variable definition is the result of evaluating this `ExpressionType` object. The expression MAY contain `SharedVariableReferenceType` objects referring to other `SharedVariableDefinitionType` objects. The expression SHALL NOT contain `SharedVariableReferenceType` objects that refer to this `SharedVariableDefinitionType` object or that refer to `SharedVariableDefinitionType` objects that directly or indirectly refer to this `SharedVariableDefinitionType` object.

Note that the expression cannot contain `VariableReferenceType` objects since the `SharedVariableDefinitionType` object is not in the scope of any `VariableDefinitionType` object.

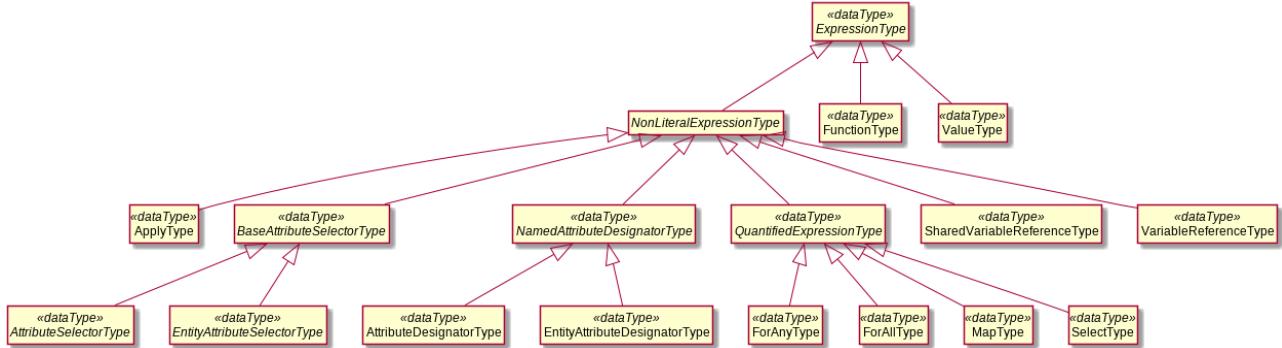
If an Expression has an (optional) `DataType` property (statically-typed Expression), i.e. its type is `ValueType`, `NamedAttributeDesignatorType`, `BaseAttributeSelectorType` or any subtype thereof, and its `DataType` property is omitted, it is set implicitly to `urn:oasis:names:tc:acal:1.0:data-type:string` by default.

*Note that the way the `DataType` identifier is defined on a `ValueType` depends on the concrete type (`ValueType` subtype) of that value. For example, `ValueType` subtypes stereotyped `<<fixedDatatype>>` have the `DataType` identifier globally defined and fixed as a Stereotype property (therefore implicitly predefined for every instance), whereas the `LiteralRestrictedStringType` have a UML attribute `DataType` that can be set to a different value (identifier) for each instance.*

## 7.14 ExpressionType

An `ExpressionType` object defines an ACAL expression. Expression evaluation is defined in Section 8.5.

UML definition (class diagram):



An `ExpressionType` object is either a `NonLiteralExpressionType` object or an object that contains exactly one of the following properties:

### Function

A `FunctionType` object identifying a function.

### Value

A `ValueType` object specifying a literal ACAL value.

An `NonLiteralExpressionType` object contains exactly one of the following properties:

### Apply

An `ApplyType` object specifying a function call.

### AttributeDesignator

An `AttributeDesignatorType` object specifying a named attribute from the request context.

### EntityAttributeDesignator

An `EntityAttributeDesignatorType` object specifying a named attribute from either the request context or a value of the `urn:oasis:names:tc:acal:1.0:data-type:entity` data type.

### AttributeSelector

An `AttributeSelectorType` object specifying Path expression - depending on the content type (e.g. XPath for XML content) - to apply to the structured content in the request context to produce a bag of ACAL values.

### EntityAttributeSelector

An `EntityAttributeSelectorType` object specifying an Path expression - depending on the content type (e.g. XPath for XML content) - to apply to the structured content either in the request context or in a value of the `urn:oasis:names:tc:acal:1.0:data-type:entity` data type.

### VariableReference

A `VariableReferenceType` object referencing a variable.

### SharedVariableReference

A `SharedVariableReferenceType` object referencing a shared variable.

### ForAny

A `QuantifiedExpressionType` object specifying existential quantification.

### ForAll

A `QuantifiedExpressionType` object specifying universal quantification.

### Select

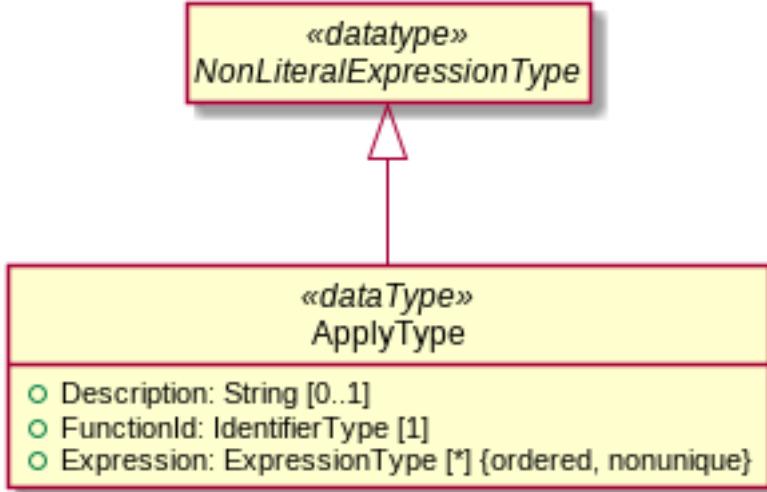
A `QuantifiedExpressionType` object specifying filtering of an input bag of values.

`Map` A `QuantifiedExpressionType` object specifying transformation of an input bag of values.

## 7.15 ApplyType

An **ApplyType** object is a kind of expression that denotes application of a function to its arguments, which are also expressions, thus encoding a function call.

UML definition (class diagram):



An **ApplyType** object contains the following properties:

**FunctionId [Required]**

An **IdentifierType** value identifying the function to be applied to the arguments. ACAL-defined functions are described in Annex C.3.

**Description [Optional]**

A free-form **String** description of the **ApplyType** object.

**Expression [Any Number]**

A sequence of **ExpressionType** objects, each defining an expression as an argument to the function.

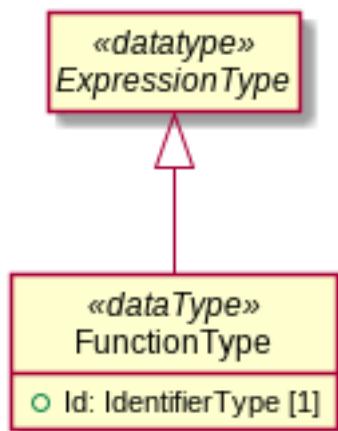
**Data Type inference rule:** if an Expression has an (optional) **Data Type** property (statically-typed Expression), i.e. its type is **ValueType**, **NamedAttributeDesignatorType**, **BaseAttributeSelectorType** or any subtype thereof, then the Expression's **Data Type** property MAY be omitted in this case if the corresponding parameter's definition in the function's signature (the function identified by **FunctionId**) has a fixed **Data Type** (which is the case for most ACAL functions). Else it SHALL match the function parameter's **Data Type**.

If one of these is a **ValueType** object and the function's signature defines explicitly the data-type of the corresponding argument, then this object does not need to (re)declare any **Data Type** identifier as it is already defined accordingly. In particular, if the concrete type (**ValueType** subtype) of the value has an (optional) **Data Type** identifier property (e.g. **LiteralRestrictedStringType**), then the value SHALL leave this attribute unset.

## 7.16 FunctionType

A **FunctionType** object is a kind of expression used to identify a function as an argument to a higher-order function defined by the argument's parent **ApplyType** object.

UML definition (class diagram):



A `FunctionType` object contains the following property:

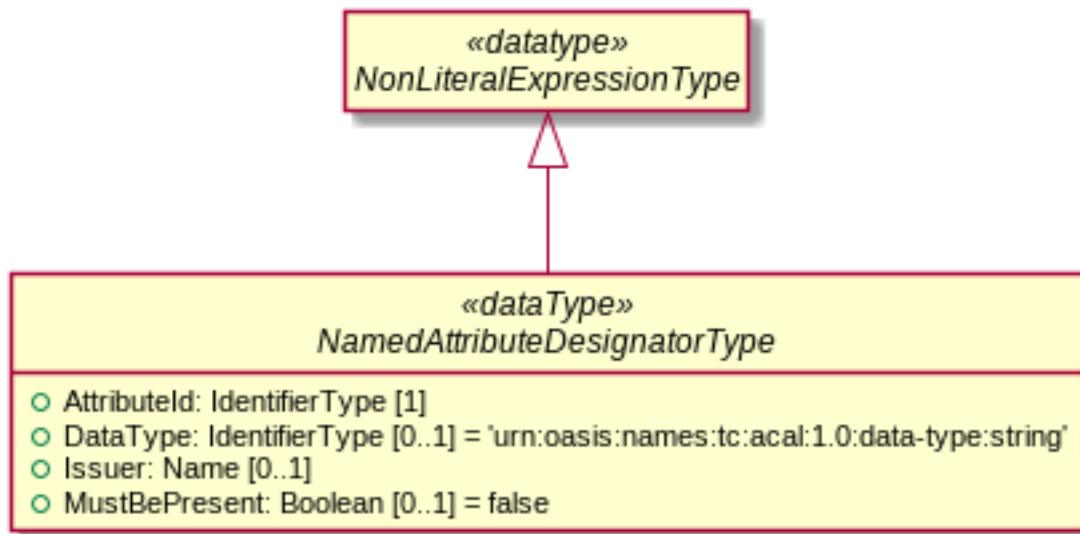
**Id [Required]**  
An `IdentifierType` value identifying a function.

## 7.17 NamedAttributeDesignatorType

The `NamedAttributeDesignatorType` object type is an abstract object type that specifies a named attribute for the retrieval of a bag of attribute values from a source of attributes. The object types derived from `NamedAttributeDesignatorType` determine the source of attributes: an `AttributeDesignatorType` object (an attribute designator) specifies a named attribute in the request context and an `EntityAttributeDesignatorType` object (an entity attribute designator) specifies a named attribute in either the request context or a value of the `urn:oasis:names:tc:acal:1.0:data-type:entity` data type.

Evaluation of an `AttributeDesignatorType` or `EntityAttributeDesignatorType` object returns a bag containing all the ACAL attribute values that are matched by the named attribute, or an error. In the event that no matching attribute is present in the source of attributes, the `MustBePresent` property governs whether this evaluation returns an empty bag or `Indeterminate`. See Section 8.4.5.

UML definition (class diagram):



A `NamedAttributeDesignatorType` object contains the following properties:

**AttributeId [Required]**  
An `IdentifierType` value specifying the `AttributeId` of the named attribute.

**DataType [Optional]**  
An `IdentifierType` value specifying the `DataType` of the named attribute. The bag returned by the designator SHALL contain values of this data type. If this property is omitted, the `DataType` inference rule(s) of the parent object SHALL apply first, if there is any rule defined in this specification in the parent

object type's section (either `ApplyType`, `PolicyReferenceType`, `SharedVariableReferenceType`, or one of the `QuantifiedExpressionTypes`); if there is not any, or if the `DataType` is still undefined after applying the rule(s), then it is treated as being set to `urn:oasis:names:tc:acal:1.0:data-type:string` by default.

#### **Issuer [Optional]**

A restricted `String` value which, if supplied, specifies the `Issuer` of the named attribute.

#### **MustBePresent [Optional, Default false]**

A Boolean value that governs whether the designator returns `Indeterminate` or an empty bag in the event the named attribute is absent from the source of attributes. See Section 8.4.5. Also see Section 8.17.2 and Section 8.17.3. If this property is omitted, then it is treated as being set to `false`.

A named attribute SHALL be considered present if there is at least one attribute in the source of attributes that matches the criteria set out below.

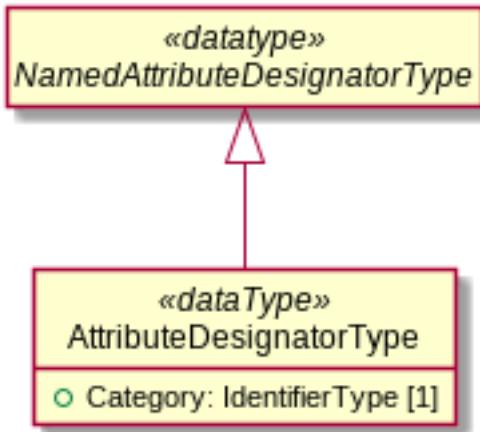
A named attribute matches an attribute in the source of attributes if the values of their respective `AttributeId`, `DataType` and `Issuer` properties match. The attribute designator's `AttributeId` property MUST match, by identifier equality, the `AttributeId` property of the attribute. The attribute designator's `DataType` property MUST match, by identifier equality, the `DataType` property of the same attribute.

If the `Issuer` property is present in the attribute designator, then it MUST match, using the `urn:oasis:names:tc:acal:1.0:function:string-equal` function, the `Issuer` property of the same attribute. If the `Issuer` property is not present in the attribute designator, then the matching of the attribute to the named attribute SHALL be governed by `AttributeId` and `DataType` properties alone.

## 7.18 AttributeDesignatorType

An `AttributeDesignatorType` object is a kind of expression that defines an attribute designator. An attribute designator retrieves a bag of values for a named attribute from the request context.

UML definition (class diagram):



The `AttributeDesignatorType` object type extends the `NamedAttributeDesignatorType` object type with the following property:

#### **Category [Required]**

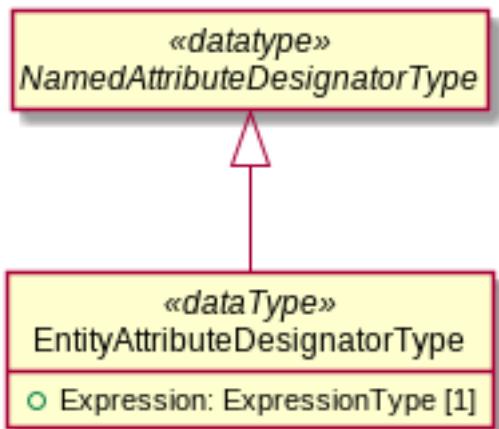
An `IdentifierType` value specifying an attribute category in the request context from which values of the named attribute are retrieved.

The properties inherited from `NamedAttributeDesignatorType` specify the named attribute. The attribute designator retrieves values from attributes matching the named attribute that are present in the `RequestEntityType` object having a `Category` property that matches, by identifier equality, the `Category` property of the attribute designator.

## 7.19 EntityAttributeDesignatorType

An `EntityAttributeDesignatorType` object is a kind of expression that defines an entity attribute designator. An entity attribute designator retrieves a bag of values for a named attribute from either the request context or a value of the `urn:oasis:names:tc:acal:1.0:data-type:entity` data type.

UML definition (class diagram):



The **EntityAttributeDesignatorType** object type extends the **NamedAttributeDesignatorType** object type with the following property:

#### **Expression [Required]**

An **ExpressionType** object defining an expression that MUST evaluate to either a single value of the `urn:oasis:names:tc:acal:1.0:data-type:entity` data type or a single value of the `urn:oasis:names:tc:acal:1.0:data-type:anyURI` data type.

The properties inherited from **NamedAttributeDesignatorType** specify the named attribute.

If the expression evaluates to a value of the `urn:oasis:names:tc:acal:1.0:data-type:entity` data type, then the entity attribute designator retrieves values from attributes matching the named attribute that are present in the value of the expression.

If the expression evaluates to a value of the `urn:oasis:names:tc:acal:1.0:data-type:anyURI` data type, then the entity attribute designator retrieves values from attributes matching the named attribute that are present in the **RequestEntityType** object having a **Category** property that matches, by identifier equality, the value of the expression. In this case the entity attribute designator emulates an attribute designator.

## **7.20 BaseAttributeSelectorType (optional)**

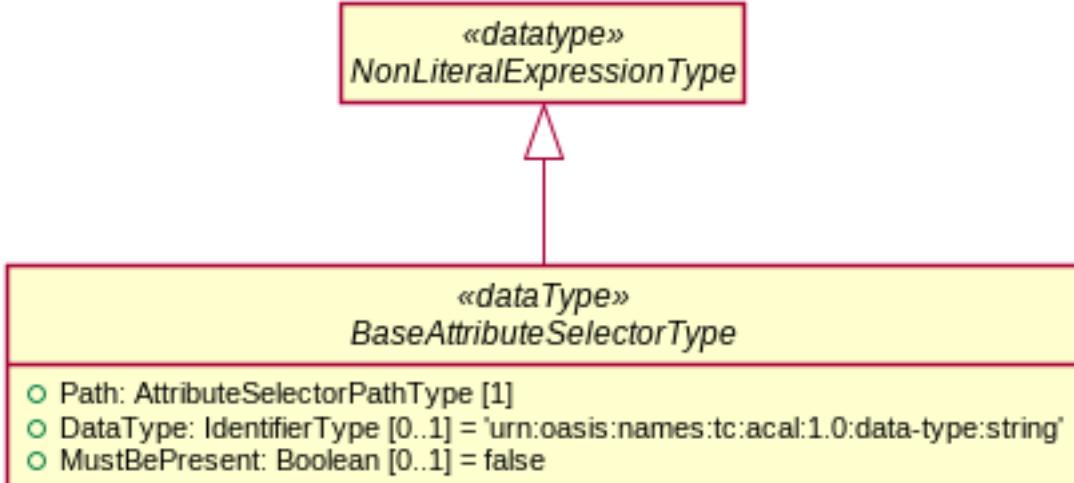
*Support for this part (attribute selectors and entity attribute selectors) is OPTIONAL.*

The **BaseAttributeSelectorType** object type is an abstract object type that specifies a Content-specific Path expression (e.g. XPath) for the production of a bag of ACAL attribute values. The object types derived from **BaseAttributeSelectorType** determine the source of the content: an **AttributeSelectorType** object (an attribute selector) specifies the **Content** property of an attribute category in the request context and an **EntityAttributeSelectorType** object (an entity attribute selector) specifies the **Content** property in either an attribute category in the request context or a value of the `urn:oasis:names:tc:acal:1.0:data-type:entity` data type.

Evaluation of an **AttributeSelectorType** or **EntityAttributeSelectorType** object returns a bag of unnamed and uncategorized ACAL attribute values, or an error.

See Section 8.4.7 for details of attribute selector and entity attribute selector evaluation.

UML definition (class diagram):



A **BaseAttributeSelectorType** object has the following properties:

**Path [Required]**

A restricted **String** value that contains an expression to be evaluated against the specified content. The concrete syntax and semantics of the expression is separate ACAL Profiles (e.g. XPath / JSONPath Profiles). See Section 8.4.7 for details of the expression evaluation during attribute selector and entity attribute selector processing.

**DataType [Optional]**

An **IdentifierType** value specifying the data type of the values returned from the evaluation of the attribute selector or entity attribute selector. If this property is omitted, the **DataType** inference rule(s) of the parent object SHALL apply first, if there is any rule defined in this specification in the parent object type's section (either **ApplyType**, **PolicyReferenceType**, **SharedVariableReferenceType**, or one of the **QuantifiedExpressionTypes**); if there is not any, or if the **DataType** is still undefined after applying the rule(s), then it is treated as being set to **urn:oasis:names:tc:acal:1.0:data-type:string** by default.

**MustBePresent [Optional, Default false]**

A **Boolean** value that governs whether the attribute selector or entity attribute selector returns **Indeterminate** or an empty bag in the event that the specified content does not exist, or the content does exist but the Path expression selects no node. See Section 8.4.5. Also see Section 8.17.2 and Section 8.19.3. If this property is omitted, then it is treated as being set to **false**.

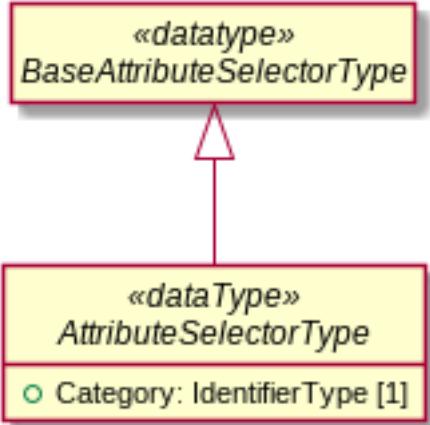
## 7.21 AttributeSelectorType (optional)

*Support for this part (attribute selectors) is OPTIONAL.*

An **AttributeSelectorType** object is an abstract object type of expression that defines an attribute selector. An attribute selector produces a bag of unnamed and uncategorized ACAL attribute values. The values shall be constructed from the node(s) selected by applying the Path expression given by the attribute selector's **Path** property to the structured **Content** indicated by the attribute selector's **Category** property. Concrete types of this are defined in ACAL Profiles, e.g. XPath Profile.

See Section 8.4.7 for details of attribute selector evaluation.

UML definition (class diagram):



The `AttributeSelectorType` object type extends the `BaseAttributeSelectorType` object type with the following property:

**Category [Required]**

An `IdentifierType` value specifying an attribute category in the request context. The `Content` property of that attribute category contains the structured content from which nodes will be selected.

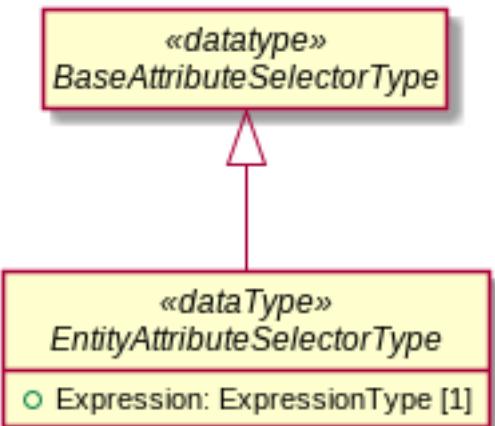
## 7.22 EntityAttributeSelectorType (optional)

*Support for this part (entity attribute selectors) is OPTIONAL.*

An `EntityAttributeSelectorType` object is an abstract object type of expression that defines an entity attribute selector. An entity attribute selector produces a bag of unnamed and uncategorized ACAL attribute values. The values shall be constructed from the node(s) selected by applying the Path expression given by the entity attribute selector's `Path` property to the structured content in the `Content` property in either an attribute category in the request context or a value of the `urn:oasis:names:tc:acal:1.0:data-type:entity` data type. Concrete types of this are defined in ACAL Profiles, e.g. XPath Profile.

See Section 8.4.7 for details of entity attribute selector evaluation.

UML definition (class diagram):



The `EntityAttributeSelectorType` object type extends the `BaseAttributeSelectorType` object type with the following property:

**Expression [Required]**

An `ExpressionType` object defining an expression that MUST evaluate to either a single value of the `urn:oasis:names:tc:acal:1.0:data-type:entity` data type or a single value of the `urn:oasis:names:tc:acal:1.0:data-type:anyURI` data type.

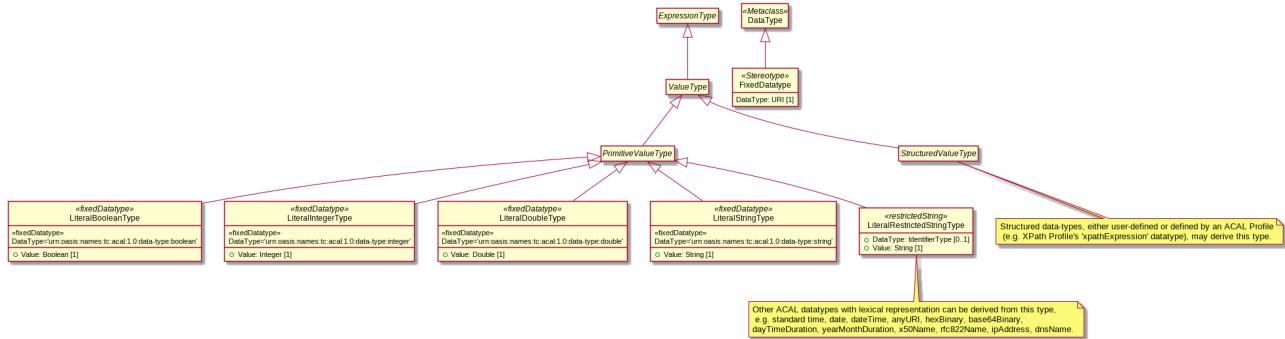
If the expression evaluates to a value of the `urn:oasis:names:tc:acal:1.0:data-type:entity` data type, then the entity attribute selector applies the Path expression given by the entity attribute selector's `Path` property to the structured content in the `Content` property of that value.

If the expression evaluates to a value of the `urn:oasis:names:tc:acal:1.0:data-type:anyURI` data type, then the entity attribute selector applies the Path expression given by the entity attribute selector's Path property to the structured content in the Content property of the RequestEntityType object having a Category property that matches, by identifier equality, the value of the expression. In this case the entity attribute selector emulates an attribute selector.

## 7.23 ValueType

A `ValueType` object is a kind of expression that contains a literal ACAL value.

UML model (class diagram):



A `ValueType` object has the following properties:

### `DataType` [Optional]

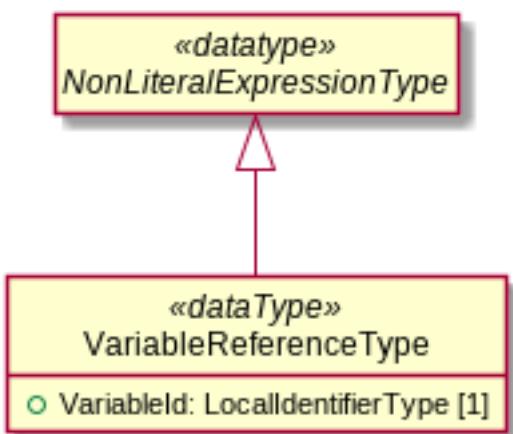
An `IdentifierType` value specifying the data type of the attribute value. If this property is omitted, the `DataType` inference rule(s) of the parent object SHALL apply first, if there is any rule defined in this specification in the parent object type's section (either `ApplyType`, `PolicyReferenceType`, `SharedVariableReferenceType`, or one of the `QuantifiedExpressionTypes`); if there is not any, or if the `DataType` is still undefined after applying the rule(s), then it is treated as being set to `urn:oasis:names:tc:acal:1.0:data-type:string` by default.

A `ValueType` is abstract and subtypes may be either primitive types - i.e. subtypes of `PrimitiveValueType` in the above diagram (`LiteralStringType`, `LiteralIntegerType`, etc.) - or structured types (see section 8.4.1) - i.e. subtypes of `StructuredValueType` in the above diagram.

## 7.24 VariableReferenceType

A `VariableReferenceType` object is a kind of expression used to reference a variable definition.

UML definition (class diagram):



A `VariableReferenceType` object contains the following property:

### `VariableId` [Required]

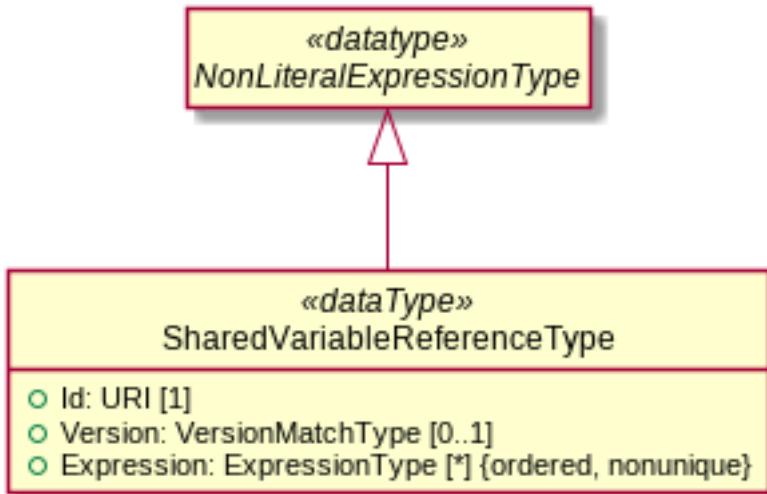
A `LocalIdentifierType` value used to refer to the ACAL value or bag of values defined in a `VariableDefinitionType` object. The property value MUST match, by identifier equality, the

`VariableId` property of exactly one `VariableDefinitionType` object that is in the `VariableDefinition` property of an enclosing `RuleType` or `PolicyType` object.

## 7.24b SharedVariableReferenceType

A `SharedVariableReferenceType` object is used to reference a shared variable by identifier and optional version.

UML definition (class diagram):



The `SharedVariableReferenceType` object type contains the following properties:

### **Id** [Required]

A URI being the `Id` of the referenced shared variable. If the URI is a URL, then it MAY be resolvable to the shared variable. However, the mechanism for resolving a shared variable reference to the corresponding shared variable is outside the scope of this specification.

### **Version** [Optional]

Specifies a matching expression for selecting an acceptable version of the referenced shared variable. The matching operation is defined in Section 7.12. If this property is present, then the selected version of the shared variable MUST match the expression. If the property is absent, then any version of the shared variable is acceptable. In the case that more than one version matches, then the most recent one SHOULD be used.

### **Expression** [Any Number]

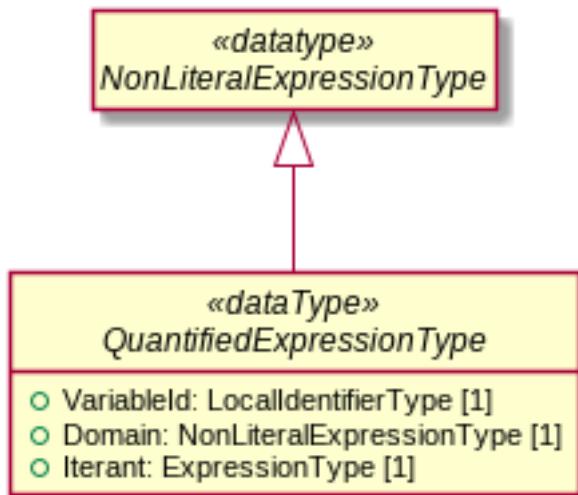
Arguments for a parameterized shared variable (`SharedVariableDefinitionType` object), in the same order as the `Parameter` declarations in the referenced shared variable, and each argument number  $n$  MUST match the definition of the `Parameter` number  $n$  (`DataType`, `isBag`). The number of arguments  $N$  may be less than the number of declared `Parameters` if and only if the `Parameters` number  $N$  (starting at zero) and above have defined default values, in which case they are used as remaining arguments.

**DataType inference rule:** if an `Expression` has an (optional) `DataType` property (statically-typed `Expression`), i.e. its type is `ValueType`, `NamedAttributeDesignatorType`, `BaseAttributeSelectorType` or any subtype thereof, then the `Expression`'s `DataType` property SHOULD be omitted in this case as it is already defined by the corresponding `Parameter`'s definition in the referenced `SharedVariableDefinitionType` object. Else it SHALL match that `Parameter`'s `DataType`.

## 7.25 QuantifiedExpressionType (optional)

A `QuantifiedExpressionType` object is a kind of expression that represents one of four kinds of quantified expression. The kind of quantified expression is determined by the name of the property that holds the object, either `ForAny`, `ForAll`, `Select` or `Map`. There are some common requirements for all four kinds of quantified expressions.

UML definition (class diagram):



A QuantifiedExpressionType object contains the following properties:

**VariableId [Required]**

A LocalIdentifierType value to identify the quantified variable that will be used by the quantified expression. The quantified variable does not have a corresponding variable definition. The value of the property MUST NOT be the same as the VariableId property of a VariableDefinitionType object in the VariableDefinition property of an enclosing PolicyType or RuleType object.

**Domain [Required]**

An ExpressionType object defining an expression. This expression is called the domain. The domain SHALL be an expression that evaluates to a bag of values of the same data type, or Indeterminate in the case of an error. The bag MAY be empty.

**Iterant [Required]**

An ExpressionType object defining an expression. This expression is called the iterant expression. The iterant expression SHALL be an expression that evaluates to a single value (for all but one kind of quantified expression the data type of that value is urn:oasis:names:tc:acal:1.0:data-type:boolean).

Quantified expressions MAY be nested in either or both of the domain and the iterant expression. A nested quantified expression SHALL NOT use the same VariableId as an enclosing quantified expression.

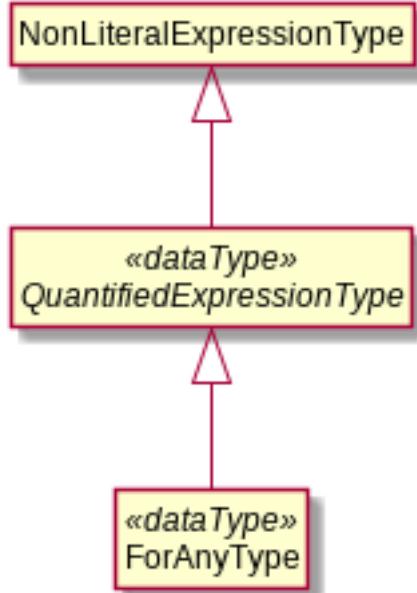
The evaluation of a quantified expression begins with the evaluation of the domain expression. The iterant expression is then evaluated once for each value from the domain with the quantified variable set to that value, except that evaluation of the quantified expression may terminate before considering all the values from the domain if the final result of the quantified expression has already been determined. The conditions for early termination are specified for each kind of quantified expression. The effect of the result of the iterant expression on the overall result of the quantified expression depends on the kind of quantified expression. Bags of values are unordered so there is no constraint on the order in which an implementation chooses to consider values from the domain.

The value of the quantified variable, i.e., the particular value from the domain, MAY be referenced one or more times from within the iterant expression using a VariableReferenceType object. It is not an error if the quantified variable is not referenced at all. The quantified variable SHALL NOT be referenced from within the domain of the quantified expression to which it belongs. A quantified variable MAY be referenced from within the domain of another quantified expression nested within the iterant expression.

### 7.25.1 ForAny Expression

The ForAny quantified expression tests whether any value of a bag satisfies the iterant expression. It is represented by a ForAny property, which is of the QuantifiedExpressionType object type.

UML definition (class diagram):



The iterant expression of a ForAny expression SHALL be an expression that evaluates to a value of the `urn:oasis:names:tc:acal:1.0:data-type:boolean` data type.

**Data Type inference rule:** if the Iterant Expression has an (optional) `DataType` property (statically-typed Expression), i.e. its type is `ValueType`, `NamedAttributeDesignatorType`, `BaseAttributeSelectorType` or any subtype thereof, then its `DataType` property SHALL be omitted as the `DataType` is already fixed to be `urn:oasis:names:tc:acal:1.0:data-type:boolean` by this definition.

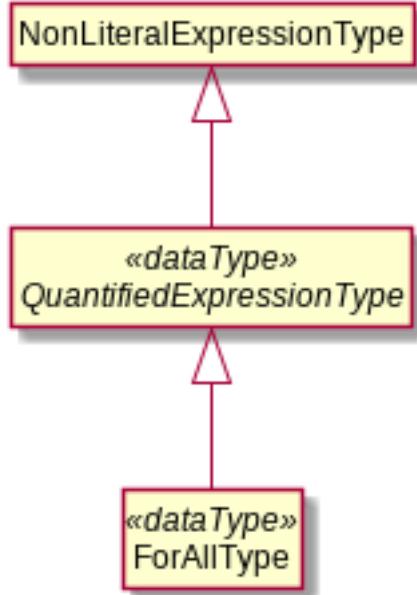
The result of a ForAny expression SHALL be a value of the `urn:oasis:names:tc:acal:1.0:data-type:boolean` data type or `Indeterminate`.

The ForAny expression evaluates to `true` if the iterant expression evaluates to `true` for any value from the domain; otherwise, the expression evaluates to `Indeterminate` if the iterant expression evaluates to `Indeterminate` for any value from the domain; otherwise, the expression evaluates to `false`. Note that the ForAny expression evaluates to `false` if the domain is an empty bag. Evaluation of the expression MAY terminate whenever the iterant expression evaluates to `true`.

### 7.25.2 ForAll Expression

The ForAll quantified expression tests whether all values of a bag satisfy the iterant expression. It is represented by the `ForAll` property, which is of the `QuantifiedExpressionType` object type.

UML definition (class diagram):



The iterant expression of a ForAll expression SHALL be an expression that evaluates to a value of the `urn:oasis:names:tc:acal:1.0:data-type:boolean` data type.

**Data Type inference rule:** if the Iterant Expression has an (optional) `DataType` property (statically-typed Expression), i.e. its type is `ValueType`, `NamedAttributeDesignatorType`, `BaseAttributeSelectorType` or any subtype thereof, then its `DataType` property SHALL be omitted as the Data Type is already fixed to be `urn:oasis:names:tc:acal:1.0:data-type:boolean` by this definition.

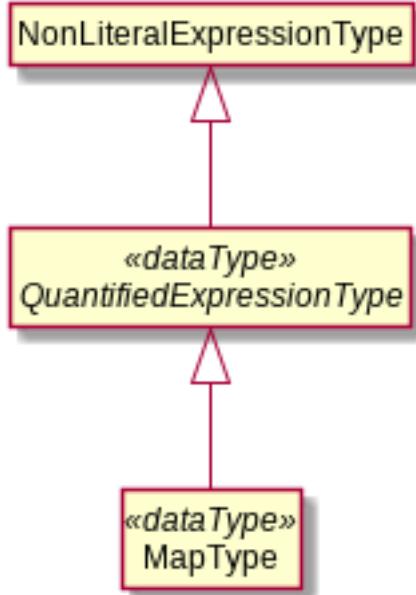
The result of a ForAll expression SHALL be a value of the `urn:oasis:names:tc:acal:1.0:data-type:boolean` data type or `Indeterminate`.

The ForAll expression evaluates to `false` if the iterant expression evaluates to `false` for any value from the domain; otherwise, the expression evaluates to `Indeterminate` if the iterant expression evaluates to `Indeterminate` for any value from the domain; otherwise, the expression evaluates to `true`. Note that the ForAll expression evaluates to `true` if the domain is an empty bag. Evaluation of the expression MAY terminate whenever the iterant expression evaluates to `false`.

### 7.25.3 Map Expression

The Map quantified expression converts a bag of values to another bag of values. It is represented by the `Map` property, which is of the `QuantifiedExpressionType` object type. If the expression is a literal value - `ValueType` object - without a defined Data Type, it SHALL be handled as string (Data Type identifier `urn:oasis:names:tc:acal:1.0:data-type:string`) by default.

UML definition (class diagram):



The iterant expression of a Map expression SHALL be an expression that evaluates to a single value (not necessarily of the same data type as the values in the domain).

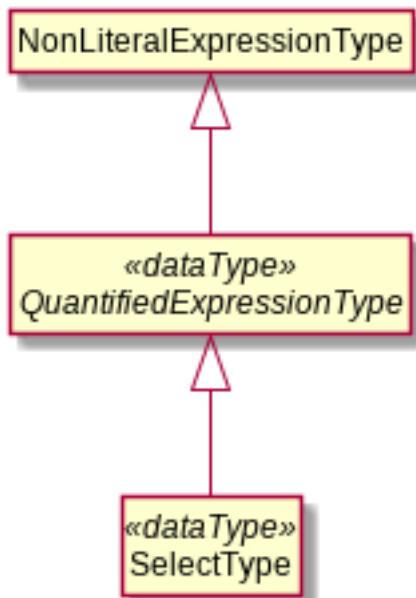
The result of the Map expression SHALL be a bag of values of the same data type as the iterant expression or **Indeterminate**.

If the iterant expression evaluates to **Indeterminate** for any value from the domain, then the result of the Map expression is **Indeterminate**; otherwise, the result bag contains the values resulting from the evaluation of the iterant expression for each value from the domain. The Map expression evaluates to an empty bag if the domain is an empty bag. Evaluation of the Map expression MAY terminate whenever the iterant expression evaluates to **Indeterminate**.

#### 7.25.4 Select Expression

The Select quantified expression returns a bag containing the values from the domain that satisfy the iterant expression. That is, the result is a subset of, or equal to, the domain. The Select expression is represented by the **Select** property, which is of the **QuantifiedExpressionType** object type.

UML definition (class diagram):



The iterant expression of a Select expression SHALL be an expression that evaluates to a value of the `urn:oasis:names:tc:acal:1.0:data-type:boolean` data type.

**DataType inference rule:** if the Iterant Expression has an (optional) DataType property (statically-typed Expression), i.e. its type is ValueType, NamedAttributeDesignatorType, BaseAttributeSelectorType or any subtype thereof, then its DataType property SHALL be omitted as the DataType is already fixed to be urn:oasis:names:tc:acal:1.0:data-type:boolean by this definition.

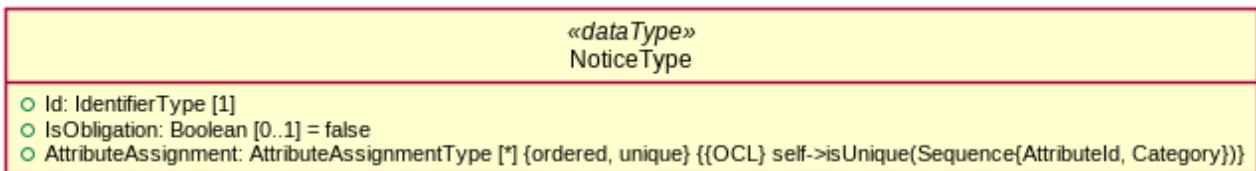
The result of a Select expression SHALL be a bag of values of the same data type as the values from the domain or Indeterminate.

If the iterant expression evaluates to Indeterminate for any value from the domain, then the result of the Select expression is Indeterminate; otherwise, the result bag contains each value from the domain for which the iterant expression evaluates to true. The Select expression evaluates to an empty bag if the domain is an empty bag. Evaluation of the Select expression MAY terminate whenever the iterant expression evaluates to Indeterminate.

## 7.26 NoticeType

A NoticeType object contains an identifier for a notice and a list of attribute assignments that form arguments of the notice.

UML definition (class diagram):



See Section 8.16 for a description of how the list of notices to be returned by the PDP is determined.

A NoticeType object contains the following properties:

### Id [Required]

An IdentifierType value specifying an identifier for the notice that the PEP associates with particular processing requirements or informational content.

### IsObligation [Optional, Default false]

A Boolean value which, if present and set to true, indicates the notice is an obligation; otherwise, the notice is advice.

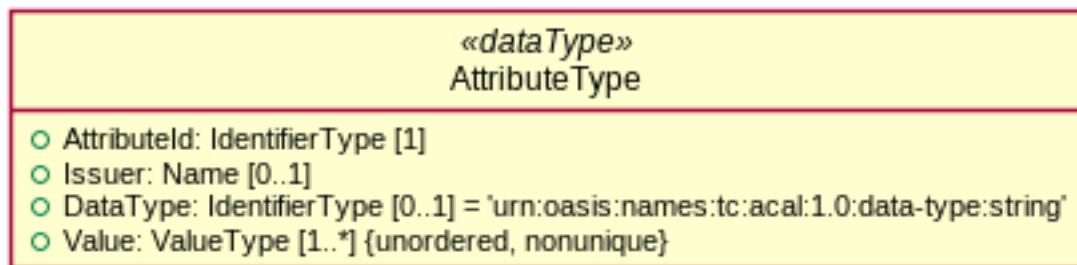
### AttributeAssignment [Any Number]

A sequence of AttributeAssignmentType objects (each of them is an attribute assignment) forming the arguments of the notice.

## 7.27 AttributeType

An AttributeType object contains attribute meta-data and one or more attribute values. The attribute meta-data comprises the attribute identifier and the attribute issuer. Attribute designators in a policy MAY refer to attributes by means of this meta-data.

UML definition (class diagram):



An AttributeType object contains the following properties:

**AttributeId [Required]**

An IdentifierType value identifying the attribute. A number of identifiers are reserved by ACAL to denote commonly used attributes. See Annex D.

**Issuer [Optional]**

A Name value specifying the issuer of the attribute. 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.

**DataType [Optional, Default urn:oasis:names:tc:acal:1.0:data-type:string]**

An IdentifierType value specifying the data type of the attribute's values. If this property is omitted, then it is treated as being set to `urn:oasis:names:tc:acal:1.0:data-type:string`.

**Value [One to Many]**

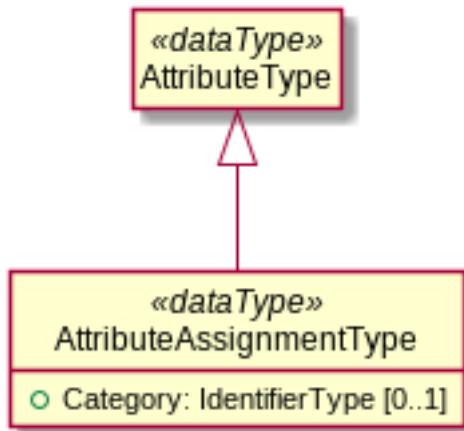
A sequence of ValueType objects, each denoting an ACAL attribute value. Duplicated attribute values are permitted. None of these objects SHALL redefine/override the DataType identifier already defined by the above DataType property, and SHOULD not have any DataType identifier property set at all, if possible.

## 7.28 AttributeAssignmentType

An AttributeAssignmentType object is used in a notice expression to include an attribute in a notice. The AttributeAssignmentType object type extends the AttributeType definition by adding a Category property.

The attribute specified SHALL be understood by the PEP, but it is not further specified by ACAL. See Section 8.16. Section 6.2.4.3 provides a number of examples of attribute assignments included in notices.

UML definition (class diagram):



The AttributeAssignmentType object type extends the AttributeType object type with the following property:

**Category [Optional]**

An IdentifierType value specifying the category of the attribute. If this property is absent, the attribute has no category. The PEP SHALL interpret the significance and meaning of any Category property. Non-normative note: an expected use of the category is to disambiguate attributes that are relayed from the request.

None of the Values may have a DataType property on its own since it is already defined by the DataType property at the AttributeAssignmentType object level (inherited from AttributeType).

## 7.29 NoticeExpressionType

A NoticeExpressionType object defines a notice expression that is potentially evaluated into a notice. A notice expression contains an identifier for the notice and a set of expressions that form arguments of the notice. The AppliesTo property indicates the effect for which this notice is eligible to be provided to the PEP. If the AppliesTo property is absent, then this notice is eligible to be provided to the PEP if the effect is either Permit or Deny. See Section 8.16 for a description of how the list of notices to be returned by the PDP is determined.

UML definition (class diagram):

«*dataType*»  
NoticeExpressionType

- Id: IdentifierType [1]
- IsObligation: Boolean [0..1]
- AppliesTo: EffectType [0..1]
- Condition: BooleanExpressionType [0..1]
- AttributeAssignmentExpression: AttributeAssignmentExpressionType [\*] {ordered, unique} {{OCL} self->isUnique(Sequence{AttributId, Category})}}

A NoticeExpressionType object contains the following properties:

**Id [Required]**

An IdentifierType value nominating the identifier for the notice that the PEP associates with particular processing requirements or informational content.

**IsObligation [Optional]**

A Boolean value that determines the setting for the IsObligation property of the notice. If IsObligation is present, then the IsObligation property of the notice is set to the same value; otherwise, the IsObligation property is omitted, then from the notice.

**AppliesTo [Optional]**

An EffectType value that specifies the effect for which this notice is eligible to be provided to the PEP, or if absent, either Permit or Deny.

**Condition [Optional]**

A BooleanExpressionType object holding an expression that must be satisfied for this notice to be provided to the PEP. An absent condition is treated as unconditionally satisfied. Evaluation of the Condition property is described in Section 8.9.

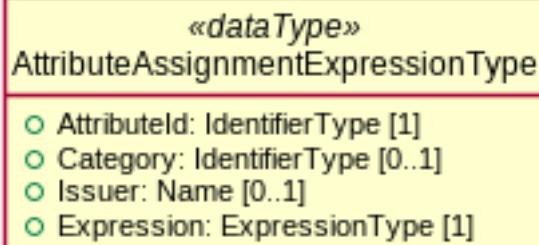
**AttributeAssignmentExpression [Any Number]**

A sequence of AttributeAssignmentExpressionType objects, each specifying a notice argument in the form of an expression. The expressions SHALL be evaluated by the PDP to constant ValueType objects or bags, which shall be the attribute assignments in the notice returned to the PEP. If an AttributeAssignmentExpressionType object evaluates to an atomic attribute value, then there MUST be one resulting AttributeAssignmentType object which MUST contain this single attribute value. If the AttributeAssignmentExpressionType object evaluates to a bag, then there MUST be a resulting AttributeAssignmentType object for each of the values in the bag. If the bag is empty, there shall be no AttributeAssignmentType objects from this AttributeAssignmentExpressionType object. The values of the notice arguments SHALL be interpreted by the PEP.

## 7.30 AttributeAssignmentExpressionType

An AttributeAssignmentExpressionType object is used to include an argument in a notice. It SHALL contain an AttributId property 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 ACAL. See Section 8.16. Section 6.2.4.3 provides a number of examples of attribute assignment expressions included in notice expressions.

UML definition (class diagram):



An AttributeAssignmentExpressionType object contains the following properties:

**Expression [Required]**

An ExpressionType object defining an expression which evaluates to a constant attribute value or a bag of zero or more attribute values. See Section 7.14. If the Expression value is a ValueType object (literal value) without a defined DataType identifier, it SHALL be handled as String (with DataType identifier urn:oasis:names:tc:acal:1.0:data-type:string) by default. Note that the way

the *DataType* identifier is defined depends on the concrete type (subtype of *ValueType*) of that value. For example, *ValueType* subtypes stereotyped <>*fixedDatatype*<> have the *DataType* identifier globally defined and fixed as a Stereotype property (therefore implicitly predefined for every instance), whereas the *LiteralRestrictedStringType* have a UML attribute *DataType* that can be set to a different value (identifier) for each instance.

#### **AttributeId [Required]**

An IdentifierType value specifying the identifier of the attribute. The value of the **AttributeId** property in the resulting **AttributeAssignmentType** object MUST be equal to this value.

#### **Category [Optional]**

An IdentifierType value specifying the category of the attribute. If this property is missing, the attribute has no category. The value of the **Category** property in the resulting **AttributeAssignmentType** object MUST be equal to this value.

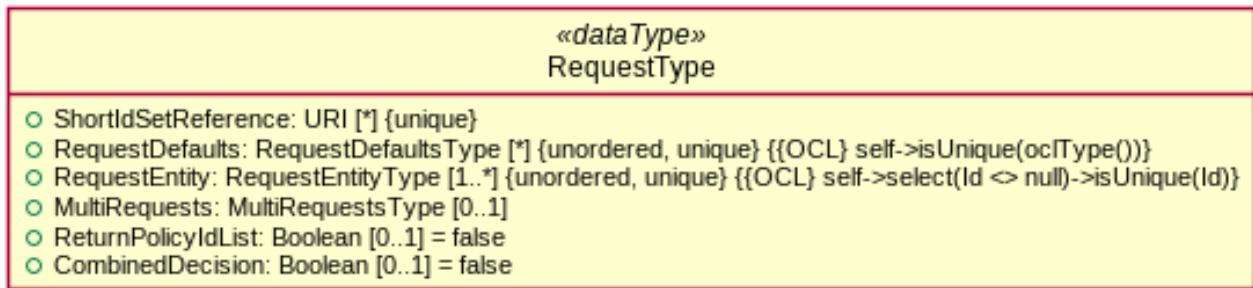
#### **Issuer [Optional]**

A Name value specifying the issuer of the attribute. If this property is absent, the attribute has no issuer. The value of the **Issuer** property in the resulting **AttributeAssignmentType** object MUST be equal to this value.

### **7.31 RequestType**

The **RequestType** object type 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 any particular syntactic form. But, any system conforming to the ACAL specification MUST produce exactly the same authorization decisions as if all the inputs had been transformed into the form of a **RequestType** object.

UML definition (class diagram):



A **RequestType** object contains the following properties:

#### **ReturnPolicyIdList [Optional, Default false]**

A Boolean value that is used to request that the PDP return a list of all fully applicable policies which were used in the decision as a part of the decision response. If this property is omitted, then it is treated as being set to **false**.

#### **CombinedDecision [Optional, Default false]**

A Boolean value that is used to request that the PDP combines multiple decisions into a single decision. The use of this property is specified in [Multi]. If the PDP does not implement the relevant functionality in [Multi], then the PDP must return an Indeterminate decision with a status code of `urn:oasis:names:tc:acal:1.0:status:processing-error` if it receives a request with this property set to **true**. If this property is omitted, then it is treated as being set to **false**.

#### **ShortIdSetReference [Any Number]**

A sequence of URI values referencing short identifier sets. The short identifiers used by the request MUST be ones defined in the referenced sets.

#### **RequestDefaults [Any Number]**

sequence of **RequestDefaultsType** objects; each contains default values specific to an ACAL Profile, for processing the request (e.g. ACAL XPath Profile defines the default XPath version). In particular, each object SHALL have a different concrete type (per ACAL profile). See Section 7.32.

#### **RequestEntity [One to Many]**

A sequence of **RequestEntityType** objects, each containing a sequence of **RequestAttributeType** objects

associated with an attribute category of the request context. Different `RequestEntityType` objects with different category identifiers are used to represent information about the subject, resource, action, environment or other categories of the access request. There may be multiple `RequestEntityType` objects with the same `Category` property value if the PDP implements the multiple decision profile, see [Multi]. Under other conditions, it is a syntax error if there are multiple `RequestEntityType` objects with the same `Category` (see Section 8.17.2 for error codes).

#### **MultiRequests [Optional]**

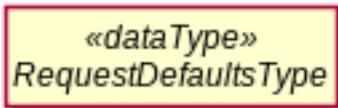
A `MultiRequestsType` object listing multiple request contexts using references to the `RequestEntityType` objects. Implementation of this property is optional. The semantics of this property are defined in [Multi]. If the implementation does not implement this property, it MUST return an `Indeterminate` result if it encounters this property. See Section 7.38.

### **7.32 RequestDefaultsType (optional)**

*Supporting this part is optional. It is required only for supporting the ACAL Profiles that define extensions (subtypes) of `RequestDefaultsType` (e.g. XPath Profile).*

`RequestDefaultsType` is an abstract object type for default values that apply to the parent `RequestType` object. Concrete subtypes of `RequestDefaultsType` are defined in separate ACAL Profiles, e.g. XPath Profile.

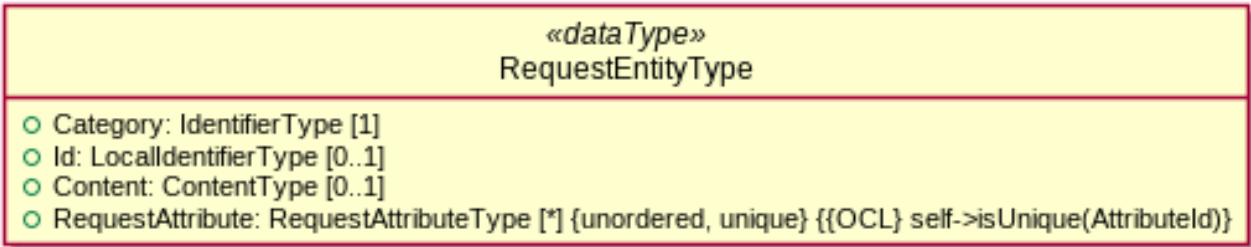
UML definition (class diagram):



### **7.33 RequestEntityType**

A `RequestEntityType` object specifies ACAL attributes of a subject, resource, action, environment or another category using a sequence of `RequestAttributeType` objects.

UML definition (class diagram):



A `RequestEntityType` object contains the following properties:

#### **Category [Required]**

An `IdentifierType` value indicating which attribute category the contained attributes belong to, and it is used to differentiate between attributes of subject, resource, action, environment or other categories.

#### **Id [Optional]**

A unique `String` identifier for this `RequestEntityType` object. It is primarily intended to be referenced in multiple requests. See [Multi].

#### **Content [Optional]**

A `ContentType` object specifying additional sources of attributes in JSON or XML document format which can be referenced using attribute selectors and entity attribute selectors.

#### **RequestAttribute [Any Number]**

A sequence of `RequestAttributeType` objects associated with the attribute category of the request.

### **7.34 ContentType (optional)**

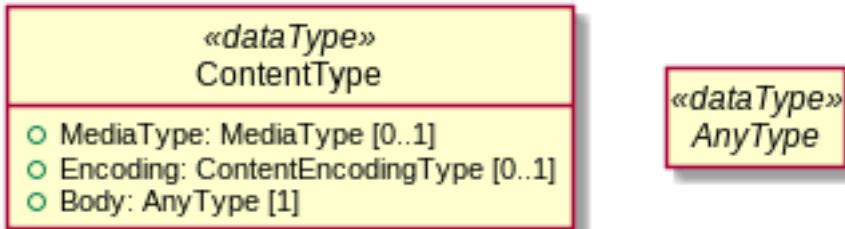
*Support for this object type is optional. It is required only if the implementation must support extensions of AttributeSelectors and/or DataTypes depending on this object type.* Such extensions

are typically defined in ACAL Profiles (e.g. XPath and JSONPath Profiles).

A **ContentType** object is a notional placeholder for additional attributes, typically the content of the resource. A **ContentType** object has a *body* property of arbitrary type.

**ContentType** objects should be used only for structured data that cannot be handled by named attributes, i.e. the *body* type should be a structured data-type.

The following UML definition (class diagram) is provided for information purposes only:



A **ContentType** object contains the following properties:

#### **MediaType** [Optional]

Content *media type* as defined by [RFC 2046]. The exact possible values for this property SHALL be defined by the **AttributeSelector** Profile(s) of ACAL, e.g. the XPath and/or JSONPath Profiles, that your implementation supports. Indeed, each of these profiles supports one or more specific Content Type(s), and for each one, defines the acceptable **MediaType** property value. In any case, the value(s) SHOULD be among the standard identifiers specified in Annex D.10 whenever applicable. More information in the **MediaType** section.

A default value for this property SHALL be defined by each ACAL representation format supporting this object type, and this default value SHALL be one of the standard identifiers specified in Annex D.10.

#### **Body** [Required]

The content body which may be any data-type representable in the ACAL representation format in use (XML, JSON, etc.). In other words, each ACAL representation format SHALL define the correct representation for **AnyType** in their own format (e.g. XSD `xs:anyType` in XML), or simply state that *the ContentType is not supported at all* in that representation.

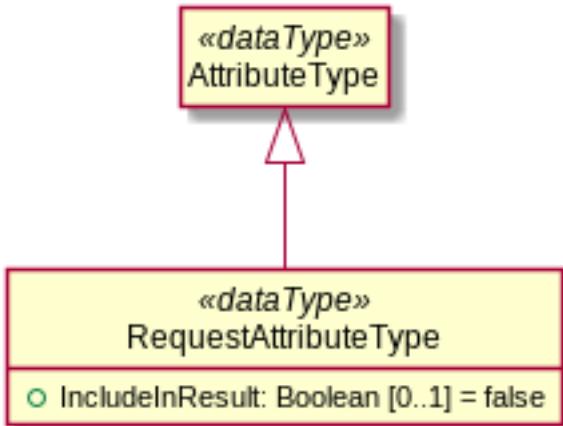
#### **Encoding** [Optional]

This property (similar to JSON schema's `contentEncoding` property) applies when the **Body** value is a string that SHALL be interpreted as encoded binary data and decoded using the encoding named by this property. The exact possible value(s) SHALL be defined by each ACAL representation format supporting this object type (e.g. `base64` encoding for the JSON representation of complex XML content). In any case, the value(s) SHOULD be among the standard identifiers specified in Annex D.11 whenever applicable (more information in the **ContentEncodingType** section). If this property is undefined, this indicates that no encoding was needed in order to represent the content in a UTF-8 string.

## 7.35 RequestAttributeType

A **RequestAttributeType** object describes an ACAL attribute in the request context.

UML definition (class diagram):



The RequestAttributeType object type extends the AttributeType object type with the following property:

**IncludeInResult [Optional, Default false]**

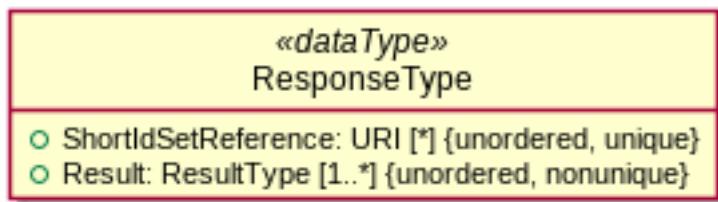
A Boolean value that governs whether this ACAL attribute is included in the result. This is useful to correlate requests with their responses in the case of multiple requests. If this property is omitted, then it is treated as being set to **false**. Note that ACAL attributes appear in the result as **AttributeType** objects.

## 7.36 ResponseType

The **ResponseType** object type is an abstraction layer used by the policy language. Any proprietary system using the ACAL specification MUST transform an ACAL context **ResponseType** object into the form of its authorization decision.

A **ResponseType** object encapsulates the authorization decision produced by the PDP. It includes a sequence of one or more results, with one **ResultType** object per requested resource. Multiple results MAY be returned by some implementations, in particular those that support the ACAL Profile for Requests for Multiple Resources [Multi]. Support for multiple results is OPTIONAL.

UML definition (class diagram):



A **ResponseType** object contains the following properties:

**ShortIdSetReference [Any Number]**

A sequence of URI values referencing short identifier sets. The short identifiers used by the response MUST be ones defined in the referenced sets.

**Result [One to Many]**

A sequence of **ResultType** objects, each an authorization decision result. See Section 7.37.

## 7.37 ResultType

A **ResultType** object represents an authorization decision result. It MAY include a list of notices. If the PEP does not understand or cannot fulfill an obligation notice, then the action of the PEP is determined by its bias, see Section 8.2. Any advice notices MAY be safely ignored by the PEP.

UML definition (class diagram):

**«*dataType*»**  
**ResultType**

- Decision: *DecisionType* [1]
- Status: *StatusType* [0..1]
- Notice: *NoticeType* [\*] {ordered, nonunique}
- ResultEntity: *ResultEntityType* [\*] {unordered, unique} {{OCL} self->isUnique(Category)}
- ApplicablePolicyReference: *ExactMatchIdReferenceType* [\*] {unordered, unique} {{OCL} self->isUnique(Id)}

A *ResultType* object contains the following properties:

**Decision [Required]**

A *DecisionType* value indicating the authorization decision: Permit, Deny, Indeterminate or NotApplicable.

**Status [Optional]**

A *StatusType* object indicating whether errors occurred during evaluation of the decision request, and optionally, information about those errors. If the *ResponseType* object contains *ResultType* objects whose *StatusType* objects are all identical, and the *ResponseType* object 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 *StatusType* object MAY be omitted from all the *ResultType* objects.

**Notice [any Number]**

A sequence of *NoticeType* objects, each a notice to be interpreted by the PEP. Each object's Id MUST be unique in the sequence. See Section 7.26. If the PEP does not understand or cannot fulfill an obligation notice, then the action of the PEP is determined by its bias, see Section 8.2. If the PEP does not understand an advice notice, the PEP may safely ignore the notice. See Section 8.16 for a description of how the list of notices to be returned by the PDP is determined.

**ResultEntity [Optional]**

A sequence of *ResultEntityType* objects, each an attribute category containing attributes that were part of the request. The choice of which attributes are included here is made with the *IncludeInResult* property of the *RequestAttributeType* objects of the request. See Section 7.27.

**ApplicablePolicyReference [Optional]**

A sequence of *ExactMatchIdReferenceType* objects. If the *ReturnPolicyIdList* property in the *RequestType* object is true (see Section 7.31), a PDP that implements this optional feature MUST return a sequence that includes the identifiers of all policies which were found to be fully applicable, whether or not the effect (after rule combining) was the same or different from the decision. The sequence is unordered. The sequence MAY include the identifiers of other policies that are currently in force, as long as no policies required for the decision are omitted. A PDP MAY satisfy this requirement by including all policies currently in force, or by including all policies which were evaluated in making the decision, or by including all policies which did not evaluate to NotApplicable, or by any other algorithm which does not omit any policies which contributed to the decision. However, a decision which returns NotApplicable MUST return an empty list.

## 7.38 MultiRequestsType (optional)

*Support for this object type and associated property (MultiRequests) is optional.*

A *MultiRequestsType* object contains a list of requests by reference to *RequestEntityType* objects in the enclosing *RequestType* object. The *MultiRequests* property is of *MultiRequestsType* object type. The semantics of this property are defined in [Multi]. If an implementation does not support this property, but receives it, the implementation MUST generate an Indeterminate response.

UML definition (class diagram):

**«*dataType*»**  
**MultiRequestsType**

- RequestReference: *RequestReferenceType* [1..\*] {unordered, unique} {{OCL} self->isUnique(RequestEntityReference->collect(Id)->asSet())}}

A *MultiRequestsType* object contains the following properties.

### **RequestReference [one to many]**

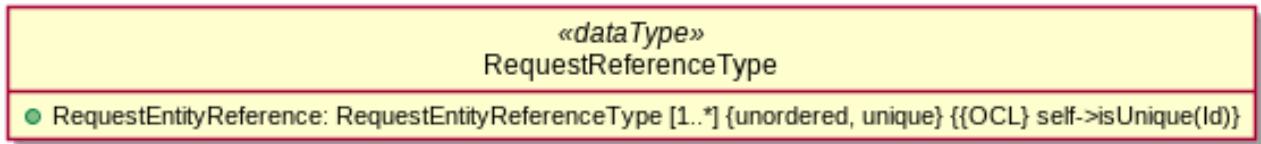
A sequence of RequestReferenceType objects, each defining a request instance by reference to RequestEntityType objects in the enclosing RequestType object. See Section 7.39.

### **7.39 RequestReferenceType (optional)**

*Support for this object type is optional.*

A RequestReferenceType object defines an instance of a request in terms of references to RequestEntityType objects. The semantics of this object type are defined in [Multi].

UML definition (class diagram):



A RequestReferenceType object contains the following properties:

### **RequestEntityReference [one to many]**

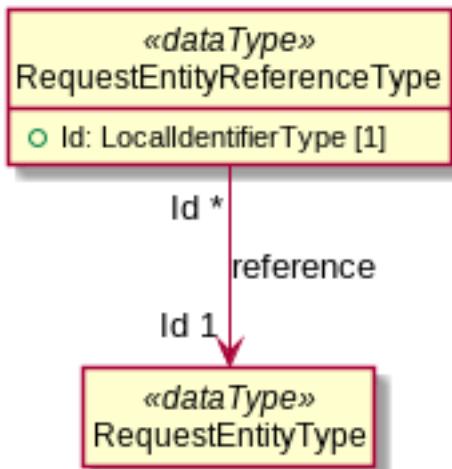
A sequence of RequestEntityReferenceType objects, each a reference to a RequestEntityType object in the enclosing RequestType object. See Section 7.40.

### **7.40 RequestEntityReferenceType (optional)**

*Support for this object is optional.*

A RequestEntityReferenceType object makes a reference to a RequestEntityType object. The meaning of this object is defined in [Multi].

UML definition (class diagram):



A RequestEntityReferenceType object contains the following properties:

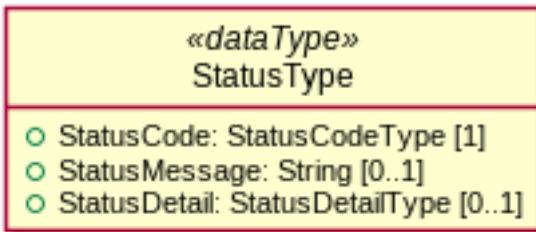
### **Id [Required]**

A LocalIdentifierType value referencing a RequestEntityType object in the enclosing RequestType object by the value of its Id property.

### **7.41 StatusType**

A StatusType object represents the status of the authorization decision result.

UML definition (class diagram):



A `StatusType` object contains the following properties:

**StatusCode [Required]**

A `StatusCodeType` object.

**StatusMessage [Optional]**

A free-form `String` description of the status code.

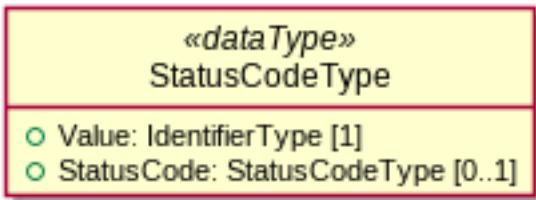
**StatusDetail [Optional]**

A `StatusDetailType` object containing additional status information.

## 7.42 `StatusCodeType`

A `StatusCodeType` object contains a major status code value and an optional recursive series of minor status codes.

UML definition (class diagram):



A `StatusCodeType` object contains the following properties:

**Value [Required]**

See Annex D.8 for a list of values.

**StatusCode [Any Number]**

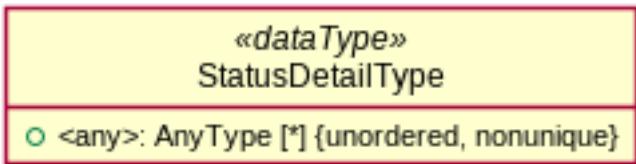
A sequence of `StatusCodeType` objects describing minor status codes. Each minor status code qualifies its parent status code.

## 7.43 `StatusDetailType` (optional)

*Support for this object type is optional.*

A `StatusDetailType` object qualifies its parent `StatusType` object with additional information. A `StatusDetailType` object allows arbitrary content.

The following UML definition (class diagram) is provided for information purposes only:



The `AnyType` data-type above represents *any type* (e.g. XSD `anyType` in XML), and the `<any>` name is only a placeholder for any property name. This UML definition is for information only, and it is the responsibility of each ACAL representation format (ACAL Profile) to either define the correct representation for arbitrary content in their own format, or simply state that *StatusDetailType objects are not supported at all* in that representation format.

For example, this can be represented in XML as follows:

```
<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>
```

Inclusion of a **StatusDetailType** object is optional. However, if a PDP returns one of the following ACAL-defined status code values, then the following rules apply.

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

A PDP MUST NOT return a **StatusDetail** property in conjunction with the **ok** status value.

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

A PDP MAY choose not to return any **StatusDetailType** object or MAY choose to return a **StatusDetailType** object containing one or more **MissingAttributeDetailType** objects.

**urn:oasis:names:tc:acal:1.0:status:syntax-error**

A PDP MUST NOT return a **StatusDetailType** object 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** property describing the problem.

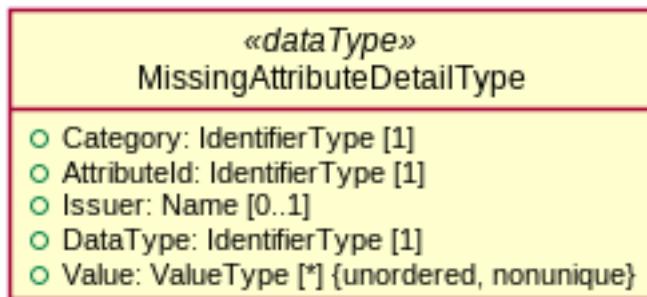
**urn:oasis:names:tc:acal:1.0:status:processing-error**

A PDP MUST NOT return a **StatusDetailType** object 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** property describing the nature of the error.

## 7.44 MissingAttributeDetailType

A **MissingAttributeDetailType** object conveys information about attributes required for policy evaluation that were missing from the request context.

UML definition (class diagram):



A **MissingAttributeDetailType** object contains the following properties:

### **Value** [Any Number]

A sequence of **ValueType** objects specifying the required value(s) of the missing attribute. None of these objects SHALL redefine/override the **DataType** identifier already defined by the **DataType** property below, and SHOULD not have any **DataType** identifier property set at all, if possible.

### **Category** [Required]

An **IdentifierType** value indicating the category of the missing attribute.

### **AttributeId** [Required]

An **IdentifierType** value indicating the attribute identifier of the missing attribute.

### **DataType** [Required]

An **IdentifierType** value indicating the data type of the missing attribute.

### **Issuer** [Optional]

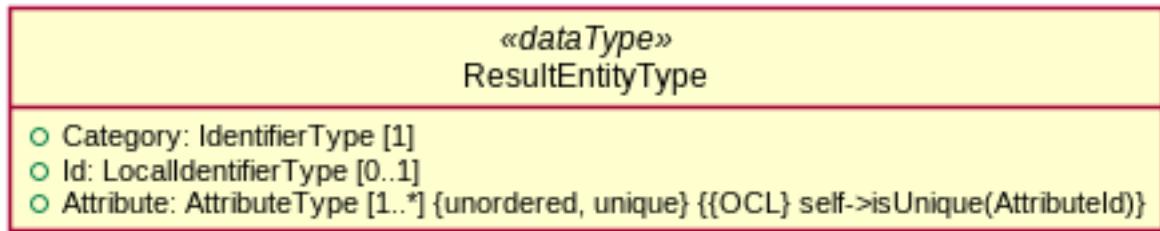
A **Name** value, which if supplied, specifies the required **Issuer** of the missing attribute.

If the PDP includes `ValueType` objects in the `MissingAttributeDetailType` object, then this indicates the acceptable values for that attribute. If no `ValueType` objects are included, then the `MissingAttributeDetailType` object 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.

## 7.45 ResultEntityType

A `ResultEntityType` object contains a sequence of `AttributeType` objects reflecting `RequestAttributeType` objects in a given attribute category from the request that had their `IncludeInResult` properties set to `true`. The `ResultEntityType` objects only appear in a result.

UML definition (class diagram):



A `ResultEntityType` object contains the following properties:

**Category [Required]**

An `IdentifierType` value indicating which attribute category the contained attributes belong to.

**Id [Optional]**

A `String` identifier matching the `Id` property of the corresponding `RequestEntityType` object in the request from which the reflected attributes are obtained.

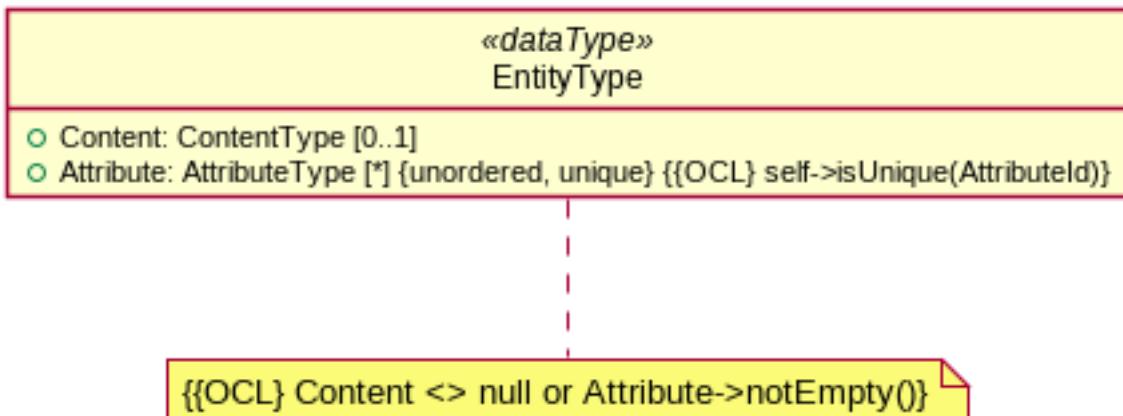
**Attribute [Any Number]**

A sequence of `AttributeType` objects representing `RequestAttributeType` objects from the same attribute category in the request.

## 7.46 EntityType

The `EntityType` object class defines the structure of values of the `urn:oasis:names:tc:acal:1.0:data-type:entity` data type. An `EntityType` object contains a sequence of `AttributeType` objects and/or syntax-specific content (i.e., XML or JSON). The `EntityType` object type is similar to the `RequestEntityType` object type but omits the `Category` property; values of the `urn:oasis:names:tc:acal:1.0:data-type:entity` data type don't self-identify with any particular attribute category.

UML definition (class diagram):



An `EntityType` object contains the following properties:

**Content [Optional]**

A **ContentType** object specifying additional sources of attributes in JSON or XML document format which can be referenced using entity attribute selectors.

**Attribute [Any Number]**

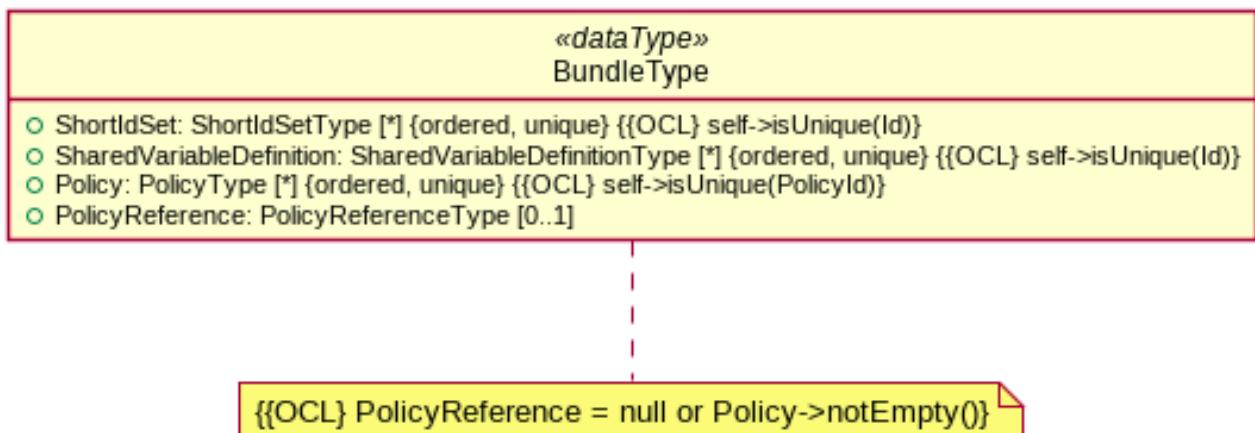
A sequence of **AttributeType** objects.

## 7.47 BundleType

A **BundleType** object type collects together the policies, short identifier sets and shared variables that a PDP uses to evaluate an authorization request along with the starting point for that evaluation. A PDP is said to be defined by a **BundleType** object, although an implementation is not required to physically represent such an object.

A **BundleType** object might also be used to convey policies, short identifier sets or shared variables between ACAL systems without necessarily being complete for the purpose of evaluating authorization requests. Protocols to enable such transfers are outside the scope of this specification.

UML definition (class diagram):



A **BundleType** object contains the following properties:

**ShortIdSet [Any Number]**

A sequence of **ShortIdSetType** objects each defining a set of short identifiers that can be referenced by the policies, shared variable definitions and other short identifier sets in the **BundleType** object. Each object's Id MUST be unique within the sequence.

**SharedVariableDefinition [Any Number]**

A sequence of **SharedVariableDefinitionType** objects, each defining an expression that can be referenced from anywhere in the **BundleType** object where an expression can appear. Each object's Id MUST be unique within the sequence.

**Policy [Any Number]**

A sequence of **PolicyType** objects. Each object's PolicyId MUST be unique within the sequence.

**PolicyReference [Optional]**

If present, a **PolicyReferenceType** object that MUST reference a policy in the **Policy** property. A PDP defined by this **BundleType** object SHALL evaluate every authorization request by evaluating this policy reference (the referenced policy is the entry point of evaluation). See Section 8.13. If this property is absent, then the PDP returns **NotApplicable** for every authorization request. The policy reference MUST specify arguments if the referenced policy is parameterized.

## 8 Functional Requirements

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

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.

## 8.1 Unicode Issues

### 8.1.1 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 ACAL 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.

### 8.1.2 Version of Unicode

The version of Unicode used by ACAL is implementation defined. It is RECOMMENDED that the latest version is used. Also note security issues in Section 10.3.

## 8.2 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.

### 8.2.1 Base PEP

If the decision is **Permit**, then the PEP SHALL permit access. If obligation notices 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 obligation notices 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.

### 8.2.2 Deny-Biased PEP

If the decision is **Permit**, then the PEP SHALL permit access. If obligation notices 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.

### 8.2.3 Permit-Biased PEP

If the decision is **Deny**, then the PEP SHALL deny access. If obligation notices 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.

## 8.3 Identifier Evaluation

Various ACAL artifacts are uniquely identified by URI [RFC2396]. This includes attribute categories, attributes, data types, functions, notices, status codes, combining algorithms. Values of the `IdentifierType` simple type are used to refer to such artifacts, possibly using short identifier names, which are not themselves URIs. Therefore an `IdentifierType` value containing short identifier names must undergo evaluation to convert the value to an absolute URI.

The value of a short identifier may also contain short identifier names. The *expanded value* of the short identifier is the value of the short identifier with any reference to another short identifier (including the curly brackets) replaced by the *expanded value* of the referenced short identifier (thus the expansion is applied recursively).

In the case where the `IdentifierType` value is already an absolute URI then the result of evaluation is that URI.

In the case where the `IdentifierType` value is just a short identifier name then the result of evaluation is the *expanded value* of that short identifier.

In the case where the `IdentifierType` value contains curly brackets then the result of evaluation is the value with any reference to a short identifier replaced by the *expanded value* of the referenced short identifier.

The end result of evaluation MUST be an absolute URI.

### 8.3.1 Identifier Examples (non-normative)

Given the following short identifiers:

```
<ShortId Name="xs" Value="urn:oasis:names:tc:acal:1.0:data-type:"/>
<ShortId Name="string" Value="{xs}string"/>
```

these `IdentifierType` values are all equivalent and evaluate to the URI of the string data type:

```
"urn:oasis:names:tc:acal:1.0:data-type:string"
"string"
"{string}"
"{xs}string"
```

### 8.3.2 Short Identifier Set Usage (non-normative)

The predefined short identifier set is not expected to contain short identifier definitions for all the identifiers an implementation or a deployment of that implementation will be using. In particular, a deployment is likely to have a number of custom ACAL attributes holding information specific to that deployment. Deployments may be part of a community of interest making use of common attributes and categories. Implementations may take advantage of the extension points in ACAL to define proprietary functions and data types, and may define custom categories and custom attributes that are appropriate for their user base. One aim of this specification is give implementors and users the ability to define their own short identifiers to use in these cases. Future ACAL extensions may also provide additional, predefined short identifier sets for any identifiers they introduce.

Ideally, all the short identifiers that a deployment intends to use would be available to a policy or a request by referencing a *single* short identifier set. Because a short identifier set cannot validly be included directly or indirectly more than once, specifications and implementations should avoid creating short identifier sets that reference short identifier sets outside of their purview and instead leave deployments to create the single short identifier set that references all the sets that are relevant to it.

Short identifier names are not globally unique, so implementations should not hard-wire specific short identifiers or short identifier sets so that users have the flexibility to compose and use alternative short identifier sets to work around any name clashes that arise from attempting to incorporate sets from various, independent sources.

The predefined short identifier set is not required to be used but its use is recommended to promote a common understanding of identifiers between deployments. For this reason, the use of alternative names should be minimized, and preferably allowed *in addition to* the predefined set. Users should not exclude the standard set and assign any short identifier names from that set to different URIs.

## 8.4 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 `RequestEntityType` objects of the request context.

#### 8.4.1 Structured Attributes

ValueType objects MAY contain an instance of a structured data type, for example and XML element `<ds:KeyInfo>`. ACAL 1.0 supports several ways for comparing the contents of such elements.

1. In some cases, such elements MAY be compared using one of the ACAL string functions, such as `string-regexp-match`, described below. This requires that the element be given the data type `urn:oasis:names:tc:acal:1.0:data-type:string`. For example, a structured data type that is actually a `ds:KeyInfo/KeyName` would appear in the context as:

```
<Value DataType="urn:oasis:names:tc:acal:1.0:data-type:string">
  &lt;ds:KeyName&gt;jhibbert-key&lt;/ds:KeyName&gt;
</Value>
```

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` property of the appropriate attribute category and an attribute selector MAY be used to select the contents of a leaf sub-element of the structured data type by means of a *path* expression whose syntax and semantics depends on the type of content (e.g. XPath for XML content). That value MAY then be compared using one of the supported ACAL functions appropriate for its data type. This method requires support by the PDP for the specific ACAL Profile that defines the particular kind of attribute selector, and type of *path* expression in particular, for the type of `Content` being used (e.g. XPath Profile for XML content).

#### 8.4.2 Attribute Bags

ACAL defines implicit collections of its data types. ACAL 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/JSON resource or ACAL request context may return more than one value.

An attribute selector uses an expression to specify the selection of data from free-form structured content (e.g. any XML element). The result of the expression evaluation may return multiple values from the content. ACAL also defines attribute designators to have the same matching methodology for attributes in the ACAL 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 ACAL SHALL contain only values that are of the same data type.

#### 8.4.3 Multivalued Attributes

If a single attribute in a request context contains multiple attribute values, then the bag of values resulting from evaluation of the attribute MUST be identical to the bag of values that results from evaluating a context in which each attribute value appears in a separate attribute, with each separate attribute carrying identical meta-data.

#### 8.4.4 Attribute Matching

A named attribute includes specific criteria with which to match attributes in the context. An attribute specifies a `Category` and `DataType`, and a named attribute also specifies an `AttributeId` and optional `Issuer`. A named attribute SHALL match an attribute if the values of their respective `Category`, `AttributeId`, `DataType` and optional `Issuer` properties match. The `Category` property of the named attribute MUST match, by identifier equality, the `Category` property of the corresponding context attribute. The `AttributeId` property of the named attribute MUST match, by identifier equality, the `AttributeId` property of the corresponding context attribute. The `DataType` property of the named attribute MUST match, by identifier equality, the `DataType` property of the corresponding context attribute. If the `Issuer` property is supplied in the named attribute, then it MUST match, using the `urn:oasis:names:tc:acal:1.0:function:string-equal` function, the `Issuer` property of the corresponding context attribute. If the `Issuer` property is not supplied in the named attribute, then the matching of the context attribute to the named attribute SHALL be governed by the `AttributeId` and `DataType` properties alone, regardless of the presence, absence, or actual value of `Issuer` property 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 `Category`, `Path` expression and `DataType` property and other properties as defined by the respective ACAL Profile being used to provide the concrete `AttributeSelectorType` object.

#### 8.4.5 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 the **MustBePresent** property governs whether the attribute designator or attribute selector returns an empty bag or an **Indeterminate** result. If the **MustBePresent** property is **false** (default value), then a missing attribute SHALL result in an empty bag. If the **MustBePresent** property 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 and policies. If the result is **Indeterminate**, then the **AttributeId**, **DataType** and **Issuer** properties of the attribute MAY be listed in the authorization decision as described in Section 8.15. 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.

#### 8.4.6 Environment Attributes

Standard environment attributes are listed in Annex D.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".

#### 8.4.7 Selector Evaluation

An **AttributeSelectorType** or **EntityAttributeSelector** object will be evaluated according to the following processing model.

Note: It is not necessary for an implementation to exactly follow this model. It is only necessary to produce results identical to those that would be produced by following this model.

If the attribute category given by the **Category** property of an **AttributeSelectorType** object is not found or does not have a **Content** property, then the return value is either **Indeterminate** or an empty bag as determined by the **MustBePresent** property.

If the **Expression** property of an **EntityAttributeSelectorType** object evaluates to a value of the **urn:oasis:names:tc:acal:1.0:data-type:entity** data type and that value does not have a **Content** property, then the return value is either **Indeterminate** or an empty bag as determined by the **MustBePresent** property.

If the **Expression** property of an **EntityAttributeSelectorType** object evaluates to a value of the **urn:oasis:names:tc:acal:1.0:data-type:anyURI** data type and an attribute category with that value as its **Category** is not found or does not have a **Content** property, then the return value is either **Indeterminate** or an empty bag as determined by the **MustBePresent** property.

If the designated attribute category or entity value has a **Content** property, then follow the steps below:

1. Parse the value of the **Content's Body** property into the data structure suitable for processing by the concrete type of **AttributeSelector/EntityAttributeSelector** defined by the ACAL Profile in use (e.g. XML content for the XPath Profile). If it is not a valid data structure according to the ACAL Profile, then the attribute selector MUST return **Indeterminate** with status code **urn:oasis:names:tc:acal:1.0:status:syntax-error**.
2. Evaluate the expression given in the **Path** property against the data structure obtained in the previous step, according to the syntax and semantics of the respective ACAL Profile in use (as indicated by the concrete subtype of **AttributeSelectorType** being used).
3. The result of step 3 is converted to a bag of values of the data type specified by the **DataType** property as follows:

If the result is a Boolean and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:boolean`, then convert the result using the `xs:boolean()` constructor function from [XF] Section 18.1.

If the result is a string and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:string`, then convert the result using the `xs:string()` constructor function from [XF] Section 18.1.

If the result is a number and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:integer`, then convert the result using the `xs:integer()` constructor function from [XF] Section 18.1.

If the result is a number and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:double`, then convert the result using the `xs:double()` constructor function from [XF] Section 18.1.

If the result is a node-set and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:boolean`, then convert the string value of each node using the `xs:boolean()` constructor function from [XF] Section 18.1.

If the result is a node-set and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:string`, then convert the string value of each node using the `xs:string()` constructor function from [XF] Section 18.1.

If the result is a node-set and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:integer`, then convert the string value of each node using the `xs:integer()` constructor function from [XF] Section 18.1.

If the result is a node-set and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:double`, then convert the string value of each node using the `xs:double()` constructor function from [XF] Section 18.1.

If the result is a node-set and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:dateTime`, then convert the string value of each node using the `xs:dateTime()` constructor function from [XF] Section 18.1.

If the result is a node-set and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:date`, then convert the string value of each node using the `xs:date()` constructor function from [XF] Section 18.1.

If the result is a node-set and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:time`, then convert the string value of each node using the `xs:time()` constructor function from [XF] Section 18.1.

If the result is a node-set and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:hexBinary`, then convert the string value of each node using the `xs:hexBinary()` constructor function from [XF] Section 18.1.

If the result is a node-set and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:base64Binary`, then convert the string value of each node using the `xs:base64Binary()` constructor function from [XF] Section 18.1.

If the result is a node-set and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:anyURI`, then convert the string value of each node using the `xs:anyURI()` constructor function from [XF] Section 18.1.

If the result is a node-set and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:yearMonthDuration`, then convert the string value of each node using the `xs:yearMonthDuration()` constructor function from [XF] Section 18.1.

If the result is a node-set and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:dayTimeDuration`, then convert the string value of each node using the `xs:dayTimeDuration()` constructor function from [XF] Section 18.1.

If the result is a node-set and every node is an element node and the specified data type is `urn:oasis:names:tc:acal:1.0:data-type:entity`, then convert each node to an `EntityType` object. Each object SHALL have a `Content` property and SHALL NOT have an `Attribute` property. The child element of the `Content`'s `Body` property SHALL be a copy of the element corresponding to the node, along with its entire content, plus whatever namespace declarations from ancestor elements as are required to define namespace prefixes used in the content. Namespace declarations from ancestor elements that are not visibly used in the content MAY be added.

If the data type is one of the types referred to above and the result of step 3 does not satisfy any of the cases, then the attribute selector MUST return `Indeterminate` with status code `urn:oasis:names:tc:acal:1.0:status:syntax-error`.

If the data type is not one of the types referred to above, then the return values SHALL be constructed from the node-set in a manner specified by the particular data type extension specification. If the data type extension does not specify an appropriate constructor function, then the attribute selector MUST return `Indeterminate` with status code `urn:oasis:names:tc:acal:1.0:status:syntax-error`.

If an error occurs when converting the values returned by the expression to the specified data type, then the result of the attribute selector MUST be `Indeterminate`, with status code `urn:oasis:names:tc:acal:1.0:status:processing-error`

If the result of step 3 is an empty node-set, then the return value is either **Indeterminate** with status code `urn:oasis:names:tc:acal:1.0:status:syntax-error`, or an empty bag, as determined by the **MustBePresent** property.

## 8.5 Expression Evaluation

ACAL specifies expressions in terms of objects of the object types listed below, of which the **ApplyType** objects recursively compose greater expressions. Expressions SHALL be type correct and satisfy cardinality requirements (i.e., single value versus bag). This means that the types of each of the objects contained within an **ApplyType** object SHALL agree with the respective argument types and cardinalities of the function that is named by the **FunctionId** property. The resultant type and cardinality of the **ApplyType** object SHALL be the resultant type and cardinality of the function. ACAL 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.

An ACAL expression is a choice between the following object types:

- **ValueType**
- **AttributeDesignatorType**
- **EntityAttributeDesignatorType**
- **AttributeSelectorType**
- **EntityAttributeSelectorType**
- **ApplyType**
- **FunctionType**
- **VariableReferenceType**
- **SharedVariableReferenceType**
- **QuantifiedExpressionType**

## 8.6 Arithmetic Evaluation

**IEEE 754 [IEEE754]** specifies how to evaluate arithmetic functions in a context, which specifies defaults for precision, rounding, etc. ACAL 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)

## 8.7 Target Evaluation

The target value SHALL be **Match** if the **Target** property is absent or the expression specified in the target evaluates to **true**. Otherwise, if the expression evaluates to **false**, then the target SHALL be **No Match**. Otherwise, the target SHALL be **Indeterminate**. The target match table is shown in Table 1.

**Table 1:** Target Match Table

Expression value	Target value
<b>true</b>	<b>Match</b>
<b>false</b>	<b>No Match</b>
<b>Indeterminate</b>	<b>Indeterminate</b>

## 8.8 VariableReference Evaluation

A **VariableReferenceType** object references a single **VariableDefinitionType** object that has a matching **VariableId** property value and is in the **VariableDefinition** property of an enclosing **RuleType** or **PolicyType** object. If no such **VariableDefinitionType** object exists, then the reference is invalid. The PDP MUST detect invalid references either at policy loading time or during runtime evaluation. If the PDP detects an invalid reference during runtime the variable reference evaluates to **Indeterminate** with status code `urn:oasis:names:tc:acal:1.0:status:processing-error`.

In any place where a `VariableReferenceType` object occurs, it has the effect as if the `ExpressionType` object defined in the `VariableDefinitionType` object replaces the `VariableReferenceType` object. Any evaluation scheme that preserves this semantic is acceptable. For instance, the expression in the `VariableDefinitionType` object may be evaluated to a particular value and cached for multiple references without consequence (i.e., the value of an `ExpressionType` object remains the same for the entire policy evaluation). This characteristic is one of the benefits of ACAL being a declarative language.

Circular references are 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:acal:1.0:status:processing-error`.

## 8.9 Condition Evaluation

The condition value `SHALL` be `true` if the `Condition` property is absent, or if the expression contained in the `Condition` property evaluates to `true`. Its value `SHALL` be `false` if the expression evaluates to `false`. The condition value `SHALL` be `Indeterminate`, if the expression evaluates to `Indeterminate`.

## 8.10 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 that are defined in terms of the extended `Indeterminate` make use of the additional information to allow for better treatment of errors in the processing.

The final decision returned by a PDP cannot be an extended `Indeterminate`. Any such decision at the top level policy 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 and policy evaluation.

## 8.11 Rule Evaluation

A rule has a value that can be calculated by evaluating the rule's condition. The rule truth table is shown in Table 4.

**Table 4:** Rule Truth Table

Condition	Rule Value
<code>true</code>	<code>Effect</code>
<code>false</code>	<code>NotApplicable</code>
<code>Indeterminate</code>	<code>Indeterminate{P}</code> if the Effect is <code>Permit</code> , or <code>Indeterminate{D}</code> if the Effect is <code>Deny</code>

## 8.12 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 evaluation of the specified combining algorithm on the contained policies and rules.

The policy truth table is shown in Table 5.

**Table 5:** Policy Truth Table

Target	Combining Algorithm Value	Policy Value
Match	NotApplicable	NotApplicable
Match	Permit	Permit
Match	Deny	Deny
Match	Indeterminate	Indeterminate{DP}
Match	Indeterminate{DP}	Indeterminate{DP}
Match	Indeterminate{P}	Indeterminate{P}
Match	Indeterminate{D}	Indeterminate{D}
No-match	Don't care	NotApplicable
Indeterminate	NotApplicable	NotApplicable
Indeterminate	Permit	Indeterminate{P}
Indeterminate	Deny	Indeterminate{D}
Indeterminate	Indeterminate	Indeterminate{DP}
Indeterminate	Indeterminate{DP}	Indeterminate{DP}
Indeterminate	Indeterminate{P}	Indeterminate{P}
Indeterminate	Indeterminate{D}	Indeterminate{D}

## 8.13 PolicyReference Evaluation

A policy reference is evaluated by resolving the reference and evaluating the referenced policy.

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

A policy 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:acal:1.0:status:processing-error`.

## 8.14 Hierarchical Resources

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

## 8.15 Authorization Decision

In relation to a particular decision request, the PDP is defined by a set of policies, a set of shared variable definitions, sets of short identifiers and a policy reference that specifies the starting point for evaluating the request. Conceptually, these items can be collected together as a single `BundleType` object, although an implementation is free to represent and organize this information in any way it chooses as long as it produces the same result as is specified here.

The PDP SHALL return a response context as if it had evaluated the policy reference in the manner specified in Section 7 and Section 8. The PDP MUST return a response context, with one `Decision` property of value `Permit`, `Deny`, `Indeterminate` or `NotApplicable`.

If the PDP cannot make a decision, then an `Indeterminate` `Decision` property SHALL be returned.

## 8.16 Notices

A rule or policy may contain one or more notice expressions. When such a rule or policy is evaluated, the notice expression SHALL be evaluated to a notice, which SHALL be passed up to the next level of evaluation (the enclosing or referencing policy or authorization decision) only if the result of the rule or policy being evaluated matches the value of the `AppliesTo` property of the notice (or matches either `Permit` or `Deny` if the `AppliesTo` property is absent) and the notice expression does not have a condition or it has a condition that evaluates to `true`. If the condition in a notice expression with a matching `AppliesTo` property evaluates to `Indeterminate`, then the whole rule or policy SHALL be `Indeterminate`. If the condition is absent or evaluates to `true` and any of the attribute assignment expressions in the notice expression evaluates to `Indeterminate`, then the whole rule or policy SHALL be `Indeterminate`. If the `AppliesTo` property is present and does not match the result of the combining algorithm or the rule evaluation, then any `Indeterminate` evaluation in a notice expression has no effect.

As a consequence of this procedure, notices SHALL NOT be returned to the PEP if the rules or policies 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 or policy does not match the decision resulting from evaluating an enclosing policy, or if the notice expression has a condition that evaluates to `false`.

If the PDP's evaluation is viewed as a tree of rules and policies, each of which returns `Permit` or `Deny`, then the collection of notices returned by the PDP to the PEP will include only the notices 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.

Also see Section 8.2.

## 8.17 Exception Handling

ACAL specifies behavior for the PDP in the following situations.

### 8.17.1 Unsupported Functionality

If the PDP attempts to evaluate a policy that contains an optional object type or function that the PDP does not support, then the PDP SHALL return a `DecisionType` value of `Indeterminate`. If a `StatusType` object is also returned, then its `Value` property SHALL be `urn:oasis:names:tc:acal:1.0:status:syntax-error` in the case of an unsupported object type, and `urn:oasis:names:tc:acal:1.0:status:processing-error` in the case of an unsupported function.

### 8.17.2 Syntax and Type Errors

If a policy that contains invalid syntax is evaluated by the ACAL PDP at the time a decision request is received, then the result of that policy SHALL be `Indeterminate` with a status code of `urn:oasis:names:tc:acal:1.0:status:syntax-error`.

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

### 8.17.3 Missing Attributes

The absence of matching attributes in the request context for any of the attribute designators or attribute selectors that are found in the policy potentially results in the policy evaluating to `Indeterminate`, if the designator or selector has the `MustBePresent` property set to `true`, as described in Section 7.17 and Section 7.20 and potentially results in a `Decision` property containing the `Indeterminate` value. In this case, if a status code is supplied, then the value

`urn:oasis:names:tc:acal: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 `StatusType` object MAY list the names and data types of any attributes that are needed by the PDP to refine its decision (see Section 7.44). A PEP MAY resubmit a refined request context in response to a `Decision` property contents of `Indeterminate` with a status code of

`urn:oasis:names:tc:acal: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` property contents of `Indeterminate`, with a status code of

`urn:oasis:names:tc:acal: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.

## 8.18 Identifier Equality

ACAL makes use of URIs 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.

Values of the `IdentifierType` simple type may use short identifiers names as aliases for URIs. Where it is necessary to determine identifier equality, values of the `IdentifierType` simple type MUST first be evaluated

according to Section 8.3 and then the comparison SHALL be performed using the resulting URI. This applies to all instances of `IdentifierType`.

This definition of equality MUST also be used by the following URI identifiers (that are not instances of `IdentifierType`):

- the `PolicyId` property in a `PolicyType` object,
- the `Id` property in a `SharedVariableDefinitionType` object and
- the `Id` property in an `IdReferenceType` object.

The following is a list of the string identifiers that MUST use this definition of equality:

- the `Issuer` property in a `NamedAttributeDesignatorType` object,
- the `Issuer` property in a `MissingAttributeDetailType` object,
- the `Issuer` property in an `AttributeType` object,
- the `Issuer` property in an `AttributeAssignmentExpressionType` object,
- the `Id` property in a `RuleType` object,
- the `VariableId` property in a `VariableDefinitionType` object and
- the `VariableId` property in a `VariableReferenceType` object.

It is RECOMMENDED that extensions to ACAL 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.

## 8.19 Short Identifiers in Responses

A `ResponseType` object may contain references to short identifier sets, and short identifier names from those sets may be used within the response. The PDP can freely choose what short identifier sets it uses in the response, if any. There are two cases that require special care.

ACAL attributes in the request that have the `IncludeInResult` property set to `true` are duplicated in the response. If those attributes contain short identifier names, then those short identifier names are interpreted with respect to the short identifier sets referenced by the request, which may be different from the short identifier sets chosen by the PDP for the response.

Notices in the response originate from notice expressions in policies. If those notice expressions contain short identifier names, then those short identifier names are interpreted with respect to the short identifier sets referenced by the policies, which may also be different from the short identifier sets chosen by the PDP for the response.

The PDP is responsible for ensuring that any short identifier names in the response reflect the correct URI when they are evaluated according to Section 8.3 in the presence of the short identifier sets it has chosen. An implementation strategy for achieving this is for the PDP and context handler to evaluate any `IdentifierType` values as they are encountered and store them in internal memory as absolute URIs. Then when the response is composed the absolute URIs can be replaced by appropriate short identifier names from the short identifier sets chosen by the PDP.

If the PDP chooses the same short identifier sets referenced by the request, then it has the advantage that they are sets known by the PEP and any attributes included in the result are more likely to have their original short identifier names restored.

---

## 9 ACAL Extensibility Points (non-normative)

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

## 9.1 Extensible Properties

The following `IdentifierType` properties have values that evaluate to URIs. They may be extended by the creation of new URIs associated with new semantics.

`Category`,  
`AttributeId`,  
`DataType`,  
`FunctionId` and the `Id` property of `FunctionType`,  
the `Id` property of `NoticeType` and `NoticeExpressionType`,  
`CombiningAlgId`,  
`StatusCodes`.

See Section 7 for definitions of these properties.

## 9.2 Structured Attributes

*This section needs a major rewrite to de-emphasize XML and incorporate the entity data type.*

A `ValueType` object MAY contain an instance of a structured XML data type. Section 8.4.1 describes a number of standard techniques to identify data items within such a structured attribute. Listed here are some additional techniques that require ACAL extensions.

1. For a given structured data type, a community of ACAL 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 ACAL-defined 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 `AttributeType` objects. Each such `AttributeType` object can be compared using the ACAL-defined functions. Using this method, the structured data type itself never appears in a `ValueType` object.
  2. A community of ACAL 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.
- 

# 10 Safety, Security, and Data Protection Considerations

This section identifies possible security and privacy compromise scenarios that should be considered when implementing an ACAL-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.

## 10.1 Threat Model

We assume here that the adversary has access to the communication channel between the ACAL 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 ACAL 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 ACAL system.

### 10.1.1 Unauthorized Disclosure

ACAL 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.

#### **10.1.2 Message Replay**

A message replay attack is one in which the adversary records and replays legitimate messages between ACAL 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.

#### **10.1.3 Message Insertion**

A message insertion attack is one in which the adversary inserts messages in the sequence of messages between ACAL 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 certificate. In order to be effective, it is necessary to confirm that the certificate subject is authorized to send the message.

#### **10.1.4 Message Deletion**

A message deletion attack is one in which the adversary deletes messages in the sequence of messages between ACAL actors. Message deletion may lead to denial of service. However, a properly designed ACAL 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.

#### **10.1.5 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.

#### **10.1.6 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 attributes 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 8.2, a PEP must deny access unless it receives an explicit **Permit** authorization decision.

### 10.1.7 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 ACAL 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 administers all policies, but when administration is distributed, as ACAL allows, this type of situation must be explicitly guarded against.

### 10.1.8 Denial of Service

A denial of service attack is one in which the adversary overloads an ACAL actor with excessive computations or network traffic such that legitimate users cannot access the services provided by the actor.

The `urn:oasis:names:tc:acal:1.0: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.

## 10.2 Safeguards

### 10.2.1 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.

## 10.2.2 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 `StatusType` object should be used to return values of missing attributes only when exposure of the identities of those attributes will not compromise security.

## 10.2.3 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.

**10.2.3.1 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 ACAL policies are outside the scope of the ACAL standard. While it is important to ensure that the integrity and confidentiality of policies 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.

**10.2.3.2 Statement Level Confidentiality** In some cases, an implementation may want to encrypt only parts of an ACAL policy.

The particular policy encryption mechanism and syntax for encrypted policy or policy parts will be described by the ACAL Representation Profiles (e.g. W3C XML Encryption standard for XACML - XML Representation Profile).

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.

## 10.2.4 Policy Integrity

ACAL policies used by the PDP to evaluate the request context are the heart of the system. Therefore, maintaining their integrity is essential. There are two aspects to maintaining the integrity of the policies. One is to ensure that the policies have not been altered since they were originally created by the PAP. The other is to ensure that policies 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.

The particular policy signature mechanism and syntax for signed policy or policy parts will be described by the ACAL Representation Profiles (W3C XML Signature Syntax and Processing standard for XACML - XML Representation Profile).

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.

### 10.2.5 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 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 <https://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.

### 10.2.6 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.

### 10.2.7 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 ACAL 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 ACAL 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 ACAL. The decision regarding whether, how, and when to deploy such mechanisms is left to the implementers associated with the environment.

## 10.3 Unicode Security Issues

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

## 10.4 Identifier Equality

Section 8.18 defines the identifier equality operation for ACAL. This definition of equality does not do any kind of canonicalization or escaping of characters. The identifiers defined in the ACAL 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 %.

---

# 11 Conformance

## 11.1 Introduction

The ACAL specification addresses the following aspect of conformance:

The ACAL 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 to this specification.

## 11.2 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 ACAL 1.0. 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 Section 7, Section 8, Annex C, Annex D and Annex E where they apply to implemented items in the following tables.

Many of these items are associated with versions of XACML preceding ACAL but have been assigned new identifiers with the `urn:oasis:names:tc:acal:1.0:` prefix. The older XACML identifiers have been listed in the tables as deprecated identifiers. Implementations MUST support a new identifier defined in this specification but MAY recognize the corresponding deprecated identifier as equivalent. It is RECOMMENDED that these deprecated identifiers not be used in new policies and requests; they are planned to be removed in a subsequent version of ACAL. Note that some items appear to be carried over from a preceding version of XACML but do not list the XACML identifier. This is because ACAL has redefined the item in some way that means it is no longer identical to the original definition in XACML, and so the identifiers can no longer be considered equivalent. Items new to ACAL 1.0 will also not list an XACML identifier.

### 11.2.1 Object Types

The implementation MUST support the object types that are marked M.

Object Type	M/O
ApplyType	M
AttributeAssignmentExpressionType	M
AttributeAssignmentType	M
AttributeDesignatorType	M
AttributeSelectorType	O
AttributeType	M
BaseAttributeSelectorType	O
BooleanExpressionType	M
BundleType	O
ContentType	O
DecisionType	M
DescriptionType	M
EntityAttributeDesignatorType	M
EntityAttributeSelectorType	O
EntityType	M
ExactMatchIdReferenceType	O
ExpressionType	M

Object Type	M/O
FunctionType	M
MissingAttributeDetailType	M
MultiRequestsType	O
NamedAttributeDesignatorType	M
NoticeExpressionType	M
NoticeType	M
PolicyDefaultsType	O
PolicyIssuerType	O
ParameterType	M
PolicyReferenceType	M
PolicyType	M
QuantifiedExpressionType	O
RequestAttributeType	M
RequestDefaultsType	O
RequestEntityReferenceType	O
RequestEntityType	M
RequestReferenceType	O
RequestType	M
ResponseType	M
ResultEntityType	M
ResultType	M
RuleType	M
SharedVariableDefinitionType	O
SharedVariableReferenceType	M
ShortIdSetType	M
ShortIdType	M
StatusType	M
StatusCodeType	M
StatusDetailType	O
ValueType	M
VariableDefinitionType	M
VariableReferenceType	M

### 11.2.2 Identifier Prefixes

The following identifier prefixes are reserved by ACAL.

Prefix
urn:oasis:names:tc:acal:1.0
urn:oasis:names:tc:xacml:4.0
urn:oasis:names:tc:xacml:3.0
urn:oasis:names:tc:xacml:2.0
urn:oasis:names:tc:xacml:1.0

### 11.2.3 Algorithms

The implementation MUST include the combining algorithms associated with the following identifiers that are marked M

Algorithm	M/O
urn:oasis:names:tc:acal:1.0:combining-algorithm:deny-overrides	M
urn:oasis:names:tc:acal:1.0:combining-algorithm:permit-overrides	M
urn:oasis:names:tc:acal:1.0:combining-algorithm:first-applicable	M
urn:oasis:names:tc:acal:1.0:combining-algorithm:ordered-deny-overrides	M
urn:oasis:names:tc:acal:1.0:combining-algorithm:ordered-permit-overrides	M
urn:oasis:names:tc:acal:1.0:combining-algorithm:deny-unless-permit	M
urn:oasis:names:tc:acal:1.0:combining-algorithm:permit-unless-deny	M

#### 11.2.4 Status Codes

Implementation support for the `StatusType` object is optional, but if the object is supported, then the following status codes must be supported and must be used in the way ACAL has specified.

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

#### 11.2.5 Environment Attributes

The implementation MUST support the attributes associated with the following identifiers as specified by ACAL. 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.

Identifier	M/O	Deprecated Identifier
urn:oasis:names:tc:acal:1.0:environment:current-time	M	urn:oasis:names:tc:xacml:1.0:environment:current-time
urn:oasis:names:tc:acal:1.0:environment:current-date	M	urn:oasis:names:tc:xacml:1.0:environment:current-date
urn:oasis:names:tc:acal:1.0:environment:current-dateTime	M	urn:oasis:names:tc:xacml:1.0:environment:current-dateT

#### 11.2.6 Attributes and Categories

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

Identifier	M/O	Deprecated Identifier
urn:oasis:names:tc:acal:1.0:subject:authn-locality:dns-name	O	urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name
urn:oasis:names:tc:acal:1.0:subject:authn-locality:ip-address	O	urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address
urn:oasis:names:tc:acal:1.0:subject:authentication-method	O	urn:oasis:names:tc:xacml:1.0:subject:authentication-method
urn:oasis:names:tc:acal:1.0:subject:authentication-time	O	urn:oasis:names:tc:xacml:1.0:subject:authentication-time
urn:oasis:names:tc:acal:1.0:subject:key-info	O	urn:oasis:names:tc:xacml:1.0:subject:key-info
urn:oasis:names:tc:acal:1.0:subject:request-time	O	urn:oasis:names:tc:xacml:1.0:subject:request-time
urn:oasis:names:tc:acal:1.0:subject:session-start-time	O	urn:oasis:names:tc:xacml:1.0:subject:session-start-time
urn:oasis:names:tc:acal:1.0:subject:subject-id	O	urn:oasis:names:tc:xacml:1.0:subject:subject-id
urn:oasis:names:tc:acal:1.0:subject:subject-id-qualifier	O	urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier
urn:oasis:names:tc:acal:1.0:resource:resource-location	O	urn:oasis:names:tc:xacml:1.0:resource:resource-location
urn:oasis:names:tc:acal:1.0:resource:resource-id	M	urn:oasis:names:tc:xacml:1.0:resource:resource-id
urn:oasis:names:tc:acal:1.0:resource:simple-file-name	O	urn:oasis:names:tc:xacml:1.0:resource:simple-file-name
urn:oasis:names:tc:acal:1.0:resource:target-namespace	O	urn:oasis:names:tc:xacml:2.0:resource:target-namespace
urn:oasis:names:tc:acal:1.0:action:action-id	O	urn:oasis:names:tc:xacml:1.0:action:action-id
urn:oasis:names:tc:acal:1.0:action:action-namespace	O	urn:oasis:names:tc:xacml:1.0:action:action-namespace
urn:oasis:names:tc:acal:1.0:action:implied-action	O	urn:oasis:names:tc:xacml:1.0:action:implied-action

Identifier	M/O	Deprecated Identifier
urn:oasis:names:tc:acal:1.0:subject-category:access-subject	M	urn:oasis:names:tc:xacml:1.0:subject-category:access-subject
urn:oasis:names:tc:acal:1.0:subject-category:codebase	O	urn:oasis:names:tc:xacml:1.0:subject-category:codebase
urn:oasis:names:tc:acal:1.0:subject-category:intermediary-subject	O	urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject
urn:oasis:names:tc:acal:1.0:subject-category:recipient-subject	O	urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject
urn:oasis:names:tc:acal:1.0:subject-category:requesting-machine	O	urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine

#### 11.2.7 Data Types

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

Identifier	M/O	Deprecated Identifier
urn:oasis:names:tc:acal:1.0:data-type:string	M	https://www.w3.org/2001/XMLSchema#string
urn:oasis:names:tc:acal:1.0:data-type:boolean	M	https://www.w3.org/2001/XMLSchema#boolean
urn:oasis:names:tc:acal:1.0:data-type:integer	M	https://www.w3.org/2001/XMLSchema#integer
urn:oasis:names:tc:acal:1.0:data-type:double	M	https://www.w3.org/2001/XMLSchema#double
urn:oasis:names:tc:acal:1.0:data-type:time	M	https://www.w3.org/2001/XMLSchema#time
urn:oasis:names:tc:acal:1.0:data-type:date	M	https://www.w3.org/2001/XMLSchema#date
urn:oasis:names:tc:acal:1.0:data-type:dateTime	M	https://www.w3.org/2001/XMLSchema#dateTime
urn:oasis:names:tc:acal:1.0:data-type:dayTimeDuration	M	https://www.w3.org/2001/XMLSchema#dayTimeDuration
urn:oasis:names:tc:acal:1.0:data-type:yearMonthDuration	M	https://www.w3.org/2001/XMLSchema#yearMonthDuration
urn:oasis:names:tc:acal:1.0:data-type:anyURI	M	https://www.w3.org/2001/XMLSchema#anyURI
urn:oasis:names:tc:acal:1.0:data-type:hexBinary	M	https://www.w3.org/2001/XMLSchema#hexBinary
urn:oasis:names:tc:acal:1.0:data-type:base64Binary	M	https://www.w3.org/2001/XMLSchema#base64Binary
urn:oasis:names:tc:acal:1.0:data-type:rfc822Name	M	urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name
urn:oasis:names:tc:acal:1.0:data-type:x500Name	M	urn:oasis:names:tc:xacml:1.0:data-type:x500Name
urn:oasis:names:tc:acal:1.0:data-type:ipAddress	M	urn:oasis:names:tc:xacml:2.0:data-type:ipAddress
urn:oasis:names:tc:acal:1.0:data-type:dnsName	M	urn:oasis:names:tc:xacml:2.0:data-type:dnsName
urn:oasis:names:tc:acal:1.0:data-type:entity	M	urn:oasis:names:tc:xacml:3.0:data-type:entity

### 11.2.8 Functions

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

Function	M/O	Deprecated Identifier
urn:oasis:names:tc:acal:1.0:function:string-equal	M	urn:oasis:names:tc:xacml:1.0:function:string-equal
urn:oasis:names:tc:acal:1.0:function:boolean-equal	M	urn:oasis:names:tc:xacml:1.0:function:boolean-equal
urn:oasis:names:tc:acal:1.0:function:integer-equal	M	urn:oasis:names:tc:xacml:1.0:function:integer-equal
urn:oasis:names:tc:acal:1.0:function:double-equal	M	urn:oasis:names:tc:xacml:1.0:function:double-equal
urn:oasis:names:tc:acal:1.0:function:date-equal	M	urn:oasis:names:tc:xacml:1.0:function:date-equal
urn:oasis:names:tc:acal:1.0:function:time-equal	M	urn:oasis:names:tc:xacml:1.0:function:time-equal
urn:oasis:names:tc:acal:1.0:function:dateTime-equal	M	urn:oasis:names:tc:xacml:1.0:function:dateTime-equal
urn:oasis:names:tc:acal:1.0:function:dayTimeDuration-equal	M	urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-equal
urn:oasis:names:tc:acal:1.0:function:yearMonthDuration-equal	M	urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-equal
urn:oasis:names:tc:acal:1.0:function:string-equal-ignore-case	M	urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case
urn:oasis:names:tc:acal:1.0:function:anyURI-equal	M	urn:oasis:names:tc:xacml:1.0:function:anyURI-equal
urn:oasis:names:tc:acal:1.0:function:x500Name-equal	M	urn:oasis:names:tc:xacml:1.0:function:x500Name-equal
urn:oasis:names:tc:acal:1.0:function:rfc822Name-equal	M	urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal
urn:oasis:names:tc:acal:1.0:function:hexBinary-equal	M	urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal
urn:oasis:names:tc:acal:1.0:function:base64Binary-equal	M	urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal
urn:oasis:names:tc:acal:1.0:function:integer-add	M	urn:oasis:names:tc:xacml:1.0:function:integer-add
urn:oasis:names:tc:acal:1.0:function:double-add	M	urn:oasis:names:tc:xacml:1.0:function:double-add
urn:oasis:names:tc:acal:1.0:function:integer-subtract	M	urn:oasis:names:tc:xacml:1.0:function:integer-subtract
urn:oasis:names:tc:acal:1.0:function:double-subtract	M	urn:oasis:names:tc:xacml:1.0:function:double-subtract
urn:oasis:names:tc:acal:1.0:function:integer-multiply	M	urn:oasis:names:tc:xacml:1.0:function:integer-multiply
urn:oasis:names:tc:acal:1.0:function:double-multiply	M	urn:oasis:names:tc:xacml:1.0:function:double-multiply
urn:oasis:names:tc:acal:1.0:function:integer-divide	M	urn:oasis:names:tc:xacml:1.0:function:integer-divide
urn:oasis:names:tc:acal:1.0:function:double-divide	M	urn:oasis:names:tc:xacml:1.0:function:double-divide
urn:oasis:names:tc:acal:1.0:function:integer-mod	M	urn:oasis:names:tc:xacml:1.0:function:integer-mod
urn:oasis:names:tc:acal:1.0:function:integer-abs	M	urn:oasis:names:tc:xacml:1.0:function:integer-abs
urn:oasis:names:tc:acal:1.0:function:double-abs	M	urn:oasis:names:tc:xacml:1.0:function:double-abs
urn:oasis:names:tc:acal:1.0:function:round	M	urn:oasis:names:tc:xacml:1.0:function:round
urn:oasis:names:tc:acal:1.0:function:floor	M	urn:oasis:names:tc:xacml:1.0:function:floor
urn:oasis:names:tc:acal:1.0:function:string-normalize-space	M	urn:oasis:names:tc:xacml:1.0:function:string-normalize-space
urn:oasis:names:tc:acal:1.0:function:string-normalize-to-lower-case	M	urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case
urn:oasis:names:tc:acal:1.0:function:double-to-integer	M	urn:oasis:names:tc:xacml:1.0:function:double-to-integer
urn:oasis:names:tc:acal:1.0:function:integer-to-double	M	urn:oasis:names:tc:xacml:1.0:function:integer-to-double
urn:oasis:names:tc:acal:1.0:function:or	M	urn:oasis:names:tc:xacml:1.0:function:or
urn:oasis:names:tc:acal:1.0:function:and	M	urn:oasis:names:tc:xacml:1.0:function:and
urn:oasis:names:tc:acal:1.0:function:n-of	M	urn:oasis:names:tc:xacml:1.0:function:n-of







Function	M/O	Deprecated Identifier
urn:oasis:names:tc:acal:1.0:function:date-union	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:date-subset	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:date-set-equals	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:dateTime-intersection	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:dateTime-at-least-one-member-of	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:dateTime-union	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:dateTime-subset	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:dateTime-set-equals	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:anyURI-intersection	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:anyURI-at-least-one-member-of	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:anyURI-union	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:anyURI-subset	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:anyURI-set-equals	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:hexBinary-intersection	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:hexBinary-at-least-one-member-of	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:hexBinary-union	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:hexBinary-subset	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:hexBinary-set-equals	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:base64Binary-intersection	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:base64Binary-at-least-one-member-of	M	urn:oasis:names:tc:acal:1.0:func
urn:oasis:names:tc:acal:1.0:function:base64Binary-union	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:base64Binary-subset	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:base64Binary-set-equals	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:dayTimeDuration-intersection	M	urn:oasis:names:tc:xacml:3.0:func
urn:oasis:names:tc:acal:1.0:function:dayTimeDuration-at-least-one-member-of	M	urn:oasis:names:tc:xacml:3.0:func
urn:oasis:names:tc:acal:1.0:function:dayTimeDuration-union	M	urn:oasis:names:tc:xacml:3.0:func
urn:oasis:names:tc:acal:1.0:function:dayTimeDuration-subset	M	urn:oasis:names:tc:xacml:3.0:func
urn:oasis:names:tc:acal:1.0:function:dayTimeDuration-set-equals	M	urn:oasis:names:tc:xacml:3.0:func
urn:oasis:names:tc:acal:1.0:function:yearMonthDuration-intersection	M	urn:oasis:names:tc:xacml:3.0:func
urn:oasis:names:tc:acal:1.0:function:yearMonthDuration-at-least-one-member-of	M	urn:oasis:names:tc:xacml:3.0:func
urn:oasis:names:tc:acal:1.0:function:yearMonthDuration-union	M	urn:oasis:names:tc:xacml:3.0:func
urn:oasis:names:tc:acal:1.0:function:yearMonthDuration-subset	M	urn:oasis:names:tc:xacml:3.0:func
urn:oasis:names:tc:acal:1.0:function:yearMonthDuration-set-equals	M	urn:oasis:names:tc:xacml:3.0:func
urn:oasis:names:tc:acal:1.0:function:x500Name-intersection	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:x500Name-at-least-one-member-of	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:x500Name-union	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:x500Name-subset	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:x500Name-set-equals	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:rfc822Name-intersection	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:rfc822Name-at-least-one-member-of	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:rfc822Name-union	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:rfc822Name-subset	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:rfc822Name-set-equals	M	urn:oasis:names:tc:xacml:1.0:func
urn:oasis:names:tc:acal:1.0:function:string-minimum	O	
urn:oasis:names:tc:acal:1.0:function:string-maximum	O	
urn:oasis:names:tc:acal:1.0:function:integer-minimum	O	
urn:oasis:names:tc:acal:1.0:function:integer-maximum	O	
urn:oasis:names:tc:acal:1.0:function:integer-sum	O	
urn:oasis:names:tc:acal:1.0:function:integer-average	O	
urn:oasis:names:tc:acal:1.0:function:double-minimum	O	
urn:oasis:names:tc:acal:1.0:function:double-maximum	O	
urn:oasis:names:tc:acal:1.0:function:double-sum	O	
urn:oasis:names:tc:acal:1.0:function:double-average	O	
urn:oasis:names:tc:acal:1.0:function:dateTime-minimum	O	
urn:oasis:names:tc:acal:1.0:function:dateTime-maximum	O	
urn:oasis:names:tc:acal:1.0:function:date-minimum	O	
urn:oasis:names:tc:acal:1.0:function:date-maximum	O	
urn:oasis:names:tc:acal:1.0:function:access-permitted	O	

---

## **Annex A License, Document Status and Notices**

(This annex forms an integral part of this Specification.)

### **A.1 Document 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. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at <https://groups.oasis-open.org/communities/tc-community-home2?CommunityKey=67afe552-0921-49b7-9a85-018dc7d3ef1d#technical>.

TC members should send comments on this document to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at <https://www.oasis-open.org/committees/xacml/>.

NOTE: any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

### **A.2 License and Notices**

Copyright © OASIS Open 2026. 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, which governs the licensure of this document, may be found at the OASIS website: [<https://www.oasis-open.org/policies-guidelines/ipr/>]

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, as provided in the OASIS IPR Policy.

This document is provided under the RF on Limited Terms IPR mode that was chosen when the project was established, as defined in the IPR Policy. For information on whether any patents have been disclosed that may be essential to implementing this document, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the project's web page (XACML IPR Policy).

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 AND ITS MEMBERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THIS DOCUMENT OR ANY PART THEREOF.

As stated in the OASIS IPR Policy, the following three paragraphs in brackets apply to OASIS Standards Final Deliverable documents (Committee Specifications, OASIS Standards, or Approved Errata).

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 Standards Final Deliverable, 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 deliverable.

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 OASIS Standards Final Deliverable 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 OASIS Standards Final Deliverable. 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 OASIS Standards Final Deliverable 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 Standards Final Deliverable, 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 document, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, its documents, while reserving the right to enforce its marks against misleading uses. Please see <https://www.oasis-open.org/policies-guidelines/trademark/> for guidance.

---

## Annex B References

(This annex forms an integral part of this Specification.)

This section contains the normative and informative references that are used in this document.

Normative references are specific (identified by date of publication and/or edition number or version number) and Informative references are either specific or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies. While any hyperlinks included in this section were valid at the time of publication, OASIS cannot guarantee their long term validity.

### B.1 Normative References

The following documents are referenced in such a way that some or all of their content constitutes requirements of this document.

[CMF]

Martin J. Dürst et al, eds., Character Model for the World Wide Web 1.0: Fundamentals, W3C Recommendation 15 February 2005, <https://www.w3.org/TR/2005/REC-charmod-20050215/>

[DS]

D. Eastlake et al., XML-Signature Syntax and Processing, <https://www.w3.org/TR/xmldsig-core/>, World Wide Web Consortium.

[ENTITIES]

*XACML v3.0 Related and Nested Entities Profile Version 1.0.* Edited by Steven Legg. 16 February 2021. OASIS Committee Specification 02. [https://docs.oasis-open.org/xacml/xacml-3.0-related-entities-v1.0-cs02.html](https://docs.oasis-open.org/xacml/xacml-3.0-related-entities/v1.0/cs02/xacml-3.0-related-entities-v1.0-cs02.html). Latest stage: <https://docs.oasis-open.org/xacml/xacml-3.0-related-entities/v1.0/xacml-3.0-related-entities-v1.0.html>.

[exc-c14n]

J. Boyer et al, eds., Exclusive XML Canonicalization, Version 1.0, W3C Recommendation 18 July 2002, <https://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/>

[Hancock]

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

[Hier]

XACML v3.0 Hierarchical Resource Profile Version 1.0. 11 March 2010. Committee Specification Draft 03. <https://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cd-03-en.html>

[IEEE754]

IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-7653-8, IEEE Product No. SH10116-TBR.

[INFOSET]

XML Information Set (Second Edition), W3C Recommendation, 4 February 2004, <https://www.w3.org/TR/xml-infoset/>

[ISO10181-3]

ISO/IEC 10181-3:1996 Information technology – Open Systems Interconnection -- Security frameworks for open systems: Access control framework.

[Kudo00]

Kudo M and Hada S, XML document security based on provisional authorization, Proceedings of the Seventh ACM Conference on Computer and Communications Security, Nov 2000, Athens, Greece, pp 87-96.

[LDAP-1]

RFC 2256, A summary of the X500(96) User Schema for use with LDAPv3, Section 5, M Wahl, December 1997, <https://www.ietf.org/rfc/rfc2256.txt>

[LDAP-2]

RFC 2798, Definition of the inetOrgPerson, M. Smith, April 2000, <https://www.ietf.org/rfc/rfc2798.txt>

[MathML]

Mathematical Markup Language (MathML), Version 2.0, W3C Recommendation, 21 October 2003, <https://www.w3.org/TR/2003/REC-MathML2-20031021/>

[Multi]

OASIS Committee Draft 03, XACML v3.0 Multiple Decision Profile Version 1.0, 11 March 2010, <https://docs.oasis-open.org/xacml/3.0/xacml-3.0-multiple-v1-spec-cd-03-en.doc>

[OCL]

OMG, Object Constraint Language (OCL) Version 2.4 (also ISO/IEC 19507), 2014. Available online: <https://www.omg.org/spec/OCL/2.4>

[Perritt93]

Perritt, H. Knowbots, Permissions Headers and Contract Law, Conference on Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment, April 1993. Available at: <https://www.cni.org/resources/historical-resources/technological-strategies-for-protecting-intellectual-property-in-the-networked-multimedia-environment/permission-headers-and-contract-law>

[RBAC]

David Ferraiolo and Richard Kuhn, Role-Based Access Controls, 15th National Computer Security Conference, 1992.

[RFC2045]

Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

[RFC2046]

Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[RFC2119]

*Key Words for Use in RFCs to Indicate Requirement Levels*, BCP 14, RFC 2119, March 1997. [Online]. Available: <https://www.rfc-editor.org/info/rfc2119>

[RFC2396]

RFC 2396, Berners-Lee T, Fielding R, Masinter L, Uniform Resource Identifiers (URI): Generic Syntax, <https://www.ietf.org/rfc/rfc2396.txt>

[RFC2732]

RFC 2732, Hinden R, Carpenter B, Masinter L, Format for Literal IPv6 Addresses in URL's, <https://www.ietf.org/rfc/rfc2732.txt>

[RFC3198]

IETF RFC 3198: Terminology for Policy-Based Management, November 2001. <https://www.ietf.org/rfc/rfc3198.txt>

[RFC4289]

Freed, N. and J. Klensin, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", BCP 13, RFC 4289, DOI 10.17487/RFC4289, December 2005, <https://www.rfc-editor.org/info/rfc4289>.

[RFC6838]

Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC8174]

*Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words*, BCP 14, RFC 8174, May 2017. [Online]. Available: <https://www.rfc-editor.org/info/rfc8174>

[UAX15]

Mark Davis, Martin Dürst, Unicode Standard Annex #15: Unicode Normalization Forms, Unicode 5.1, <https://unicode.org/reports/tr15/>

[UML]

OMG, Unified Modeling Language (UML) Version 2.5.1 (also ISO/IEC 19505), 2017. Available online: <https://www.omg.org/spec/UML/2.5.1/>

[UTR36]

Davis, Mark, Suignard, Michel, Unicode Technical Report #36: Unicode Security Considerations, <https://www.unicode.org/reports/tr36/>

[XACMLAdmin]

OASIS Committee Draft 03, XACML v3.0 Administration and Delegation Profile Version 1.0. 11 March 2010, <https://docs.oasis-open.org/xacml/3.0/xacml-3.0-administration-v1-spec-cd-03-en.doc>

[XML]

Bray, Tim, et.al. eds, Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation 26 November 2008, <https://www.w3.org/TR/2008/REC-xml-20081126/>

[XMLid]

Marsh, Jonathan, et.al. eds, xml:id Version 1.0. W3C Recommendation 9 September 2005, <https://www.w3.org/TR/2005/REC-xml-id-20050909/>

[XF]

W3C XQuery, XPath, and XSLT Functions and Operators Namespace Document (XPath and XQuery Functions and Operators 3.1) 21 March 2017, <https://www.w3.org/2005/xpath-functions/>

[XS]

XML Schema 1.1, parts 1 and 2. Available at: <https://www.w3.org/TR/xmlschema11-1/> and <https://www.w3.org/TR/xmlschema11-2/>

[XPath]

XML Path Language (XPath) 3.1, W3C Recommendation 21 March 2017, <https://www.w3.org/TR>xpath-31/>

[XSLT]

## B.2 Informative References

The following referenced documents are not required for the application of this document but may assist the reader with regard to a particular subject area.

[CM]

Character Model for the World Wide Web: String Matching W3C Working Group Note 11 August 2021, <https://www.w3.org/TR/charmod-norm/>, World Wide Web Consortium.

[Hinton94]

Hinton, H, M, Lee, E, S, The Compatibility of Policies, Proceedings 2nd ACM Conference on Computer and Communications Security, Nov 1994, Fairfax, Virginia, USA.

[NISTIR8318]

Black, P. (2020), DADS: The On-Line Dictionary of Algorithms and Data Structures, NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD, online (Accessed December 16, 2025)

[Sloman94]

Sloman, M. Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Volume 2, part 4. Plenum Press. 1994.

[XACML]

*eXtensible Access Control Markup Language (XACML) Version 3.0 Plus Errata 01*. Edited by Erik Rissanen. OASIS Standard incorporating Approved Errata. <https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html>.

---

## Annex C Data Types and Functions

(This annex forms an integral part of this Specification.)

### C.1 Introduction

This section specifies the data types and functions used in ACAL to create predicates for conditions and targets.

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. The standard functions are named and their operational semantics are described.

### C.2 Data Types

Although a syntactic representation of ACAL objects may represent most data types as strings, an ACAL 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 string representations to values that can be compared with values in their domain of discourse, such as numbers. The following data types are specified for use with ACAL and have explicit data representations:

- urn:oasis:names:tc:acal:1.0:data-type:string
- urn:oasis:names:tc:acal:1.0:data-type:boolean
- urn:oasis:names:tc:acal:1.0:data-type:integer
- urn:oasis:names:tc:acal:1.0:data-type:double
- urn:oasis:names:tc:acal:1.0:data-type:time
- urn:oasis:names:tc:acal:1.0:data-type:date
- urn:oasis:names:tc:acal:1.0:data-type:dateTime

- urn:oasis:names:tc:acal:1.0:data-type:anyURI
- urn:oasis:names:tc:acal:1.0:data-type:hexBinary
- urn:oasis:names:tc:acal:1.0:data-type:base64Binary
- urn:oasis:names:tc:acal:1.0:data-type:dayTimeDuration
- urn:oasis:names:tc:acal:1.0:data-type:yearMonthDuration
- urn:oasis:names:tc:acal:1.0:data-type:x500Name
- urn:oasis:names:tc:acal:1.0:data-type:rfc822Name
- urn:oasis:names:tc:acal:1.0:data-type:ipAddress
- urn:oasis:names:tc:acal:1.0:data-type:dnsName

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

An ACAL PDP SHALL be capable of converting string representations into various data types. For double values, implementations SHALL use the conversions described in [IEEE754].

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

- urn:oasis:names:tc:acal:1.0:data-type:x500Name,
- urn:oasis:names:tc:acal:1.0:data-type:rfc822Name,
- urn:oasis:names:tc:acal:1.0:data-type:ipAddress and
- urn:oasis:names:tc:acal:1.0:data-type:dnsName.

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

ACAL defines a data type for representing structured data:

- urn:oasis:names:tc:acal:1.0:data-type:entity

### C.2.1 X.500 Directory Name

The urn:oasis:names:tc:acal:1.0:data-type:x500Name data 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".

### C.2.2 RFC 822 Name

The urn:oasis:names:tc:acal:1.0:data-type:rfc822Name data 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".

### C.2.3 IP Address

The urn:oasis:names:tc:acal:1.0:data-type:ipAddress data 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.)

### C.2.4 DNS Name

The urn:oasis:names:tc:acal:1.0:data-type:dnsName data 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:acal:1.0:data-type:ipAddress` and `urn:oasis:names:tc:acal:1.0:data-type:dnsName` data types, the port or port range syntax SHALL be

```
portrange = portnumber | `--> 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`.]

## C.2.6 Entity

The `urn:oasis:names:tc:acal:1.0:data-type:entity` data type is used to represent an entity nested within another entity. Values of this data type are objects of the `EntityType` object type Section 7.46.

## C.3 Functions

ACAL specifies the following functions. Unless otherwise specified, if an argument of one of these functions were to evaluate to `Indeterminate`, then the function SHALL evaluates 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.

### C.3.1 Equality Predicates

The following functions are the equality functions for the various data types. Each function for a particular data type follows a specified standard convention for that data type.

**`urn:oasis:names:tc:acal:1.0:function:string-equal`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://www.w3.org/2005/xpath-functions/collation/codepoint/> by [XF].

**`urn:oasis:names:tc:acal:1.0:function:string-equal-ignore-case`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The result SHALL be `true` if and only if the two strings are equal as defined by `urn:oasis:names:tc:acal:1.0:function:string-equal` after they have both been converted to lower case with `urn:oasis:names:tc:acal:1.0:function:string-normalize-to-lowercase`.

**`urn:oasis:names:tc:acal:1.0:function:boolean-equal`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:boolean` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The function SHALL return `true` if and only if the arguments are equal. Otherwise, it SHALL return `false`.

**`urn:oasis:names:tc:acal:1.0:function:integer-equal`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:integer` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The function SHALL return `true` if and only if the two arguments represent the same number.

**`urn:oasis:names:tc:acal:1.0:function:double-equal`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:double` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. It SHALL perform its evaluation on doubles according to IEEE 754 [IEEE754].

**`urn:oasis:names:tc:acal:1.0:function:date-equal`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:date` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. It SHALL perform its evaluation according to the `op:date-equal` function [XF] Section 8.4.4.

**`urn:oasis:names:tc:acal:1.0:function:time-equal`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:time`

and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. It SHALL perform its evaluation according to the `op:time-equal` function [XF] Section 8.4.7.

**`urn:oasis:names:tc:acal:1.0:function:dateTime-equal`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:dateTime` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. It SHALL perform its evaluation according to the `op:dateTime-equal` function [XF] Section 8.4.1.

**`urn:oasis:names:tc:acal:1.0:function:dayTimeDuration-equal`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:dayTimeDuration` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. This function shall perform its evaluation according to the `op:duration-equal` function [XF] Section 8.2.5. Note that the lexical representation of each argument MUST be converted to a value expressed in fractional seconds [XS].

**`urn:oasis:names:tc:acal:1.0:function:yearMonthDuration-equal`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:yearMonthDuration` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. This function shall perform its evaluation according to the `op:duration-equal` function [XF] Section 8.2.5. Note that the lexical representation of each argument MUST be converted to a value expressed in months [XS].

**`urn:oasis:names:tc:acal:1.0:function:anyURI-equal`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:anyURI` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The function SHALL convert the arguments to strings with `urn:oasis:names:tc:acal:1.0: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:acal:1.0:function:x500Name-equal`**

This function SHALL take two arguments of `urn:oasis:names:tc:acal:1.0:data-type:x500Name` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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:acal:1.0:function:rfc822Name-equal`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:rfc822Name` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 `urn:oasis:names:tc:acal:1.0:function:string-equal` to the normalized arguments.

**`urn:oasis:names:tc:acal:1.0:function:hexBinary-equal`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:hexBinary` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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:acal: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.

**`urn:oasis:names:tc:acal:1.0:function:base64Binary-equal`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:base64Binary` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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:acal: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.

### C.3.2 Arithmetic Functions

All of the following functions SHALL take two arguments of the specified data type, integer, or double, and SHALL return a value 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`.

`urn:oasis:names:tc:acal:1.0:function:integer-add`

This function MUST accept two or more arguments.

`urn:oasis:names:tc:acal:1.0:function:double-add`

This function MUST accept two or more arguments.

`urn:oasis:names:tc:acal:1.0:function:integer-subtract`

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

`urn:oasis:names:tc:acal:1.0:function:double-subtract`

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

`urn:oasis:names:tc:acal:1.0:function:integer-multiply`

This function MUST accept two or more arguments.

`urn:oasis:names:tc:acal:1.0:function:double-multiply`

This function MUST accept two or more arguments.

`urn:oasis:names:tc:acal:1.0:function:integer-divide`

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

`urn:oasis:names:tc:acal:1.0:function:double-divide`

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

`urn:oasis:names:tc:acal:1.0:function:integer-mod`

The result is the 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 `urn:oasis:names:tc:acal:1.0:data-type:double` and return a value of the data type `urn:oasis:names:tc:acal:1.0:data-type:double`.

- `urn:oasis:names:tc:acal:1.0:function:integer-abs`
- `urn:oasis:names:tc:acal:1.0:function:double-abs`
- `urn:oasis:names:tc:acal:1.0:function:round`
- `urn:oasis:names:tc:acal:1.0:function:floor`

### C.3.3 String Conversion Functions

The following functions convert between values of the data type `urn:oasis:names:tc:acal:1.0:data-type:string` data type.

`urn:oasis:names:tc:acal:1.0:function:string-normalize-space`

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type: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:acal:1.0:function:string-normalize-to-lower-case`

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type: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.

### C.3.4 Numeric Data Type Conversion Functions

The following functions convert values between the `urn:oasis:names:tc:acal:1.0:data-type:integer` and `urn:oasis:names:tc:acal:1.0:data-type:double` data types.

#### `urn:oasis:names:tc:acal:1.0:function:double-to-integer`

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:double` and SHALL truncate its numeric value to a whole number and return a value of data type `urn:oasis:names:tc:acal:1.0:data-type:integer`.

#### `urn:oasis:names:tc:acal:1.0:function:integer-to-double`

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:integer` and SHALL promote its value to a value of data type `urn:oasis:names:tc:acal:1.0:data-type:double` with the same numeric value. If the integer argument is outside the range that can be represented by a double, the result SHALL be `Indeterminate`, with status code `urn:oasis:names:tc:acal:1.0:status:processing-error`.

### C.3.5 Logical Functions

This section contains the specification for logical functions that operate on arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:boolean`.

#### `urn:oasis:names:tc:acal:1.0:function:or`

This function SHALL take zero or more arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:boolean`.

This function returns `true` if at least one of its arguments evaluates to `true`; otherwise, the function returns `Indeterminate` if any argument evaluates to `Indeterminate`; otherwise, the function returns `false` (which applies in the case of zero arguments). The order of evaluation SHALL be from the first argument to the last and SHALL stop immediately if any argument evaluates to `true`, leaving the rest of the arguments unevaluated.

#### `urn:oasis:names:tc:acal:1.0:function:and`

This function SHALL take zero or more arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:boolean`.

This function returns `false` if at least one of its arguments evaluates to `false`; otherwise, the function returns `Indeterminate` if any argument evaluates to `Indeterminate`; otherwise, the function returns `true` (which applies in the case of zero arguments). The order of evaluation SHALL be from the first argument to the last and SHALL stop immediately if any argument evaluates to `false`, leaving the rest of the arguments unevaluated.

#### `urn:oasis:names:tc:acal:1.0:function:n-of`

The first argument to this function SHALL be of data type `urn:oasis:names:tc:acal:1.0:data-type:integer`.

The remaining arguments (the Boolean arguments) SHALL be of data type `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The first argument specifies the minimum number of the Boolean arguments that must evaluate to `true` for the expression to be considered `true`.

Let N be the value of the first argument and let M be the number of Boolean arguments. This function returns `Indeterminate` if N is `Indeterminate`; otherwise, the function returns `true` if N is less than or equal to 0; otherwise, the function returns `false` if N is greater than M; otherwise, the function returns `true` if at least N of the Boolean arguments evaluate to `true`; otherwise, the function returns `false` if at least (M + 1 - N) of the Boolean arguments evaluate to `false`; otherwise, the function returns `Indeterminate`.

The order of evaluation SHALL be: evaluate the first argument and if it is not `Indeterminate`, then evaluate each Boolean argument in order. The evaluation SHALL stop immediately leaving the rest of the Boolean arguments unevaluated if the requirements to return `true` or `false` have been satisfied or if there are insufficient remaining Boolean arguments to satisfy the requirements to return either `true` or `false`.

Note that this function differs from the similarly-named function in XACML [XACML], which returned `Indeterminate` if N was less than zero or greater than M.

#### `urn:oasis:names:tc:acal:1.0:function:not`

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:boolean`.

This function returns `true` if the argument evaluates to `false`; otherwise, the function returns `false` if the argument evaluates to `true`; otherwise, the function returns `Indeterminate`.

#### `urn:oasis:names:tc:acal:1.0:function:ternary-if`

This function SHALL take three arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:boolean`.

If the first argument evaluates to `true`, then the function returns the result of evaluating the second

argument and the third argument need not be evaluated; otherwise, if the first argument evaluates to `false`, then the function returns the result of evaluating the third argument and the second argument need not be evaluated; otherwise, the function returns `Indeterminate`.

### C.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:acal:1.0:function:integer-greater-than`
- `urn:oasis:names:tc:acal:1.0:function:integer-greater-than-or-equal`
- `urn:oasis:names:tc:acal:1.0:function:integer-less-than`
- `urn:oasis:names:tc:acal:1.0:function:integer-less-than-or-equal`
- `urn:oasis:names:tc:acal:1.0:function:double-greater-than`
- `urn:oasis:names:tc:acal:1.0:function:double-greater-than-or-equal`
- `urn:oasis:names:tc:acal:1.0:function:double-less-than`
- `urn:oasis:names:tc:acal:1.0:function:double-less-than-or-equal`

### C.3.7 Date and Time Arithmetic Functions

These functions perform arithmetic operations with date and time.

#### `urn:oasis:names:tc:acal:1.0:function:dateTime-add-dayTimeDuration`

This function SHALL take two arguments, the first SHALL be of data type `urn:oasis:names:tc:acal:1.0:data-type` and the second SHALL be of data type `urn:oasis:names:tc:acal:1.0:data-type:dayTimeDuration`.

It SHALL return a result of `urn:oasis:names:tc:acal:1.0:data-type: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:acal:1.0:function:dateTime-add-yearMonthDuration`

This function SHALL take two arguments, the first SHALL be a `urn:oasis:names:tc:acal:1.0:data-type:dateTime` and the second SHALL be a `urn:oasis:names:tc:acal:1.0:data-type:yearMonthDuration`. It SHALL return a result of `urn:oasis:names:tc:acal:1.0:data-type: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:acal:1.0:function:dateTime-subtract-dayTimeDuration`

This function SHALL take two arguments, the first SHALL be a `urn:oasis:names:tc:acal:1.0:data-type:dateTime` and the second SHALL be a `urn:oasis:names:tc:acal:1.0:data-type:dayTimeDuration`. It SHALL return a result of `urn:oasis:names:tc:acal:1.0:data-type:dateTime`. If the second argument is a positive duration, then this function SHALL return the value by adding the corresponding negative duration, as per the specification [XS] Appendix E. If the second argument is a negative duration, then the result SHALL be as if the function `urn:oasis:names:tc:acal:1.0:function:dateTime-add-dayTimeDuration` had been applied to the corresponding positive duration.

#### `urn:oasis:names:tc:acal:1.0:function:dateTime-subtract-yearMonthDuration`

This function SHALL take two arguments, the first SHALL be a `urn:oasis:names:tc:acal:1.0:data-type:dateTime` and the second SHALL be a `urn:oasis:names:tc:acal:1.0:data-type:yearMonthDuration`. It SHALL return a result of `urn:oasis:names:tc:acal:1.0:data-type:dateTime`. If the second argument is a positive duration, then this function SHALL return the value by adding the corresponding negative duration, as per the specification [XS] Appendix E. If the second argument is a negative duration, then the result SHALL be as if the function `urn:oasis:names:tc:acal:1.0:function:dateTime-add-yearMonthDuration` had been applied to the corresponding positive duration.

#### `urn:oasis:names:tc:acal:1.0:function:date-add-yearMonthDuration`

This function SHALL take two arguments, the first SHALL be a `urn:oasis:names:tc:acal:1.0:data-type:date` and the second SHALL be a `urn:oasis:names:tc:acal:1.0:data-type:yearMonthDuration`. It SHALL return a result of `urn:oasis:names:tc:acal:1.0:data-type:date`. This function SHALL return the value by adding the second argument to the first argument according to the specification of adding duration to date [XS] Appendix E.

#### **urn:oasis:names:tc:acal:1.0:function:date-subtract-yearMonthDuration**

This function SHALL take two arguments, the first SHALL be a `urn:oasis:names:tc:acal:1.0:data-type:date` and the second SHALL be a `urn:oasis:names:tc:acal:1.0:data-type:yearMonthDuration`. It SHALL return a result of `urn:oasis:names:tc:acal:1.0:data-type:date`. If the second argument is a positive duration, then this function SHALL return the value by adding the corresponding negative duration, as per the specification [XS] Appendix E. If the second argument is a negative duration, then the result SHALL be as if the function `urn:oasis:names:tc:acal:1.0:function:date-add-yearMonthDuration` had been applied to the corresponding positive duration.

### **C.3.8 Non-numeric Comparison Functions**

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

#### **urn:oasis:names:tc:acal:1.0:function:string-greater-than**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://www.w3.org/2005/xpath-functions/collation/codepoint/> by [XF].

#### **urn:oasis:names:tc:acal:1.0:function:string-greater-than-or-equal**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://www.w3.org/2005/xpath-functions/collation/codepoint/> by [XF].

#### **urn:oasis:names:tc:acal:1.0:function:string-less-than**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. It SHALL return `true` if and only if the first argument is lexicographically strictly less than the second argument. Otherwise, it SHALL return `false`. The comparison SHALL use Unicode codepoint collation, as defined for the identifier <https://www.w3.org/2005/xpath-functions/collation/codepoint/> by [XF].

#### **urn:oasis:names:tc:acal:1.0:function:string-less-than-or-equal**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. It SHALL return `true` if and only if the first argument is lexicographically less than or equal to the second argument. Otherwise, it SHALL return `false`. The comparison SHALL use Unicode codepoint collation, as defined for the identifier <https://www.w3.org/2005/xpath-functions/collation/codepoint/> by [XF].

#### **urn:oasis:names:tc:acal:1.0:function:time-greater-than**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:time` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://www.w3.org/2001/XMLSchema#time> by [XS] part 2, 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.

#### **urn:oasis:names:tc:acal:1.0:function:time-greater-than-or-equal**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:time` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://www.w3.org/2001/XMLSchema#time> by [XS] part 2, 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.

#### **urn:oasis:names:tc:acal:1.0:function:time-less-than**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:time` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://www.w3.org/2001/XMLSchema#time> by [XS] part 2, 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.

**urn:oasis:names:tc:acal:1.0:function:time-less-than-or-equal**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:time` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://www.w3.org/2001/XMLSchema#time> by [XS] part 2, 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.

**urn:oasis:names:tc:acal:1.0:function:time-in-range**

This function SHALL take three arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:time` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. It SHALL return `true` if the first argument falls in the range defined inclusively by the second and third arguments. Otherwise, it SHALL return `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:acal:1.0:function:dateTime-greater-than**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:dateTime` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://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:acal:1.0:function:dateTime-greater-than-or-equal**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:dateTime` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://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:acal:1.0:function:dateTime-less-than**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:dateTime` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 `urn:oasis:names:tc:acal:1.0:data-type: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:acal:1.0:function:dateTime-less-than-or-equal**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:dateTime` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://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:acal:1.0:function:date-greater-than**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:date` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://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:acal:1.0:function:date-greater-than-or-equal**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:date` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://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:acal:1.0:function:date-less-than**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:date` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://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:acal:1.0:function:date-less-than-or-equal**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:date` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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 <https://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].

### C.3.9 String Functions

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

**urn:oasis:names:tc:acal:1.0:function:string-concatenate**

This function SHALL take two or more arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return a `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the concatenation, in order, of the arguments.

**urn:oasis:names:tc:acal:1.0:function:boolean-from-string**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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:acal:1.0:status:syntax-error`.

**urn:oasis:names:tc:acal:1.0:function:string-from-boolean**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:boolean`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the Boolean converted to a string in the canonical form specified in [XS].

**urn:oasis:names:tc:acal:1.0:function:integer-from-string**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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:acal:1.0:status:syntax-error`.

**urn:oasis:names:tc:acal:1.0:function:string-from-integer**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:integer`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the integer converted to a string in the canonical form specified in [XS].

**urn:oasis:names:tc:acal:1.0:function:double-from-string**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type: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:acal:1.0:status:syntax-error`.

**urn:oasis:names:tc:acal:1.0:function:string-from-double**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:double`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the double converted to a string in the canonical form specified in [XS].

**urn:oasis:names:tc:acal:1.0:function:time-from-string**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:time`. The result SHALL be the string converted to a time. If the argument is not a valid lexical representation of a time, then the result SHALL be Indeterminate with status code `urn:oasis:names:tc:acal:1.0:status:syntax-error`.

**urn:oasis:names:tc:acal:1.0:function:string-from-time**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:time`,

and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the time converted to a string in the canonical form specified in [XS].

**`urn:oasis:names:tc:acal:1.0:function:date-from-string`**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:date`. The result SHALL be the string converted to a date. If the argument is not a valid lexical representation of a date, then the result SHALL be Indeterminate with status code `urn:oasis:names:tc:acal:1.0:status:syntax-error`.

**`urn:oasis:names:tc:acal:1.0:function:string-from-date`**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:date`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the date converted to a string in the canonical form specified in [XS].

**`urn:oasis:names:tc:acal:1.0:function:dateTime-from-string`**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:dateTime`. The result SHALL be the string converted to a dateTime. If the argument is not a valid lexical representation of a dateTime, then the result SHALL be Indeterminate with status code `urn:oasis:names:tc:acal:1.0:status:syntax-error`.

**`urn:oasis:names:tc:acal:1.0:function:string-from-datetime`**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:dateTime`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the dateTime converted to a string in the canonical form specified in [XS].

**`urn:oasis:names:tc:acal:1.0:function:anyURI-from-string`**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string`, and SHALL return a `urn:oasis:names:tc:acal:1.0:data-type:anyURI`. The result SHALL be the URI constructed by converting the argument to a URI. If the argument is not a valid lexical representation of a URI, then the result SHALL be Indeterminate with status code `urn:oasis:names:tc:acal:1.0:status:syntax-error`.

**`urn:oasis:names:tc:acal:1.0:function:string-from-anyURI`**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:anyURI`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the URI converted to a string in the form it was originally represented.

**`urn:oasis:names:tc:acal:1.0:function:dayTimeDuration-from-string`**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:dayTimeDuration`. The result SHALL be the string converted to a dayTimeDuration. If the argument is not a valid lexical representation of a dayTimeDuration, then the result SHALL be Indeterminate with status code `urn:oasis:names:tc:acal:1.0:status:syntax-error`.

**`urn:oasis:names:tc:acal:1.0:function:string-from-dayTimeDuration`**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:dayTimeDuration`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the dayTimeDuration converted to a string in the canonical form specified in [XS].

**`urn:oasis:names:tc:acal:1.0:function:yearMonthDuration-from-string`**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:yearMonthDuration`. 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:acal:1.0:status:syntax-error`.

**`urn:oasis:names:tc:acal:1.0:function:string-from-yearMonthDuration`**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:yearMonthDuration`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the yearMonthDuration converted to a string in the canonical form specified in [XS].

**`urn:oasis:names:tc:acal:1.0:function:x500Name-from-string`**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string`, and SHALL return an `urn:oasis:names:tc:acal: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:acal:1.0:status:syntax-error`.

**urn:oasis:names:tc:acal:1.0:function:string-from-x500Name**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:x500Name`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the `x500Name` converted to a string in the form it was originally represented.

**urn:oasis:names:tc:acal:1.0:function:rfc822Name-from-string**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:rfc822Name`. 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:acal:1.0:status:syntax-error`.

**urn:oasis:names:tc:acal:1.0:function:string-from-rfc822Name**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:rfc822Name`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the `rfc822Name` converted to a string in the form it was originally represented.

**urn:oasis:names:tc:acal:1.0:function:ipAddress-from-string**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string`, and SHALL return an `urn:oasis:names:tc:acal:1.0: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:acal:1.0:status:syntax-error`.

**urn:oasis:names:tc:acal:1.0:function:string-from-ipAddress**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:ipAddress`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the `ipAddress` converted to a string in the form it was originally represented.

**urn:oasis:names:tc:acal:1.0:function:dnsName-from-string**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string`, and SHALL return an `urn:oasis:names:tc:acal:1.0: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:acal:1.0:status:syntax-error`.

**urn:oasis:names:tc:acal:1.0:function:string-from-dnsName**

This function SHALL take one argument of data type `urn:oasis:names:tc:acal:1.0:data-type:dnsName`, and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the `dnsName` converted to a string in the form it was originally represented.

**urn:oasis:names:tc:acal:1.0:function:string-starts-with**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return a `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The result SHALL be true if the first string begins with the second string, and false otherwise. Equality testing SHALL be done as defined for `urn:oasis:names:tc:acal:1.0:function:string-equal`.

Note that the arguments to this function are swapped compared to the similarly-named function in XACML [XACML].

**urn:oasis:names:tc:acal:1.0:function:anyURI-starts-with**

This function SHALL take a first argument of data type `urn:oasis:names:tc:acal:1.0:data-type:anyURI` and a second argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return a `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The result SHALL be true if the `URI` converted to a string with `urn:oasis:names:tc:acal:1.0:function:string-from-anyURI` begins with the string, and false otherwise. Equality testing SHALL be done as defined for `urn:oasis:names:tc:acal:1.0:function:string-equal`.

Note that the arguments to this function are swapped compared to the similarly-named function in XACML [XACML].

**urn:oasis:names:tc:acal:1.0:function:string-ends-with**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return a `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The result SHALL be true if the first string ends with the second string, and false otherwise. Equality testing SHALL be done as defined for `urn:oasis:names:tc:acal:1.0:function:string-equal`.

Note that the arguments to this function are swapped compared to the similarly-named function in XACML [XACML].

**`urn:oasis:names:tc:acal:1.0:function:anyURI-ends-with`**

This function SHALL take a first argument of data type `urn:oasis:names:tc:acal:1.0:data-type:anyURI` and a second argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return a `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The result SHALL be true if the URI converted to a string with `urn:oasis:names:tc:acal:1.0:function:string-from-anyURI` ends with the string, and false otherwise. Equality testing SHALL be done as defined for `urn:oasis:names:tc:acal:1.0:function:string-equal`.

Note that the arguments to this function are swapped compared to the similarly-named function in XACML [XACML].

**`urn:oasis:names:tc:acal:1.0:function:string-contains`**

This function SHALL take two arguments of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return a `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The result SHALL be true if the first string contains the second string, and false otherwise. Equality testing SHALL be done as defined for `urn:oasis:names:tc:acal:1.0:function:string-equal`.

Note that the arguments to this function are swapped compared to the similarly-named function in XACML [XACML].

**`urn:oasis:names:tc:acal:1.0:function:anyURI-contains`**

This function SHALL take a first argument of data type `urn:oasis:names:tc:acal:1.0:data-type:anyURI` and a second argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return a `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The result SHALL be true if the URI converted to a string with `urn:oasis:names:tc:acal:1.0:function:string-from-anyURI` contains the string, and false otherwise. Equality testing SHALL be done as defined for `urn:oasis:names:tc:acal:1.0:function:string-equal`.

Note that the arguments to this function are swapped compared to the similarly-named function in XACML [XACML].

**`urn:oasis:names:tc:acal:1.0:function:string-substring`**

This function SHALL take a first argument of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and a second and a third argument of type `urn:oasis:names:tc:acal:1.0:data-type:integer` and SHALL return a `urn:oasis:names:tc:acal:1.0:data-type: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:acal:1.0:status:processing-error`.

**`urn:oasis:names:tc:acal:1.0:function:anyURI-substring`**

This function SHALL take a first argument of data type `urn:oasis:names:tc:acal:1.0:data-type:anyURI` and a second and a third argument of type `urn:oasis:names:tc:acal:1.0:data-type:integer` and SHALL return a `urn:oasis:names:tc:acal:1.0:data-type:string`. The result SHALL be the substring of the first argument converted to a string with `urn:oasis:names:tc:acal:1.0: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:acal: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 code of `urn:oasis:names:tc:acal:1.0:status:processing-error`.

### C.3.10 Bag Functions

These functions operate on a bag of `type` values, where `type` is one of the ACAL 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` if not satisfied.

**`urn:oasis:names:tc:acal:1.0: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:acal:1.0:function:type-bag-size**

This function SHALL take a bag of type values as an argument and SHALL return an **urn:oasis:names:tc:acal:1.0:data-type:boolean** indicating the number of values in the bag.

**urn:oasis:names:tc:acal:1.0: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 **urn:oasis:names:tc:acal:1.0:data-type:boolean**. The function SHALL evaluate to **true** if and only if the first argument matches by **urn:oasis:names:tc:acal:1.0:function:type-equals** any value in the bag. Otherwise, it SHALL return **false**.

**urn:oasis:names:tc:acal:1.0: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.

**urn:oasis:names:tc:acal:1.0:function:is-empty**

This function SHALL take a single bag argument of any data type and SHALL return an **urn:oasis:names:tc:acal:1.0:data-type:boolean**. The function SHALL evaluate to **true** if and only if the bag argument has no values.

**urn:oasis:names:tc:acal:1.0:function:is-not-empty**

This function SHALL take a single bag argument of any data type and SHALL return an **urn:oasis:names:tc:acal:1.0:data-type:boolean**. The function SHALL evaluate to **true** if and only if the bag argument has at least one value.

### C.3.11 Set Functions

These functions operate on bags mimicking sets by eliminating duplicate values from the bag.

**urn:oasis:names:tc:acal:1.0: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 values that are common between the two bags, which is determined by **urn:oasis:names:tc:acal:1.0:function:type-equal**. Duplicates, as determined by **urn:oasis:names:tc:acal:1.0:function:type-equal**, SHALL NOT exist in the result.

**urn:oasis:names:tc:acal:1.0: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 **urn:oasis:names:tc:acal:1.0:data-type:boolean**. The function SHALL evaluate to **true** if and only if at least one value of the first argument is contained in the second argument as determined by **urn:oasis:names:tc:acal:1.0:function:type-is-in**.

**urn:oasis:names:tc:acal:1.0: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 the values of all the argument bags. Duplicates, as determined by **urn:oasis:names:tc:acal:1.0:function:type-equal**, SHALL NOT exist in the result.

**urn:oasis:names:tc:acal:1.0:function:type-subset**

This function SHALL take two arguments that are both a bag of type values. It SHALL return a **urn:oasis:names:tc:acal:1.0:data-type: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:acal:1.0:function:type-equal**, before the subset calculation.

**urn:oasis:names:tc:acal:1.0:function:type-set-equals**

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

### C.3.12 Higher-order Bag Functions

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

Examples in this section are presented in both the XML and JSON representations and assume the use of the standardized short identifier set `urn:oasis:names:tc:acal:1.0:core:identifiers`.

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

This function applies a Boolean function between single values and a bag of values, and SHALL return `true` if and only if the function is `true` for at least one value in the bag.

This function SHALL take  $n+1$  arguments, where  $n$  is one or greater. The first argument SHALL be a `FunctionType` object that names a Boolean function that takes  $n$  single value arguments. Under the remaining  $n$  arguments,  $n-1$  arguments SHALL be single values and one SHALL be a bag of values. The expression SHALL be evaluated as if the function named in the `FunctionType` object were applied to the  $n-1$  single value arguments and each value in the bag argument and the results are combined with `urn:oasis:names:tc:acal:1.0:function:or`.

For example, the following expression evaluates to `true`:

```
<Apply FunctionId="any-of">
  <Function Id="string-equal"/>
  <Value>Paul</Value>
  <Apply FunctionId="string-bag">
    <Value>John</Value>
    <Value>Paul</Value>
    <Value>George</Value>
    <Value>Ringo</Value>
  </Apply>
</Apply>

"Apply": {
  "FunctionId": "any-of",
  "Expression": [
    { "Function": { "Id": "string-equal" } },
    { "Value": "Paul" },
    { "Apply": {
      "FunctionId": "string-bag",
      "Expression": [
        { "Value": "John" },
        { "Value": "Paul" },
        { "Value": "George" },
        { "Value": "Ringo" }
      ]
    }}
  ]
}
```

This expression is `true` because the first argument is equal to at least one of the values in the bag, according to the function.

- `urn:oasis:names:tc:acal:1.0:function:all-of`

This function applies a Boolean function between single values and a bag of values, and returns `true` if and only if the function is `true` for every value in the bag.

This function SHALL take  $n+1$  arguments, where  $n$  is one or greater. The first argument SHALL be a `FunctionType` object that names a Boolean function that takes  $n$  single value arguments. Under the remaining  $n$  arguments,  $n-1$  arguments SHALL be single values and one SHALL be a bag of values. The expression SHALL be evaluated as if the function named in the `FunctionType` object were applied to the  $n-1$  single-value arguments and each value of the bag argument and the results are combined with `urn:oasis:names:tc:acal:1.0:function:and`.

For example, the following expression evaluates to `true`:

```
<Apply FunctionId="all-of">
  <Function Id="integer-greater-than"/>
  <Value>10</Value>
  <Apply FunctionId="integer-bag">
    <Value>9</Value>
    <Value>3</Value>
    <Value>4</Value>
    <Value>2</Value>
  </Apply>
</Apply>

"Apply": {
  "FunctionId": "all-of",
```

```

"Expression": [
  { "Function":{ "Id":"integer-greater-than" } },
  { "Value": 10 },
  { "Apply":{
    "FunctionId":"integer-bag",
    "Expression": [
      { "Value": 9 },
      { "Value": 3 },
      { "Value": 4 },
      { "Value": 2 }
    ]
  }}
]
}

```

The expression is **true** because the first argument (10) is greater than all of the values in the bag (9, 3, 4 and 2).

- **urn:oasis:names:tc:acal:1.0:function:any-of-any**

This function applies a Boolean function on each tuple from the cross product on all bag arguments, and returns **true** if and only if the function 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 **FunctionType** object that names a Boolean function that takes n single-value arguments. The remaining arguments are either single values or bags of values. The expression SHALL be evaluated as if the function named in the **FunctionType** object was applied between every tuple of the cross product on all bags and single values, and the results were combined using **urn:oasis:names:tc:acal:1.0:function:or**. The semantics are that the result of the expression SHALL be **true** if and only if the applied function is **true** for at least one function call on the tuples from the bags and single values.

For example, the following expression evaluates to **true**:

```

<Apply FunctionId="any-of-any">
  <Function Id="string-equal"/>
  <Apply FunctionId="string-bag">
    <Value>Ringo</Value>
    <Value>Mary</Value>
  </Apply>
  <Apply FunctionId="string-bag">
    <Value>John</Value>
    <Value>Paul</Value>
    <Value>George</Value>
    <Value>Ringo</Value>
  </Apply>
</Apply>

"Apply": {
  "FunctionId": "any-of-any",
  "Expression": [
    { "Function":{ "Id":"string-equal" } },
    { "Apply":{
      "FunctionId": "string-bag",
      "Expression": [
        { "Value": "Ringo" },
        { "Value": "Mary" }
      ]
    }},
    { "Apply":{
      "FunctionId": "string-bag",
      "Expression": [
        { "Value": "John" },
        { "Value": "Paul" },
        { "Value": "George" },
        { "Value": "Ringo" }
      ]
    }}
  ]
}

```

The expression is **true** because at least one of the values of the first bag, namely **Ringo**, is equal to at least one of the values of the second bag.

- **urn:oasis:names:tc:acal:1.0:function:all-of-any**

This function applies a Boolean function between the values of two bags. The expression SHALL be **true** if and only if the supplied function is **true** between each value of the first bag and any value of the second bag.

This function SHALL take three arguments. The first argument SHALL be a `FunctionType` object that names a Boolean function that takes two single-value arguments. The second argument SHALL be a bag of values. The third argument SHALL be a bag of values. The expression SHALL be evaluated as if the `urn:oasis:names:tc:acal: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 function, and the results were then combined using `urn:oasis:names:tc:acal:1.0:function:and`.

For example, the following expression evaluates to **true**:

```
<Apply FunctionId="all-of-any">
  <Function Id="integer-greater-than"/>
  <Apply FunctionId="integer-bag">
    <Value>10</Value>
    <Value>20</Value>
  </Apply>
  <Apply FunctionId="integer-bag">
    <Value>1</Value>
    <Value>3</Value>
    <Value>5</Value>
    <Value>19</Value>
  </Apply>
</Apply>

"Apply": {
  "FunctionId": "all-of-any",
  "Expression": [
    { "Function": { "Id": "integer-greater-than" } },
    { "Apply": {
      "FunctionId": "integer-bag",
      "Expression": [
        { "Value": 10 },
        { "Value": 20 }
      ]
    }},
    { "Apply": {
      "FunctionId": "integer-bag",
      "Expression": [
        { "Value": 1 },
        { "Value": 3 },
        { "Value": 5 },
        { "Value": 19 }
      ]
    }}
  ]
}
```

The expression is **true** because each of the values of the first bag is greater than at least one of the values of the second bag.

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

This function applies a Boolean function between the values of two bags. The expression SHALL be **true** if and only if the supplied function is **true** between each value of the second bag and any value of the first bag.

This function SHALL take three arguments. The first argument SHALL be a `FunctionType` object that names a Boolean function that takes two single-value arguments. The second argument SHALL be a bag of values. The third argument SHALL be a bag of values. The expression SHALL be evaluated as if the `urn:oasis:names:tc:acal:1.0:function:any-of` function had been applied to each value of the second bag and the whole of the first bag using the supplied function, and the results were then combined using `urn:oasis:names:tc:acal:1.0:function:and`.

For example, the following expression evaluates to **true**:

```
<Apply FunctionId="any-of-all">
  <Function Id="integer-greater-than"/>
  <Apply FunctionId="integer-bag">
    <Value>3</Value>
    <Value>5</Value>
  </Apply>
```

```

<Apply FunctionId="integer-bag">
  <Value>1</Value>
  <Value>2</Value>
  <Value>3</Value>
  <Value>4</Value>
</Apply>
</Apply>

"Apply":{

  "FunctionId": "any-of-all",
  "Expression": [
    { "Function": { "Id": "integer-greater-than" } },
    { "Apply": {
        "FunctionId": "integer-bag",
        "Expression": [
          { "Value": 3 },
          { "Value": 5 }
        ]
      }
    },
    { "Apply": {
        "FunctionId": "integer-bag",
        "Expression": [
          { "Value": 1 },
          { "Value": 2 },
          { "Value": 3 },
          { "Value": 4 }
        ]
      }
    }
  ]
}

```

The 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:acal:1.0:function:all-of-all**

This function applies a Boolean function between the values of two bags. The expression SHALL be **true** if and only if the supplied function is **true** between each and every value of the first bag collectively against all the values of the second bag.

This function SHALL take three arguments. The first argument SHALL be an **FunctionType** object that names a Boolean function that takes two single-value arguments. The second argument SHALL be a bag of values. The third argument SHALL be a bag of values. The expression is evaluated as if the function named in the **FunctionType** object were applied between every value in the second argument and every value in the third argument and the results were combined using **urn:oasis:names:tc:acal:1.0:function:and**. The semantics are that the result of the expression is **true** if and only if the applied function is **true** for all the values of the first bag compared to all the values of the second bag.

For example, the following expression evaluates to **true**:

```

<Apply FunctionId="all-of-all">
  <Function Id="integer-greater-than"/>
  <Apply FunctionId="integer-bag">
    <Value>6</Value>
    <Value>5</Value>
  </Apply>
  <Apply FunctionId="integer-bag">
    <Value>1</Value>
    <Value>2</Value>
    <Value>3</Value>
    <Value>4</Value>
  </Apply>
</Apply>

"Apply":{

  "FunctionId": "all-of-all",
  "Expression": [
    { "Function": { "Id": "integer-greater-than" } },
    { "Apply": {
        "FunctionId": "integer-bag",
        "Expression": [
          { "Value": 6 },
          { "Value": 5 }
        ]
      }
    },
    { "Apply": {
      }
    }
  ]
}

```

```

    "FunctionId": "integer-bag",
    "Expression": [
        { "Value": 1 },
        { "Value": 2 },
        { "Value": 3 },
        { "Value": 4 }
    ]
}
]
}

```

The expression is `true` because all values of the first bag, i.e., 5 and 6, are each greater than all of the integer values 1, 2, 3, 4 of the second bag.

- `urn:oasis:names:tc:acal:1.0: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 `FunctionType` object naming a function that takes  $n$  single-value arguments and returns a single value. Under the remaining  $n$  arguments,  $n-1$  arguments SHALL be single values and one SHALL be a bag of values. The expression SHALL be evaluated as if the function named in the `FunctionType` object were applied to the  $n-1$  single-value arguments and each value in the bag argument and resulting in a bag of the converted values. The result SHALL be a bag of the data type that is returned by the function named in the `FunctionType` object.

For example, the following expression,

```

<Apply FunctionId="map">
    <Function Id="string-normalize-to-lower-case"/>
    <Apply FunctionId="string-bag">
        <Value>Hello</Value>
        <Value>World!</Value>
    </Apply>
</Apply>

"Apply": {
    "FunctionId": "map",
    "Expression": [
        { "Function": { "Id": "string-normalize-to-lower-case" } },
        { "Apply": {
            "FunctionId": "string-bag",
            "Expression": [
                { "Value": "Hello" },
                { "Value": "World!" }
            ]
        }}
    ]
}

```

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

### C.3.13 Regular-Expression-Based Functions

These functions operate on various types using regular expressions and evaluate to `urn:oasis:names:tc:acal:1.0:data-type:boolean`.

Note that the arguments to these functions are swapped compared to the similarly-named functions in XACML [XACML].

- `urn:oasis:names:tc:acal:1.0:function:string-regexp-match`

This function decides a regular expression match. It SHALL take two arguments of `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The first argument SHALL be a general string and the second argument SHALL be a regular expression. The function specification SHALL be that of the `xf:matches` function [XF] Section 5.6.3.

- `urn:oasis:names:tc:acal:1.0:function:anyURI-regexp-match`

This function decides a regular expression match. It SHALL take two arguments; the first is of type `urn:oasis:names:tc:acal:1.0:data-type:anyURI` and the second is of type `urn:oasis:names:tc:acal:1.0:data-type:string`. It SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The first argument SHALL be a URI and the second argument SHALL be a regular expression. The function SHALL convert the first argument to type `urn:oasis:names:tc:acal:1.0:data-type:string` with `urn:oasis:names:tc:acal:1.0:function:string-from-anyURI`, then apply `urn:oasis:names:tc:acal:1.0:function:string-regexp-match`.

- **urn:oasis:names:tc:acal:1.0:function:ipAddress-regexp-match**

This function decides a regular expression match. It SHALL take two arguments; the first is of type `urn:oasis:names:tc:acal:1.0:data-type:ipAddress` and the second is of type `urn:oasis:names:tc:acal:1.0:data-type:string`. It SHALL return an `urn:oasis:names:tc:acal:1.0:data-type`. The first argument SHALL be an IPv4 or IPv6 address and the second argument SHALL be a regular expression. The function SHALL convert the first argument to type `urn:oasis:names:tc:acal:1.0:data-type:string` with `urn:oasis:names:tc:acal:1.0:function:string-from-ipAddress`, then apply `urn:oasis:names:tc:acal:1.0:function:regexp-match`.

- **urn:oasis:names:tc:acal:1.0:function:dnsName-regexp-match**

This function decides a regular expression match. It SHALL take two arguments; the first is of type `urn:oasis:names:tc:acal:1.0:data-type:dnsName` and the second is of type `urn:oasis:names:tc:acal:1.0:data-type:boolean`. It SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The first argument SHALL be a DNS name and the second argument SHALL be a regular expression. The function SHALL convert the first argument to type `urn:oasis:names:tc:acal:1.0:data-type:string` with `urn:oasis:names:tc:acal:1.0:function:string-from-dnsName`, then apply `urn:oasis:names:tc:acal:1.0:function:regexp-match`.

- **urn:oasis:names:tc:acal:1.0:function:rfc822Name-regexp-match**

This function decides a regular expression match. It SHALL take two arguments; the first is of type `urn:oasis:names:tc:acal:1.0:data-type:rfc822Name` and the second is of type `urn:oasis:names:tc:acal:1.0:data-type:string`. It SHALL return an `urn:oasis:names:tc:acal:1.0:data-type`. The first argument SHALL be an RFC 822 name and the second argument SHALL be a regular expression. The function SHALL convert the first argument to type `urn:oasis:names:tc:acal:1.0:data-type:string` with `urn:oasis:names:tc:acal:1.0:function:string-from-rfc822Name`, then apply `urn:oasis:names:tc:acal:1.0:function:regexp-match`.

- **urn:oasis:names:tc:acal:1.0:function:x500Name-regexp-match**

This function decides a regular expression match. It SHALL take two arguments; the first is of type `urn:oasis:names:tc:acal:1.0:data-type:x500Name` and the second is of type `urn:oasis:names:tc:acal:1.0:data-type:boolean`. It SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. The first argument SHALL be an X.500 directory name and the second argument SHALL be a regular expression. The function SHALL convert the first argument to type `urn:oasis:names:tc:acal:1.0:data-type:string` with `urn:oasis:names:tc:acal:1.0:function:string-from-x500Name`, then apply `urn:oasis:names:tc:acal:1.0:function:regexp-match`.

### C.3.14 Aggregate Functions

These functions perform a calculation on a bag of values to produce a summary result as a single value.

#### **urn:oasis:names:tc:acal:1.0:function:string-minimum**

This function returns the lexicographically least value in a bag of strings. The function takes one or two arguments. The first argument MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:string` data type. The second argument, if present, MUST be a value of the `urn:oasis:names:tc:acal:1.0:data-type:string` data type. If the first argument evaluates to a bag with at least one value then the function SHALL return the lexicographically least value in the bag, which may occur more than once. The comparison SHALL use Unicode codepoint collation, as defined for the identifier <https://www.w3.org/2005/xpath-functions/collation/codepoint/> by [XF]. If the first argument evaluates to an empty bag and the second argument is present, then the function SHALL return the value of the second argument. If the first argument evaluates to an empty bag and the second argument is absent, then the function SHALL return Indeterminate.

#### **urn:oasis:names:tc:acal:1.0:function:string-maximum**

This function returns the lexicographically greatest value in a bag of strings. The function takes one or two arguments. The first argument MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:string` data type. The second argument, if present, MUST be a value of the `urn:oasis:names:tc:acal:1.0:data-type:string` data type. If the first argument evaluates to a bag with at least one value then the function SHALL return the lexicographically greatest value in the bag, which may occur more than once. The comparison SHALL use Unicode codepoint collation, as defined for the identifier <https://www.w3.org/2005/xpath-functions/collation/codepoint/> by [XF]. If the first argument evaluates to an empty bag and the second argument is present, then the function SHALL return the value of the second argument. If the first argument evaluates to an empty bag and the second argument is absent, then the function SHALL return Indeterminate.

#### **urn:oasis:names:tc:acal:1.0:function:integer-minimum**

This function returns the least integer value in a bag of integers. The function takes one or two arguments.

The first argument MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:integer` data type. The second argument, if present, MUST be a value of the `urn:oasis:names:tc:acal:1.0:data-type:integer` data type. If the first argument evaluates to a bag with at least one value then the function SHALL return the least integer value in the bag, which may occur more than once. If the first argument evaluates to an empty bag and the second argument is present, then the function SHALL return the value of the second argument. If the first argument evaluates to an empty bag and the second argument is absent, then the function SHALL return Indeterminate.

#### **`urn:oasis:names:tc:acal:1.0:function:integer-maximum`**

This function returns the greatest integer value in a bag of integers. The function takes one or two arguments. The first argument MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:integer` data type. The second argument, if present, MUST be a value of the `urn:oasis:names:tc:acal:1.0:data-type:integer` data type. If the first argument evaluates to a bag with at least one value then the function SHALL return the greatest integer value in the bag, which may occur more than once. If the first argument evaluates to an empty bag and the second argument is present, then the function SHALL return the value of the second argument. If the first argument evaluates to an empty bag and the second argument is absent, then the function SHALL return Indeterminate.

#### **`urn:oasis:names:tc:acal:1.0:function:integer-sum`**

This function returns the sum of a bag of integers. The function takes one argument, which MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:integer` data type. If the argument evaluates to a bag with at least one value then the function SHALL return the sum of the values in the bag as a value of the `urn:oasis:names:tc:acal:1.0:data-type:integer` data type. If the argument evaluates to an empty bag, then the function SHALL return zero as an `urn:oasis:names:tc:acal:1.0:data-type:integer` value.

#### **`urn:oasis:names:tc:acal:1.0:function:integer-average`**

This function returns the average of a bag of integers. The function takes one or two arguments. The first argument MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:integer` data type. The second argument, if present, MUST be a value of the `urn:oasis:names:tc:acal:1.0:data-type:double` data type. If the first argument evaluates to a bag with at least one value then the function SHALL return the average of the values in the bag as a value of the `urn:oasis:names:tc:acal:1.0:data-type:double` data type. If the first argument evaluates to an empty bag and the second argument is present, then the function SHALL return the value of the second argument. If the first argument evaluates to an empty bag and the second argument is absent, then the function SHALL return Indeterminate.

#### **`urn:oasis:names:tc:acal:1.0:function:double-minimum`**

This function returns the least numeric value in a bag of floating-point numbers. The function takes one or two arguments. The first argument MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:double` data type. The second argument, if present, MUST be a value of the `urn:oasis:names:tc:acal:1.0:data-type:double` data type. If the first argument evaluates to a bag with at least one value then the function SHALL return the least numeric value in the bag, which may occur more than once. If the first argument evaluates to an empty bag and the second argument is present, then the function SHALL return the value of the second argument. If the first argument evaluates to an empty bag and the second argument is absent, then the function SHALL return Indeterminate.

#### **`urn:oasis:names:tc:acal:1.0:function:double-maximum`**

This function returns the greatest numeric value in a bag of floating-point numbers. The function takes one or two arguments. The first argument MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:double` data type. The second argument, if present, MUST be a value of the `urn:oasis:names:tc:acal:1.0:data-type:double` data type. If the first argument evaluates to a bag with at least one value then the function SHALL return the greatest numeric value in the bag, which may occur more than once. If the first argument evaluates to an empty bag and the second argument is present, then the function SHALL return the value of the second argument. If the first argument evaluates to an empty bag and the second argument is absent, then the function SHALL return Indeterminate.

#### **`urn:oasis:names:tc:acal:1.0:function:double-sum`**

This function returns the sum of a bag of floating-point numbers. The function takes one argument, which MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:double` data type. If the argument evaluates to a bag with at least one value then the function SHALL return the sum of the values in the bag as a value of the `urn:oasis:names:tc:acal:1.0:data-type:double` data type. If the argument evaluates to an empty bag, then the function SHALL return zero as an `urn:oasis:names:tc:acal:1.0:data-type:double` value.

**urn:oasis:names:tc:acal:1.0:function:double-average**

This function returns the average of a bag of floating-point numbers. The function takes one or two arguments. The first argument MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:double` data type. The second argument, if present, MUST be a value of the `urn:oasis:names:tc:acal:1.0:data-type:double` data type. If the first argument evaluates to a bag with at least one value then the function SHALL return the average of the values in the bag as a value of the `urn:oasis:names:tc:acal:1.0:data-type:double` data type. If the first argument evaluates to an empty bag and the second argument is present, then the function SHALL return the value of the second argument. If the first argument evaluates to an empty bag and the second argument is absent, then the function SHALL return `Indeterminate`.

**urn:oasis:names:tc:acal:1.0:function:dateTime-minimum**

This function returns the least `dateTime` in a bag of `dateTime` values. The function takes one or two arguments. The first argument MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:dateTime` data type. The second argument, if present, MUST be a value of the `urn:oasis:names:tc:acal:1.0:data-type:dateTime` data type. If a `dateTime` value does not include a time-zone, then the time-zone SHALL be assumed to be `+14:00`. If such a value is determined to be the least `dateTime` then it is returned as is, without a time-zone. If the first argument evaluates to a bag with at least one value then the function SHALL return the least value in the bag, which may occur more than once. The comparison SHALL use the order relation specified for <https://www.w3.org/2001/XMLSchema#dateTime> by [XS], part 2, Section 3.2.7]. If multiple `dateTime` values are equally least, then a value with a time-zone MUST be preferred over one without. If the first argument evaluates to an empty bag and the second argument is present, then the function SHALL return the value of the second argument. If the first argument evaluates to an empty bag and the second argument is absent, then the function SHALL return `Indeterminate`.

**urn:oasis:names:tc:acal:1.0:function:dateTime-maximum**

This function returns the greatest `dateTime` in a bag of `dateTime` values. The function takes one or two arguments. The first argument MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:dateTime` data type. The second argument, if present, MUST be a value of the `urn:oasis:names:tc:acal:1.0:data-type:dateTime` data type. If a `dateTime` value does not include a time-zone, then the time-zone SHALL be assumed to be `-14:00`. If such a value is determined to be the greatest `dateTime` then it is returned as is, without a time-zone. If the first argument evaluates to a bag with at least one value then the function SHALL return the greatest value in the bag, which may occur more than once. The comparison SHALL use the order relation specified for <https://www.w3.org/2001/XMLSchema#dateTime> by [XS] part 2, Section 3.2.7]. If multiple `dateTime` values are equally greatest, then a value with a time-zone MUST be preferred over one without. If the first argument evaluates to an empty bag and the second argument is present, then the function SHALL return the value of the second argument. If the first argument evaluates to an empty bag and the second argument is absent, then the function SHALL return `Indeterminate`.

**urn:oasis:names:tc:acal:1.0:function:date-minimum**

This function returns the least `date` in a bag of `date` values. The function takes one or two arguments. The first argument MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:date` data type. The second argument, if present, MUST be a value of the `urn:oasis:names:tc:acal:1.0:data-type:date` data type. If a `date` value does not include a time-zone, then the time-zone SHALL be assumed to be `+14:00`. If such a value is determined to be the least `date` then it is returned as is, without a time-zone. If the first argument evaluates to a bag with at least one value then the function SHALL return the least value in the bag, which may occur more than once. The comparison SHALL use the order relation specified for <https://www.w3.org/2001/XMLSchema#date> by [XS] part 2, Section 3.2.9]. If multiple `date` values are equally least, then a value with a time-zone MUST be preferred over one without. If the first argument evaluates to an empty bag and the second argument is present, then the function SHALL return the value of the second argument. If the first argument evaluates to an empty bag and the second argument is absent, then the function SHALL return `Indeterminate`.

**urn:oasis:names:tc:acal:1.0:function:date-maximum**

This function returns the greatest `date` in a bag of `date` values. The function takes one or two arguments. The first argument MUST be a bag of the `urn:oasis:names:tc:acal:1.0:data-type:date` data type. The second argument, if present, MUST be a value of the `urn:oasis:names:tc:acal:1.0:data-type:date` data type. If a `date` value does not include a time-zone, then the time-zone SHALL be assumed to be `-14:00`. If such a value is determined to be the greatest `date` then it is returned as is, without a time-zone. If the first argument evaluates to a bag with at least one value then the function SHALL return the greatest value in the bag, which may occur more than once. The comparison SHALL use the order relation specified for <https://www.w3.org/2001/XMLSchema#date> by [XS] part 2, Section 3.2.9]. If multiple `date` values are equally greatest, then a value with a time-zone MUST be preferred over one without. If the first argument evaluates to an empty bag and the second argument is present, then the function SHALL

return the value of the second argument. If the first argument evaluates to an empty bag and the second argument is absent, then the function SHALL return **Indeterminate**.

### C.3.15 Special Match Functions

These functions operate on various types and evaluate to `urn:oasis:names:tc:acal:1.0:data-type:boolean` based on the specified standard matching algorithm.

#### `urn:oasis:names:tc:acal:1.0:function:x500Name-match`

This function shall take two arguments of `urn:oasis:names:tc:acal:1.0:data-type:x500Name` and shall return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. It shall return `true` if and only if the second argument matches some terminal sequence of RDNs from the first argument when compared using `x500Name-equal`.

As an example (non-normative), if the first argument is `cn=John Smith,o=Medico Corp,c=US` and the second argument is `O=Medico Corp,C=US`, then the function will return `true`.

Note that the arguments to this function are swapped compared to the similarly-named function in XACML [XACML].

#### `urn:oasis:names:tc:acal:1.0:function:rfc822Name-match`

This function SHALL take two arguments, the first is of data type `urn:oasis:names:tc:acal:1.0:data-type:rfc822Name` and the second is of data type `urn:oasis:names:tc:acal:1.0:data-type:string` and SHALL return an `urn:oasis:names:tc:acal:1.0:data-type:boolean`. This function SHALL evaluate to `true` if the second argument matches the first 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 first argument contains a complete rfc822Name. The second argument is a complete or partial rfc822Name used to select appropriate values in the first argument as follows.

In order to match a particular address in the first argument, the second argument must specify the complete mail address to be matched. For example, if the second argument is `Anderson@sun.com`, this matches a value in the first 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 first argument, the second argument must specify only a domain name (usually a DNS name). For example, if the second argument is `sun.com`, this matches a value in the first argument of `Anderson@sun.com` or `Baxter@SUN.COM`, but not `Anderson@east.sun.com`.

In order to match any address in a particular domain in the first argument, the second argument must specify the desired domain-part with a leading ... For example, if the second argument is `.east.sun.com`, this matches a value in the first argument of `Anderson@east.sun.com` and `anne.anderson@ISRG.EAST.SUN.COM` but not `Anderson@sun.com`.

Note that the arguments to this function are swapped compared to the similarly-named function in XACML [XACML].

### C.3.16 Other Functions

#### `urn:oasis:names:tc:acal:1.0:function:access-permitted`

This function SHALL take an `urn:oasis:names:tc:acal:1.0:data-type:anyURI` and an `urn:oasis:names:tc:acal:` as arguments. The first argument SHALL be interpreted as an attribute category. The second argument SHALL be interpreted as a `RequestEntityType` object with the `Category` property set to the value of first argument. The function evaluates to an `urn:oasis:names:tc:acal:1.0:data-type: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 `RequestEntityType` object with `Category` property equal to the first argument. The second function argument SHALL be added to the context as a `RequestEntityType` object with the `Category` property set to the value of 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:acal:1.0:status:processing-error` status code. Also, see the security considerations in Section 10.1.8.

### C.3.18 Extension Functions and Data Types

Functions and data types are specified by string identifiers allowing for the introduction of functions in addition to those specified by ACAL. This approach allows one to extend the ACAL module with special functions and special data types.

In order to preserve the integrity of the ACAL 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.

## Annex D ACAL Identifiers

(This annex forms an integral part of this Specification.)

This section defines standard identifiers for commonly used definitions.

### D.1 ACAL Namespaces

ACAL 1.0 is defined using this identifier.

`urn:oasis:names:tc:acal:1.0:core:schema`

### D.2 Attribute Categories

The following attribute category identifiers MUST be used when an XACML 3.0 or earlier policy or request is translated into ACAL 1.0.

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 ACAL policies or requests.

`urn:oasis:names:tc:acal:1.0:attribute-category:resource`

`urn:oasis:names:tc:acal:1.0:attribute-category:action`

`urn:oasis:names:tc:acal:1.0:attribute-category:environment`

Attributes previously placed in the Subject section of a request are placed in an attribute category which is identical to the subject category in XACML 2.0, as defined below. It is RECOMMENDED that they are used to list attributes of subjects when authoring ACAL 1.0 policies or requests.

This identifier indicates the system entity that initiated the access request, that is, the initial entity in a request chain; if the subject category is not specified in XACML 2.0, this is also the default translation value:

`urn:oasis:names:tc:acal: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:acal:1.0:subject-category:recipient-subject`

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

`urn:oasis:names:tc:acal: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:acal:1.0:subject-category:codebase`

This identifier indicates a system entity associated with the computer that initiated the access request (for example, an IPsec identity):

`urn:oasis:names:tc:acal:1.0:subject-category:requesting-machine`

### D.3 Data Types

The following identifiers indicate data types that are defined in Annex C.2:

- `urn:oasis:names:tc:acal:1.0:data-type:x500Name`
- `urn:oasis:names:tc:acal:1.0:data-type:rfc822Name`
- `urn:oasis:names:tc:acal:1.0:data-type:ipAddress`
- `urn:oasis:names:tc:acal:1.0:data-type:dnsName`

The following data type identifiers are defined by W3C XML Schema [XS] (each `urn:oasis:names:tc:acal:1.0:data-type:<name>` below corresponds to the `xs:<name>` type in the XML schema Data Types specification):

- `urn:oasis:names:tc:acal:1.0:data-type:string`
- `urn:oasis:names:tc:acal:1.0:data-type:boolean`
- `urn:oasis:names:tc:acal:1.0:data-type:integer`
- `urn:oasis:names:tc:acal:1.0:data-type:double`
- `urn:oasis:names:tc:acal:1.0:data-type:time`
- `urn:oasis:names:tc:acal:1.0:data-type:date`
- `urn:oasis:names:tc:acal:1.0:data-type:dateTime`
- `urn:oasis:names:tc:acal:1.0:data-type:anyURI`
- `urn:oasis:names:tc:acal:1.0:data-type:hexBinary`
- `urn:oasis:names:tc:acal:1.0:data-type:base64Binary`.
- `urn:oasis:names:tc:acal:1.0:data-type:dayTimeDuration`
- `urn:oasis:names:tc:acal:1.0:data-type:yearMonthDuration`.

### D.4 Subject Attributes

These identifiers indicate attributes of a subject. When used, it is RECOMMENDED that they appear within a `RequestEntityType` object of the request context with a subject category (see Annex D.2).

At most one of each of these attributes is associated with each subject. Each attribute associated with authentication included within a single `RequestEntityType` object relates to the same authentication event.

This identifier indicates the name of the subject:

`urn:oasis:names:tc:acal: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:acal:1.0:subject:subject-id-qualifier`

This identifier indicates a public key used to confirm the subject's identity:

`urn:oasis:names:tc:acal:1.0:subject:key-info`

This identifier indicates the time at which the subject was authenticated:

`urn:oasis:names:tc:acal:1.0:subject:authentication-time`

This identifier indicates the method used to authenticate the subject:

`urn:oasis:names:tc:acal: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:acal: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:acal: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:acal:1.0:subject:authn-locality:ip-address`

The corresponding attribute SHALL be of data type `urn:oasis:names:tc:acal:1.0:data-type:ipAddress`.

- This identifier indicates that the location is expressed as a DNS name.

`urn:oasis:names:tc:acal:1.0:subject:authn-locality:dns-name`

The corresponding attribute SHALL be of data type `urn:oasis:names:tc:acal:1.0:data-type:dnsName`.

Where a suitable attribute is already defined in LDAP [LDAP-1], [LDAP-2], the ACAL identifier SHALL be the OID of the LDAP attribute represented as a URN. For example, the ACAL attribute identifier for the LDAP `title` attribute defined in RFC 2256 with the OID 2.5.4.12 would be `urn:oid:2.5.4.12`.

Short identifiers can be used to improve readability. For example `title` could be defined as a short identifier name for `urn:oid:2.5.4.12`.

## D.5 Resource Attributes

These identifiers indicate attributes of the resource. When used, it is RECOMMENDED they appear within the `RequestEntityType` object of the request context with the `Category` property that evaluates to `urn:oasis:names:tc:acal:1.0:attribute-category:resource`.

This attribute identifies the resource to which access is requested:

`urn:oasis:names:tc:acal:1.0:resource:resource-id`

**This attribute identifies the namespace of the top element(s) of the contents of the `ContentType` object's Body:**

`urn:oasis:names:tc:acal:1.0:resource:target-namespace`

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 `ContentType` object's Body. The data type of the corresponding attribute SHALL be `urn:oasis:names:tc:acal:1.0:data-type:anyURI`.

This attribute is used to identify the location of a resource, such as a file path or a network address:

`urn:oasis:names:tc:acal:1.0:resource:resource-location`

This attribute indicates the last (rightmost) component of a file name identifying the resource. For example, if the URI is `file://home/my/status#pointer`, then the simple file name is "status":

`urn:oasis:names:tc:acal:1.0:resource:simple-file-name`

## D.6 Action Attributes

These identifiers indicate attributes of the action being requested. When used, it is RECOMMENDED they appear within the `RequestEntityType` object of the request context with the `Category` property that evaluates to `urn:oasis:names:tc:acal:1.0:attribute-category:action`.

This attribute identifies the action for which access is requested:

`urn:oasis:names:tc:acal:1.0:action:action-id`

Where the action is implicit, the value of the `action-id` attribute SHALL be:

`urn:oasis:names:tc:acal:1.0:action:implied-action`

This attribute identifies the namespace in which the `action-id` attribute is defined:

`urn:oasis:names:tc:acal:1.0:action:action-namespace`

## D.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 RequestEntityType object of the request context with the Category property that evaluates to urn:oasis:names:tc:acal:1.0:attribute-category:environment.

**This identifier indicates the current time at the context handler:**

`urn:oasis:names:tc:acal:1.0:environment:current-time`

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, 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 `urn:oasis:names:tc:acal:1.0:data-type:time`.

**This identifier indicates the current date at the context handler:**

`urn:oasis:names:tc:acal:1.0:environment:current-date`

The corresponding attribute SHALL be of data type `urn:oasis:names:tc:acal:1.0:data-type:date`.

**This identifier indicates the current date and time at the context handler:**

`urn:oasis:names:tc:acal:1.0:environment:current-datetime`

The corresponding attribute SHALL be of data type `urn:oasis:names:tc:acal:1.0:data-type:dateTime`.

## D.8 Status Codes

The following status code values are defined.

This identifier indicates success:

`urn:oasis:names:tc:acal:1.0:status:ok`

This identifier indicates that all the attributes necessary to make a policy decision were not available (see Section 7.44):

`urn:oasis:names:tc:acal: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:acal: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:acal:1.0:status:processing-error`

## D.9 Combining Algorithms

The deny-overrides combining algorithm has the following value for the CombiningAlgId property:

`urn:oasis:names:tc:acal:1.0:combining-algorithm:deny-overrides`

The permit-overrides combining algorithm has the following value for the CombiningAlgId property:

`urn:oasis:names:tc:acal:1.0:combining-algorithm:permit-overrides`

The first-applicable combining algorithm has the following value for the CombiningAlgId property:

`urn:oasis:names:tc:acal:1.0:combining-algorithm:first-applicable`

The ordered-deny-overrides combining algorithm has the following value for the CombiningAlgId property:

`urn:oasis:names:tc:acal:1.0:combining-algorithm:ordered-deny-overrides`

The ordered-permit-overrides combining algorithm has the following value for the CombiningAlgId property:

`urn:oasis:names:tc:acal:1.0:combining-algorithm:ordered-permit-overrides`

The deny-unless-permit combining algorithm has the following value for the CombiningAlgId property:

`urn:oasis:names:tc:acal:1.0:combining-algorithm:deny-unless-permit`

The permit-unless-deny combining algorithm has the following value for the CombiningAlgId property:

## D.10 Content Types

These standard *Media Type* identifiers ([RFC 2046]) - registered at IANA ([RFC6838]) - SHALL be used as values for the `MediaType` property of `ContentType` objects, whenever applicable to the Content type:

- For XML content: `application/xml`;
- For JSON content: `application/json`.

## D.11 Content Encodings

This standard *Encoding mechanism* identifier ([RFC 2045] Section 6) - registered at IANA ([RFC4289]) - SHALL be used as values for the `Encoding` property of `ContentType` objects, whenever applicable to the Content type:

- `base64`.
- 

# Annex E Combining Algorithms

(This annex forms an integral part of this Specification.)

This section contains a description of the combining algorithms specified by ACAL. Pseudo code is normative, descriptions in English are non-normative.

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.

## E.1 Extended Indeterminate Values

Some combining algorithms are defined in terms of an extended set of `Indeterminate` values. See Section 8.10 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`.

## E.2 Deny Overrides

This section defines the `deny-overrides` combining algorithm of a policy.

This combining algorithm makes use of the extended `Indeterminate`.

The combining algorithm defined here has the following identifier:

The following is a non-normative 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 or rules) of the policy. The children may be processed in any order, so the list of notices 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;
}
```

Notices SHALL be combined as described in Section 8.16.

## E.3 Ordered Deny Overrides

The following specification defines the `ordered-deny-overrides` combining algorithm of a policy. The behavior of this algorithm is identical to that of the `deny-overrides` combining algorithm with one exception: the order in which the collection of policies and rules is evaluated SHALL match the order as listed in the policy.

The combining algorithm defined here has the following identifier:

urn:oasis:names:tc:acal:1.0:combining-algorithm:ordered-deny-overrides

## E.4 Permit Overrides

This section defines the `permit-overrides` combining algorithm of a policy.

This combining algorithm makes use of the extended `Indeterminate`.

The combining algorithm defined here has the following identifier:

urn:oasis:names:tc:acal:1.0:combining-algorithm:permit-overrides

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 or rules) of the policy. The children may be processed in any order, so the list of notices provided by this algorithm is not deterministic.

```
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;
}

```

Notices SHALL be combined as described in Section 8.16.

## E.5 Ordered Permit Overrides

The following specification defines the `ordered-permit-overrides` combining algorithm of a policy. The behavior of this algorithm is identical to that of the `permit-overrides` combining algorithm with one exception: the order in which the collection of policies and rules is evaluated SHALL match the order as listed in the policy.

The combining algorithm defined here has the following identifier:

`urn:oasis:names:tc:acal:1.0:combining-algorithm:ordered-permit-overrides`

## E.6 Deny Unless Permit

This section defines the `deny-unless-permit` combining algorithm of a policy.

The combining algorithm defined here has the following identifier:

`urn:oasis:names:tc:acal:1.0:combining-algorithm:deny-unless-permit`

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 or rules) of the policy. The children may be processed in any order, so the list of notices provided by this algorithm is not deterministic.

```

Decision denyUnlessPermitCombiningAlgorithm(Node[] children)
{
    for( i=0 ; i < lengthOf(children) ; i++ )
    {
        if (children[i].evaluate() == Permit)
        {
            return Permit;
        }
    }
    return Deny;
}

```

Notices SHALL be combined as described in Section 8.16.

## E.7 Permit Unless Deny

This section defines the `permit-unless-deny` combining algorithm of a policy.

The combining algorithm defined here has the following identifier:

urn:oasis:names:tc:acal:1.0: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 or rules) of the policy. The children may be processed in any order, so the list of notices provided by this algorithm is not deterministic.

```
Decision permitUnlessDenyCombiningAlgorithm(Node[] children)
{
    for( i=0 ; i < lengthOf(children) ; i++ )
    {
        if (children[i].evaluate() == Deny)
        {
            return Deny;
        }
    }
    return Permit;
}
```

Notices SHALL be combined as described in Section 8.16.

## E.8 First Applicable

This section defines the **first-applicable** combining algorithm of a policy.

The combining algorithm defined here has the following identifier:

urn:oasis:names:tc:acal:1.0:combining-algorithm:first-applicable

The following is a non-normative informative description of the **first-Applicable** combining algorithm of a policy.

For a particular child policy or rule, if that policy or rule evaluates to **Permit**, **Deny** or **Indeterminate**, then the evaluation SHALL halt and the enclosing policy shall evaluate to the value of that policy or rule. If the policy or rule evaluates to **NotApplicable**, then the next child policy or rule in the order SHALL be evaluated. If no further child policy or rule exists in the order, then the enclosing policy shall evaluate to **NotApplicable**. If an error occurs while evaluating a policy or rule, or a policy reference is considered invalid, then the evaluation SHALL halt, and the enclosing policy shall evaluate to **Indeterminate**, with the appropriate error status.

The following pseudo-code represents the normative specification of this combining algorithm.

```
Decision firstApplicableEffectRuleCombiningAlgorithm(Node[] children)
{
    for( i = 0 ; i < lengthOf(children) ; i++ )
    {
        Decision decision = children[i].evaluate();
        if (decision == Deny)
        {
            return Deny;
        }
        if (decision == Permit)
        {
            return Permit;
        }
        if (decision == NotApplicable)
        {
            continue;
        }
        if (decision == Indeterminate)
        {
            return Indeterminate;
        }
    }
}
```

```
    return NotApplicable;
}
```

Notices SHALL be combined as described in Section 8.16.

---

## Annex F How to generate HTML and PDF Versions

### Online generation

HTML/PDF versions are generated automatically online via Github Actions after each update pushed to the main branch of OASIS XACML TC Github repository. Go to Github Actions on the github repository, then go to the latest workflow run, and, if the run succeeded, the summary should display the links to the generated HTML/PDF documents.

### Offline generation

#### Prerequisites

Install Pandoc, Graphviz and PlantUML on your system; or simply use Docker with the following shell alias:

```
$ alias pandoc='docker run --rm --volume "$(pwd):/data" cdang/pandoc-plantuml'
```

*The Dockerfile (named Dockerfile) of the docker image used in the alias above is provided in the pandoc folder next to this markdown file for your convenience if you wish to build it yourself.*

OASIS staff are currently using pandoc 3.0 from <https://github.com/jgm/pandoc/releases/tag/3.0>.

Git clone or get a local copy of OASIS XACML TC Github repository, open a terminal and **change your working directory to the root directory of your local copy of the repository**.

#### CSS stylesheet

The generation command uses a CSS stylesheet file (-c argument) provided by OASIS. It may be changed to one of these (or the local version in the `styles` folder) to get a different style of output:

- <https://docs.oasis-open.org/templates/css/markdown-styles-v1.7.3.css>
- <https://docs.oasis-open.org/templates/css/markdown-styles-v1.7.3a.css> (this one produces HTML that resembles the github display more closely, especially for blocks of code) This template already includes a reference (in HTML code) to this .css file.
- [https://docs.oasis-open.org/templates/css/markdown-styles-v1.8.1-cn\\_final.css](https://docs.oasis-open.org/templates/css/markdown-styles-v1.8.1-cn_final.css)

#### HTML generation

Run the following command line to generate HTML from this markdown file (named `acal-core-v1.0-csd01.md`) to an output file `/tmp/acal-core-v1.0-csd01.html`:

```
$ pandoc -f gfm+definition_lists -t html -c styles/markdown-styles-v1.7.3a.css -s --lua-filter pandoc/diagram.lua --defaults pa
```

Note this command generates a Table of Contents (TOC) in HTML which is located at the top of the HTML document, and which requires additional editing in order to be published in the expected OASIS style. This editing will be handled by OASIS staff during publication.

#### PDF generation

For PDF output (file `/tmp/acal-core-v1.0-csd01.pdf`), the command line is the following (different -t and -H arguments):

```
$ pandoc -f gfm+definition_lists -t pdf -c styles/markdown-styles-v1.7.3a.css -H pandoc/custom_latex_header_for_pandoc_pdf_outp
```

---

## Appendix 1 Acknowledgments

(This appendix does not form an integral part of this Specification and is informational.)

## Leadership

The following individuals have had significant leadership positions during the development of this document, not just this version of the document, and they are gratefully acknowledged:

- Chairs
  - Bill Parducci, Individual
- Secretaries
  - Bill Parducci, Individual
- Editors
  - Steven Legg, ViewDS Identity Solutions
  - Cyril Dangerville, THALES

## Special Thanks

The following individuals have made substantial contributions to this document, not just this version of the document, and their contributions are gratefully acknowledged:

- Steven Legg, ViewDS Identity Solutions
- Cyril Dangerville, THALES

## Participants

The following individuals were members of this committee during the creation of this document, not just this version of the document, and their contributions are gratefully acknowledged:

### XACML TC Members:

- Hal Lockhart, Individual
  - Bill Parducci, Individual
  - Steven Legg, ViewDS Identity Solutions
  - Cyril Dangerville, THALES
- 

## Appendix 2 Changes From Previous Version

(This appendix does not form an integral part of this Specification and is informational.)

ACAL 1.0 is a successor to XACML 3.0. ACAL 1.0 differs from XACML 3.0 in the following ways:

- ACAL constructs can be represented in JSON, YAML or XML at the implementor's discretion.
- The policy (**Policy**) and policy set (**PolicySet**) constructs have been merged into a single construct which is known as a policy (**Policy**) in ACAL 1.0.
  - Compared to XACML 3.0, the **<PolicySet>**, **<PolicySetIdReference>**, **<PolicySetCombinerParameters>** and **<PolicySetDefaults>** elements and **PolicySet** type no longer appear.
  - Compared to XACML 3.0, the **PolicyType** type now allows child **<Policy>** and **<PolicyReference>** (was **PolicyIdReference**) elements.
  - Compared to XACML 3.0, a **Policy** may contain more than one **PolicyDefaults** element, one per ACAL Profile possibly.
  - Separate rule and policy combining algorithms have been replaced with a single collection of combining algorithms. Legacy combining algorithms have been removed. The **only-one-applicable** policy combining algorithm has been removed.
  - EarliestVersion and LatestVersion attributes removed from **<PolicyReference>**
- The target (**Target** element) of a policy has been changed to have the same structure as the condition of a rule (i.e., a single Boolean expression). From the XML perspective, this means that the **<AnyOf>**, **<AllOf>** and **<Match>** elements no longer appear.
- Rules no longer have a target (**Target** element removed).

- Obligations and advice no longer have distinct syntactic representations. Instead they now share the common `NoticeType` object type. The difference between obligations and advice is indicated by an `IsObligation` property. (XML elements `ObligationExpressions`, `ObligationExpression`, `Obligations`, `Obligation`, `AdviceExpressions`, `AdviceExpression`, `AssociatedAdvice`, `Advice` replaced with `NoticeExpression` and `Notice`.)
- Combiner parameters are no longer supported and the `<CombinerParameter>`, `<CombinerParameters>`, `<PolicyCombinerParameters>` and `<RuleCombinerParameters>` elements no longer appear.
- Users are able to define short identifiers (`ShortIdSet` objects), which provide simple alias names to use in place of absolute URIs to refer to ACAL definitions. A predefined set of short identifiers for standard-defined URIs is also provided.
- The `IncludeInResult` XML attribute of the `<Attribute>` element has been prohibited in results and entity data type values. It is meaningless in these cases.
- `IncludeInResult`, `ReturnPolicyIdList` and `CombinedDecision` attributes (in a `Request`) are now optional with default value `false`.
- A `Request` may contain more than one `RequestDefaults` element, one per ACAL Profile possibly.
- Different types are now used to represent attribute categories in a request versus a response. Categories in the result don't have the `Content` property, since there is no mechanism to request their inclusion, and their attributes don't have the `IncludeInResult` property because it is meaningless in that context. Categories in the request have both `Content` properties and attributes with `IncludeInResult` properties. Attribute categories in the request are `RequestEntityType` objects and attributes in the response are `ResultEntityType` objects. These types supersede the XML Schema `AttributesType` complex type. As a result:
  - `Attributes` element replaced with new `RequestEntity` and `ResultEntity` elements in the `Request` and `Result` elements respectively
  - `AttributesReference` replaced with new `RequestEntityReference`.
- `Attribute` has a new optional `DataType` attribute (set to the standard string type as default), to have a common data-type for all the attribute values. In XACML 3.0, defining the data-type for each `AttributeValue` individually led to the risk of mixing different datatypes in the same attribute. We consider this bad practice (one may simply split in different attributes if using different data-types).
- `AttributeValue` is replaced with more generic `Value` whose `DataType` is now optional, i.e. it may be omitted when it is already defined by at the parent or ancestor level, which simplifies expressions with literal values. In particular, as a result of the previous change, in an `Attribute`, the `Value`'s `DataType` is now omitted since defined at the `Attribute` level already.
- A `Content` may now contain XML, JSON and possibly other content types in the future, and therefore has a new data model - more generic - that consists of three properties: `Body` (content body which is similar to former `Content` value), `MediaType` (Content media type which indicates whether it is XML, JSON or some other content type) and `Encoding` which indicates whether the body is encoded with a particular mechanism, e.g. Base64-encoded data.
- `AttributeSelector` changes:
  - `DataType` attribute changed to be optional with the standard string type as default value to simplify the element declaration in most cases.
  - `MustBePresent`: changed to be optional with `false` as default value, to simplify the element declaration in most cases.
  - `AttributeSelector` and `Path` type of expression are abstract in ACAL model, concrete types of `AttributeSelector` `Path` expressions to be defined in ACAL Profiles, e.g. XPath Profile.
- `AttributeDesignator` changes to simplify the declaration in most cases:
  - `DataType` attribute changed to be optional, with the standard string type as default.
  - `MustBePresent`: changed to be optional with `false` as default value.
- The `xml:id` attribute ([`XMLId`]) for an attribute category is replaced by a generic 'Id' property so that all ACAL representation formats are on an equal footing.
- The quantified expressions (`ForAny`, `ForAll`, `Select` and `Map`) and the `entity` data type from the XACML 3.0 Entities Profile [ENTITIES] have been incorporated in ACAL version 1.0. The `attribute-designator`

function from the profile has been reinvented as the `EntityAttributeDesignatorType` expression object type and the `attribute-selector` function from the the profile has been reinvented as the `EntityAttributeSelectorType` expression object type.

- Added new `BundleType` object which allows to bundle variable definitions (`SharedVariableDefintion`), short identifiers (`ShortIdSet`) and policies together in order to be portable / reused from one ACAL system to another, more particularly one PDP to another.
- The arguments to each of the following functions have been swapped: `string-starts-with`, `anyURI-starts-with`, `string-ends-with`, `anyURI-ends-with`, `string-contains`, `anyURI-contains`, `x500Name-match`, `rfc822Name-match`, `string-regexp-match`, `anyURI-regexp-match`, `ipAddress-regexp-match`, `dnsName-regexp-match`, `rfc822Name-regexp-match` and `x500Name-regexp-match`.
- Added a new logical function (annex C.3) commonly known as *ternary (conditional) operator*: `function ternary-if(a,b,c) -> if a is true, return b, else return c.`
- Added new aggregate functions (annex C.3) for computing:
  - Sums and averages of integers/doubles;
  - Minimum and maximum values among integers, doubles, strings, times, dates, date-times.
- XPath features moved to separate ACAL XPath Profile: XPath-based AttributeSelector, XPath-based functions, XPath expression datatype.
- Deprecated prefixes `urn:oasis:names:tc:xacml:` and `https://www.w3.org/2001/XMLSchema#` in favor of `urn:oasis:names:tc:acal:` for all standard identifiers (algorithms, status codes, data-types, functions, attributes and categories)

## Revision History

Latest revision history can be obtained from OASIS XACML TC's github repository.

---