



---

# eXtensible Access Control Markup Language (XACML) Version 4.0 (XML Representation of ACAL Version 1.0)

Committee Specification Draft 01

18 February 2026

**This version:**

- [\(Authoritative\) \](https://docs.oasis-open.org/xacml/acal/xacml/core/v4.0/csd01/acal-core-xml-v4.0-csd01.html)
- [\](https://docs.oasis-open.org/xacml/acal/xacml/core/v4.0/csd01/acal-core-xml-v4.0-csd01.pdf)
- <https://docs.oasis-open.org/xacml/acal/xacml/core/v4.0/csd01/acal-core-xml-v4.0-csd01.md>

**Previous version:**

- [\(Authoritative\)](http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.doc)
- <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html>
- <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.pdf>

**Latest version:**

- [\(Authoritative\) \](https://docs.oasis-open.org/xacml/acal/xacml/core/v4.0/csd01/acal-core-xml-v4.0-csd01.html)
- [\](https://docs.oasis-open.org/xacml/acal/xacml/core/v4.0/csd01/acal-core-xml-v4.0-csd01.pdf)
- <https://docs.oasis-open.org/xacml/acal/xacml/core/v4.0/csd01/acal-core-xml-v4.0-csd01.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

**Additional artifacts:**

This document is one component of a Work Product that also includes:

- Core XML schema: acal-core-xml-v4.0-schema.xsd
- Core Schematron rules: acal-core-xml-v4.0-schematron.sch
- Short identifier set: acal-core-xml-v4.0-identifiers.xml
- XPath Profile XML schema: acal-xpath-xml-v4.0-schema.xsd
- JSONPath Profile XML schema: acal-jsonpath-xml-v4.0-schema.xsd

## **Abstract:**

This specification defines Version 4.0 of the eXtensible Access Control Markup Language which now represents the XML representation Profile of the ACAL Version 1.0, also produced by the same OASIS XACML Technical Committee.

## **Citation format:**

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

[**XACML-Core-4.0**] *eXtensible Access Control Markup Language (XACML) Version 4.0*. Edited by Steven Legg and Cyril Dangerville. 18 February 2026. OASIS Committee Specification Draft 01. <https://docs.oasis-open.org/xacml/acal/xacml/core/v4.0/csd01/acal-core-xml-v4.0-csd01.html>. Latest stage: <https://docs.oasis-open.org/xacml/acal/xacml/core/v4.0/csd01/acal-core-xml-v4.0-csd01.html>.

## **Related work:**

This specification 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 specification is related to:

- *Attribute-Centric Authorization Language (ACAL) Version 1.0*. Edited by Steven Legg and Cyril Dangerville. 18 February 2026. OASIS Committee Specification Draft 01.

## **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
  - 3.3 Schema organization and namespaces
- 4 Introduction (non-normative)
  - 4.1 Requirements
  - 4.2 Abstraction Layer
  - 4.3 Example Short Identifier Set
  - 4.4 Changes From the Previous Version
- 5 Syntax (normative, with the exception of the schema fragments)
  - 5.1 Mapping ACAL primitive types
    - 5.1.1 Primitive types mapped to standard XSD data-types
    - 5.1.2 Restricted String types (UML stereotype <<restrictedString>>)
    - 5.1.3 Enum types (UML stereotype <<enumeration>>)
  - 5.2 Mapping complex ACAL types (UML stereotype <<complexType>>)
    - 5.2.1 AnyType mapping rule
    - 5.2.2 Single-use empty datatypes
    - 5.2.3 ValueType mapping rules
    - 5.2.4 Default mapping rules for complex ACAL types (other than ValueType)
    - 5.2.5 Property mapping rules

- 5.2.6 Mapping complex ACAL constraints (OCL)
    - 5.2.6.1 Option 1: XML Schema 1.1 assertions
    - 5.2.6.2 Option 2: Schematron rules
  - 5.3 Content Types and Body representations (optional)
  - 6 Safety, Security and Privacy Considerations (non-normative)
    - 6.1 Threat model
    - 6.2 Safeguards
      - 6.2.1 Policy confidentiality
      - 6.2.2 Policy integrity
  - 7 Conformance
    - 7.1 Introduction
    - 7.2 Conformance tables
      - 7.2.1 Schema elements
  - 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 XACML identifiers (normative)
    - C.1 XACML namespaces
  - Annex D XML Schema (normative)
  - Annex E 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 the XML representation format of the [ACAL-Core-1.0] model and any XML-specific syntax, semantics and processing instructions that are not already specified by [ACAL-Core-1.0]. For more information on the scope, please refer to [ACAL-Core-1.0].

---

## 2 Definitions and Acronyms

### 2.1 Definitions

#### 2.1.1 Terms Defined Elsewhere

This document uses the following terms defined elsewhere:

None.

#### 2.1.2 Terms Defined in this Document

This document defines the following terms:

None.

#### 2.1.3 Related terms

None.

## 2.2 Abbreviations and Acronyms

This document uses the following abbreviations and acronyms:

---

## 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

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

Listings of XACML schema and code listings appear like this.

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

- The prefix `xacml`: stands for the XACML 4.0 namespace.
- The prefix `ds`: stands for the W3C XML Signature namespace [DS].
- The prefix `xs`: stands for the W3C XML Schema namespace [XS].
- The prefix `xf`: stands for the XPath and XQuery Functions and Operators 3.1 specification namespace [XF].
- The prefix `xml`: stands for the XML namespace <https://www.w3.org/XML/1998/> namespace.

This specification uses the following typographical conventions in text: `<XACMLElement>`, `<ns:ForeignElement>`, `Attribute`, `Datatype`, `OtherCode`. Terms in ***bold-face italic*** are intended to have the meaning defined in Section 2.

### 3.3 Schema organization and namespaces

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

`urn:oasis:names:tc:xacml:4.0:core:schema`

---

## 4 Introduction (non-normative)

### 4.1 Requirements

The XML representation should be as aligned as possible with ACAL.

### 4.2 Abstraction Layer

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

Similarly, in the case where the resource to which access is requested is an XML document, the resource itself may be included in, or referenced by, the request context. Then, through the use of XPath expressions [XPath] in the policy, values in the resource may be included in the policy evaluation. The use of XPath expressions is not specified here but in the XPath Profile of ACAL.

### 4.3 Example Short Identifier Set

A set of Short Identifiers with the Id `urn:oasis:names:tc:acal:1.0:core:identifiers` is defined by ACAL in the XML Representation according to this specification, for the various identifiers assigned by ACAL, and provided attached to this profile. 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 defines an XML representation of 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"/>

</ShortIdSet>
```

Short Identifiers can reuse other Short Identifiers in their values, typically as prefix, for example, given the following short identifiers:

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

The following IdentifierType values are all equivalent and evaluate to the URI of the string data type:

```
"string"
"{string}"
"{xs}string"
```

## 4.4 Changes From the Previous Version

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

---

# 5 Syntax (normative, with the exception of the schema fragments)

The next sections describe the rules that SHALL be applied for mapping the [ACAL-Core-1.0] agnostic model (UML-based) to XML schema definitions for this XML representation (XACML). These rules have been applied to produce XACML's core XML schema in Annex D (also in the Core XML schema file accompanying this document) from [ACAL-Core-1.0] core model.

In all XSD definitions from now, the XACML core namespace `urn:oasis:names:tc:xacml:4.0:core:schema` is the default namespace.

## 5.1 Mapping ACAL primitive types

For each primitive type (stereotyped `<<primitive>>` or `<<enumeration>>`) in [ACAL-Core-1.0] model, apply the mapping rules in the next subsections.

### 5.1.1 Primitive types mapped to standard XSD data-types

The ACAL primitive types in the following table have a direct XML equivalent in the XSD standard, provided in the second column, and that is used for the XML representation (the expression `xs:foo` means the `foo` type in W3C XML schema standard):

Table 1: Mapping ACAL primitive types to standard XSD data-types

ACAL (UML)	XML schema
String	xs:string
Boolean	xs:boolean
Double	xs:double
Integer	xs:integer
NonNegativeInteger	xs:nonNegativeInteger
URI	xs:anyURI
Name	xs:Name

### 5.1.2 Restricted String types (UML stereotype <<restrictedString>>)

Each ACAL primitive type `FooType` with stereotype <<restrictedString>>, i.e. with a given `pattern` property set to a regular expression , is mapped to the following XSD definition:

```
<xs:simpleType name="FooType">
  <xs:restriction base="xs:string">
    <xs:pattern value="<REGEX without ^$ anchors>"/>
  </xs:restriction>
</xs:simpleType>
```

In XSD, the `^$` anchors are implicit as the pattern is always matched against the whole string, therefore they MUST be omitted from the pattern.

`FooType` may also have a (optional) `minLength` property set to a (strictly) positive integer `N`, in which case the XSD definition becomes:

```
<xs:simpleType name="FooType">
  <xs:restriction base="xs:string">
    <xs:pattern value="<REGEX without ^$ anchors>"/>
    <xs:minLength value="<N>" />
  </xs:restriction>
</xs:simpleType>
```

For example, ACAL `VersionType` translates to the following XSD definition:

```
<xs:simpleType name="VersionType">
  <xs:restriction base="xs:string">
    <xs:pattern value="(0|[1-9]\d*)(\.(0|[1-9]\d*)){0,3}" />
  </xs:restriction>
</xs:simpleType>
```

### 5.1.3 Enum types (UML stereotype <<enumeration>>)

Each ACAL enumerated type `FooType` (stereotyped <<enumeration>>) with enum values `V1`, `V2`, ... `Vn` is mapped to the following XSD definition:

```
<xs:simpleType name="FooType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="V1"/>
    <xs:enumeration value="V2"/>
    ...
    <xs:enumeration value="Vn"/>
  </xs:restriction>
</xs:simpleType>
```

For example, ACAL `DecisionType` translates to the following XSD definition:

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

## 5.2 Mapping complex ACAL types (UML stereotype <<complexType>>)

For each complex type (stereotyped <<complexType>>) in [ACAL-Core-1.0] model, apply the mapping rules in the next subsections.

The expression `xs:foo type` (*respectively element*) means the `foo` type (*respectively element*) defined in W3C XML schema standard.

### 5.2.1 AnyType mapping rule

The ACAL `AnyType` is mapped to the `xs:anyType` type in XML.

**WARNING:** for safety/security reasons, in production, ACAL implementers should add further restrictions to the XML schema and/or enforce security measures in the XML processor to mitigate possible security issues that may occur when allowing any XML type as input.

### 5.2.2 Single-use empty datatypes

There are complex ACAL datatypes playing the role of *Union* datatype, such as `CombinerInputType`, which are *empty* (no property and no class inheritance) and have as many subtypes as possible types in the *union*. These are usually used once for a property that can take one of multiple alternative types, such as `CombinerInput`.

Except the `AnyType` already addressed in the previous section, a single-use empty ACAL datatype `FooType` (i.e. used only for one property P in the whole ACAL model (e.g. `CombinerInput`) and not to be used in any ACAL extension) SHALL not be mapped to any dedicated XML type, element or attribute in the XSD. The mapping is done only at the level of property P (section 5.2.5) to a `xs:choice` of XSD elements corresponding to the subtypes of `FooType`.

### 5.2.3 ValueType mapping rules

As an exception, if the ACAL complex type is `ValueType` from [ACAL-Core-1.0] section 7.23, it is always mapped to the following XSD:

```
<xs:complexType name="ValueType" mixed="true">
  <xs:complexContent mixed="true">
    <xs:extension base="ExpressionType">
      <xs:sequence>
        <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="DataType" type="IdentifierType" use="optional"/>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

### 5.2.4 Default mapping rules for complex ACAL types (other than ValueType)

If a complex ACAL type `FooType` is not in the previous cases (section 5.2.1, 5.2.2 and 5.2.3), then:

1. If `FooType` is abstract (italicized title in the UML diagram), then:

- 1.1. If `FooType` is empty (no class inheritance, no property), then:
  - 1.1.1. If and only if `FooType` is used by more than one property in ACAL model (e.g. `ExpressionType`), then map it to the following XSD definition:

```
<xs:complexType name="FooType" abstract="true"/>
```
  - 1.1.2. Else (it is single-used) skip it as instructed in section 5.2.2.
- 1.2. Else (`FooType` is not empty, i.e. inherits a type and/or has at least one property):
  - 1.2.1. If `FooType` inherits from a Datatype `BarType` (Generalization relationship), then:
    - 1.2.1.1. If `BarType` is a single-use empty type as defined in section 5.2.2, `BarType` is not mapped to any XSD type (mapping is done only at the property level), therefore `FooType` (e.g. `PolicyType`) SHALL be mapped to the following XSD definition (not inheriting from `BarType`):

```
<xs:complexType name="FooType" abstract="true">
  ...property mapping rules (5.2.5) apply...
</xs:complexType>
```
    - 1.2.1.2. Else (`BarType` is not a single-use empty type, therefore is mapped to a dedicated XML type) if `FooType` has no property (e.g. `ForAnyType`), map it to the following XSD element definition (no need for an XSD type):

```
<xs:element name="Foo" type="BarType" substitutionGroup="Super" abstract="true"/>
```

where `Super` is the name of the `SuperType` element and `SuperType` is the closest datatype to `BarType` in `BarType`'s class hierarchy (including `BarType` itself) that is used as at least one property's datatype in ACAL model.
  - 1.2.1.3. Else (`FooType` has at least one property):

```
<xs:complexType name="FooType" abstract="true">
  <xs:complexContent>
    <xs:extension base="BarType">
      ...property mapping rules (5.2.5) apply...
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

- 1.2.2. Else (*FooType* does not inherit any type) map *FooType* to the following XSD definition:

```
<xs:complexType name="FooType" abstract="true">
  ...property mapping rules (5.2.5) apply...
</xs:complexType>
```

2. Else (*FooType* is not abstract) apply the same mapping rules as in the first case, except remove `abstract="true"` from the `xs:complexType` and `xs:element` definitions.
3. If *FooType* has a UML *link* (kind of *Association*) via its property *keyRefProp* to another type *BarType*'s property *keyProp* (e.g. a reference to a *BarType* instance via its identifier), then add a `<xs:keyRef>` element as defined in section 3.11.2 of [XS] (Identity-constraint Definitions), into the XML element declaration of the root object type (*RootObjectType*) enclosing the property(ies) of type *FooType* and *BarType* (created previously), as follows:

```
<xs:element name="RootObject" ...>
  ...
  <!-- The xs:key should already be declared from mapping BarType before FooType. -->
  <xs:key name="RootObject_Bar_keyProp">
    <xs:selector xpath="<XPath_to_Bar_element>" />
    <xs:field xpath="@keyProp" />
  </xs:key>

  <!-- New keyRef -->
  <xs:keyref name="RootObject_Foo_keyRefProp" refer="RootObject_Bar_keyProp">
    <xs:selector xpath="<XPath_to_Foo_element>" />
    <xs:field xpath="@keyRefProp" />
  </xs:keyref>
</xs:element>
```

where `<XPath_to_Foo_element>` is the XPath expression to select the *Foo* element of *FooType*.

4. If *FooType* has *Object-level constraints* as defined in section 7.1.1.2 of [ACAL-Core-1.0], implementers SHOULD apply the recommended mappings in the section 5.2.6, or they MAY also apply alternative implementation-specific mapping mechanisms as they see fit.

## 5.2.5 Property mapping rules

Inside the `xs:complexType` or `xs:extension` element created by the previous mapping rules, go the property mappings for each property *Prop* of the complex ACAL type. Let *PropType* be *Prop*'s datatype.

1. If *Prop* has special name `<any>` (and special type `AnyType`) in ACAL, then map the property to the following XSD definition (replace `{min}` with the lower bound of *Prop*'s multiplicity in the ACAL (UML) model, and `{max}` with the upper bound, using `unbounded` as equivalent for `*`):

```
<xs:sequence>
  <xs:any namespace="##any" processContents="lax" minOccurs="{min}" maxOccurs="{max}"/>
</xs:sequence>
```

2. Else if *PropType* is a primitive type, then:

- 2.1. Map *PropType* to a XSD type according to mapping rules of section 5.1. The obtained XSD type is referred to as `PropXsdType` in the next steps.
- 2.2. If *PropType* is other than (unrestricted) `String` (it may be a restricted `String` type as in section 5.1.2) AND single-valued (the upper bound of the multiplicity is 1), then map to an XML attribute as follows (if *Prop*'s multiplicity is `0..1`, replace `$use` with `optional`, else with `required`):

```
<xs:attribute name="Prop" type="PropXsdType" use="$use"/>
```

If *Prop* has a default value `$default`, then add the `default` attribute:

```
<xs:attribute name="Prop" type="PropXsdType" use="$use" default="$default"/>
```

- 2.3. Else (if *PropType* is `String` (unrestricted, i.e. not a derived type) or multi-valued), then:

Define a global element named *Prop* in XACML namespace:

```
<xs:element name="Prop" type="PropXsdType"/>
```

If this is the first property in the datatype, then create a `<xs:sequence>`.

Add an element reference to the *Prop* global element above inside the `<xs:sequence>` as follows (replace `$min` with the lower bound and `$max` with the upper bound of *Prop*'s multiplicity in the ACAL (UML) model, using `unbounded` as equivalent for `*`):

```

<xs:sequence>
    ...other properties mapped to elements...
    <xs:element ref="Prop" minOccurs="$min" maxOccurs="$max"/>
    ...other properties mapped to elements...
</xs:sequence>

```

- 2.4. If *Prop* is multivalued and has a *simple uniqueness constraint* as defined in section 7.1.1.1.1 of [ACAL-Core-1.0], i.e. a UML `unique` annotation, then add an `<xs:key>` element as defined in section 3.11.2 of [XS] (Identity-constraint Definitions), into the XML element declaration of the root object type enclosing *Prop* (created previously), as follows:

```

<xs:element name="RootAncestorOfProp" ...>
    ...
    <xs:key name="RootAncestorOfProp_Prop">
        <xs:selector xpath="<XPath_to_Prop>" />
        <xs:field xpath=". ." />
    </xs:key>
</xs:element>

```

where `<XPath_to_Prop>` is the XPath expression to select the *Prop* element from `RootAncestorOfProp` element.

3. Else if *PropType* is a *single-use empty type* as defined in Section 5.2.2 (e.g.`CombinerInputType`), i.e. an empty abstract type only used by *Prop* in the ACAL model, with subtypes *Sub1Type*, *Sub2Type*, etc., then for each subtype *SubXType* of *PropType*, apply the mapping rules in the previous section and this section to *SubXType*, and create an element `<xs:element name="SubX" type="SubXType" />` in XACML namespace if it does not already exist (`<xs:element name="SubX" type="SubXType" abstract="true"/>` if *SubXType* is abstract). Then *PropType* maps to the following XSD definition (replace `$min` with the lower bound of *Prop*'s multiplicity in the ACAL (UML) model, and `$max` with the upper bound, using `unbounded` as equivalent for `*`):

```

<xs:choice minOccurs="$min" maxOccurs="$max">
    <xs:element ref="Sub1"/>
    <xs:element ref="Sub2"/>
    ...
</xs:choice>

```

4. Else (*PropType* is not a single-use empty type), apply the mapping rules in the previous section and this section to *PropType*, then:

- 4.1. Define a global element named *Prop* of type *PropType* in XACML namespace if it does not already exist:

```
<xs:element name="Prop" type="PropType"/>
```

If *PropType* is abstract, add the `abstract="true"` attribute:

```
<xs:element name="Prop" type="PropType" abstract="true"/>
```

If *PropType* inherits from a type *BarType*, apply the mapping rules in the previous section and this section to *BarType* if the XSD type *BarType* is not already defined, then create a global element named *Bar* of type *BarType* in XACML namespace by re-applying this same rule 4.1 with *Bar* as *Prop* and *BarType* as *PropType*, if the element *Bar* does not already exist. Then add *Prop* element to *Bar* substitution group:

```
<xs:element name="Prop" type="PropType" substitutionGroup="Bar"/>
```

- 4.2. Map *Prop* to the following XSD definition (replace `$min` with the lower bound of *Prop*'s multiplicity in the ACAL (UML) model, and `$max` with the upper bound, using `unbounded` as equivalent for `*`):

```

<xs:sequence>
    ...other properties mapped to elements...
    <xs:element ref="Prop" minOccurs="$min" maxOccurs="$max"/>
    ...other properties mapped to elements...
</xs:sequence>

```

- 4.3. If *Prop* has a *mandatory-property-based uniqueness constraint* as defined in section 7.1.1.1.2 of [ACAL-Core-1.0], i.e. `self->isUnique(itemProp)`, then add an `<xs:key>` element as defined in section 3.11.2 of [XS] (Identity-constraint Definitions), into the XML element of the root object type enclosing *Prop* (created previously), as follows (such *itemProp* is always mapped to an XML attribute in this case):

```

<xs:element name="RootAncestorOfProp" ...>
    <xs:key name="RootAncestorOfProp_Prop">

```

```

<xs:selector xpath="<XPath_to_Prop>" />
<xs:field xpath="@itemProp" />
</xs:key>
</xs:element>

```

where *<XPath\_to\_Prop>* is the XPath expression to select the *Prop* element from *RootAncestorOfProp* element.

- 4.4 Else if *Prop* has a *optional-property-based uniqueness constraint* as defined in section 7.1.1.1.2 of [ACAL-Core-1.0], i.e. `self->select(itemProp <> null)->isUnique(itemProp)`, then add an *<xs:unique>* element as defined in section 3.11.2 of [XS] (Identity-constraint Definitions), into the XML element of the root object type enclosing *Prop* (created previously), as follows (such *itemProp* is always mapped to an XML attribute in this case):

```

<xs:element name="RootAncestorOfProp" ...>
  <xs:unique name="RootAncestorOfProp_Prop">
    <xs:selector xpath="<XPath_to_Prop>" />
    <xs:field xpath="@itemProp" />
  </xs:unique>
</xs:element>

```

where *<XPath\_to\_Prop>* is the XPath expression to select the *Prop* element from *RootAncestorOfProp* element.

- 4.5. Else if *Prop* has a *multi-property-based uniqueness constraint* as defined in section 7.1.1.1.2 of [ACAL-Core-1.0], i.e. `self->isUnique(Sequence{itemProp1, itemProp2, ...})`, then add an *<xs:unique>* element as defined in section 3.11.2 of [XS] (Identity-constraint Definitions), into the XML element of the root object type enclosing *Prop* (created previously), as follows (such *itemProp1*, *itemProp2*, etc. are always mapped to XML attributes in this case):

```

<xs:element name="RootAncestorOfProp" ...>
  <xs:unique name="RootAncestorOfProp_Prop">
    <xs:selector xpath="<XPath_to_Prop>" />
    <xs:field xpath="@itemProp1" />
    <xs:field xpath="@itemProp2" />
    ...
  </xs:unique>
</xs:element>

```

where *<XPath\_to\_Prop>* is the XPath expression to select the *Prop* element from *RootAncestorOfProp* element.

- 4.6. Else if *Prop* has other kinds of constraints defined in section 7.1.1.1.2 of [ACAL-Core-1.0] not mentioned previously, implementers SHOULD apply the recommended mappings in the next section, or they MAY also apply alternative implementation-specific mapping mechanisms as they see fit.

## 5.2.6 Mapping complex ACAL constraints (OCL)

This section is non-normative.

The following kinds of OCL-defined UML constraints in the [ACAL-Core-1.0] model do not have any equivalent in [XSD 1.0]:

- Some of the OCL-defined property-level constraints in [ACAL-Core-1.0] section 7.1.1.1.2:
  - Multivalued-property-based uniqueness constraint;
  - Value type uniqueness constraint.
- All object-level constraints in [ACAL-Core-1.0] section 7.1.1.1.2.

This document does not mandate a particular method to map and/or implement these more complex constraints, and leaves this part of the specification implementation-defined. Instead, this document provides only an implementation guidance that recommends to possible options described in the next sections:

- Option 1: [XSD 1.1] schema with XML Schema 1.1 *assertions*;
- Option 2: [XSD 1.0] schema with ISO Schematron rules.

**5.2.6.1 Option 1: XML Schema 1.1 assertions** Implementations supporting W3C XML Schema 1.1 standard SHOULD use XSD 1.1 assertions as defined in section 3.13 of XSD 1.1 standard to implement the aforementioned constraints. The general mapping rule consists for each constraint specified on a ACAL Datatype *FooType* or one of its properties, to define an equivalent XSD 1.1 assertion `<xs:assert test="expression"/>`

- where *expression* is the XPath equivalent of the OCL constraint expression - inside the `xs:complexType` definition of `FooType`.

The equivalent XSD 1.1 assertions are already defined in the *Core XML schema* accompanying this document (`<xs:assert>` elements). However, certain generic assertions may be problematic for certain XSD 1.1 / XPath engines or use cases, in which case alternative assertions are suggested in XML comments before or after each possibly problematic `<xs:assert>` element currently in use in the schema. Implementers SHOULD review these before use.

Moreover, implementers MAY choose to replace some of the assertions with possibly more optimal implementations as long as they provide equivalent constraint enforcement.

For information only, the following table suggests generic mappings for some of the aforementioned complex UML constraints used in ACAL model:

**Table 2:** ACAL/UML constraints mapped to [XSD 1.1] assertions

ACAL constraint's OCL expression	XSD 1.1 assertion's XPath expression
<code>X or Y</code> ( <i>X, Y can be any of the predicates below</i> )	<code>X or Y</code>
<code>prop &lt;&gt; null</code> ( <i>prop is single-valued</i> )	<code>prop</code> ( <i>the prop element occurs</i> )
<code>prop = null</code> ( <i>prop is single-valued</i> )	<code>not(prop)</code> ( <i>no prop element occurs</i> )
<code>prop-&gt;notEmpty()</code> ( <i>prop is multivalued</i> )	<code>prop</code> ( <i>at least one prop element occurs</i> )

You may find further guidance about XSD-1.1-based validation on OASIS XACML TC's code repository.

**5.2.6.2 Option 2: Schematron rules** Due to the current general lack of adoption of XSD 1.1 (and limited availability of implementations), implementations only supporting XSD 1.0 MAY achieve the same result by combining the following:

- An XSD 1.0 version of the *Core XML Schema*, obtained by filtering out all the schema elements with attribute `vc:minVersion="1.1"` (according to [XSV]) from the XSD 1.1 schema (e.g. with an [XSLT] stylesheet);
- ISO Schematron rules provided with this document (file with `.sch` extension), that implement the complex OCL constraints with similar - if not the same - XPath expressions as the XSD 1.1 assertions in the previous Option 1. Therefore, like XSD 1.1 assertions, certain generic Schematron assertions may be problematic for certain XSLT / XPath engines or use cases, in which case alternative assertions are suggested in XML comments before or after each possibly problematic `<assert>` element currently in use in the schema. Implementers SHOULD review these before use. Moreover, implementers MAY choose to replace some of the assertions with possibly more optimal implementations as long as they provide equivalent constraint enforcement. Each Schematron `pattern` element has the corresponding [ACAL-Core-1.0] model's UML constraint (with OCL expression) reminded in the child `title` element. You may find further guidance about Schematron-based validation on OASIS XACML TC's code repository.

### 5.3 Content Types and Body representations (optional)

Although this specification defines an XML representation, both XML and non-XML data may be represented in a `<Content>` element (corresponding to an ACAL `ContentType` object). This specification defines the following `Content` types in order to support ACAL Profiles with AttributeSelector and/or DataType extensions based on such Content (e.g. XPath and JSONPath Profiles):

- XML document:
  - `MediaType` attribute SHALL be set to `application/xml` (default value);
  - `Encoding` attribute unused;
  - `Body` element has only a single child element that is the XML document itself.
- JSON object (*note that a JSON array can be wrapped in a JSON object if there is a need to support JSON arrays*):
  - `MediaType` attribute SHALL be `application/json`;
  - `Encoding` attribute unused;
  - `Body` element has only a text node that contains the JSON object, either escaped to fit in a simple XML string, or unescaped in a XML CDATA section as follows:

```
<Body>
<! [CDATA[
  ... JSON object ...
]]>
</Body>
```

The implementation SHALL support a given content type in this list if and only if there is an ACAL Profile that makes it mandatory (refer to the Profile's specification for more information).

---

## 6 Safety, Security and Privacy Considerations (non-normative)

This section identifies possible security and privacy compromise scenarios that should be considered when implementing this profile. 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.

### 6.1 Threat model

Refer to [ACAL-Core-1.0] section 11.1.

### 6.2 Safeguards

Refer to [ACAL-Core-1.0] section 11.2 for general considerations.

#### 6.2.1 Policy confidentiality

Where the policy is represented in XML Representation defined by this profile, the *XML Encryption Syntax and Processing Candidate Recommendation* from W3C can be used to encrypt all or parts of an XML document. This specification is recommended for use with XACML.

#### 6.2.2 Policy integrity

The selection of the appropriate mechanisms is left to the implementers. However, when **policy** is distributed between organizations to be acted on at a later time, or when the **policy** travels with the protected **resource**, it would be useful to sign the **policy**. In these cases and when the XML representation of policies (according to this profile) is used, the *XML Signature Syntax and Processing standard* from W3C is recommended to be used with XACML.

---

## 7 Conformance

### 7.1 Introduction

The XACML specification addresses the following aspect of conformance:

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

### 7.2 Conformance tables

This section lists those portions of the specification that MUST be included in an implementation of a **PDP** that claims to conform to XACML 4.0. 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 5 and Annex C where they apply to implemented items in the following tables.

### 7.2.1 Schema elements

The implementation MUST support those schema elements in the XACML core namespace ([urn:oasis:names:tc:xacml:4.0:core-schema#](http://urn:oasis:names:tc:xacml:4.0:core-schema#)) that are marked M.

Element name	M/O
Apply	M
ApplicablePolicyReference	O
Attribute	M
AttributeAssignment	M
AttributeAssignmentExpression	M
AttributeDesignator	M
AttributeSelector	O
Bundle	O
Category	M
Condition	M
Content	O
Description	M
EntityAttributeDesignator	O
EntityAttributeSelector	O
Expression	M
ForAll	O
ForAny	O
Function	M
Map	O
MissingAttributeDetail	M
MultiRequests	O
Notice	M
NoticeExpression	M
Policy	M
PolicyDefaults	O
PolicyReference	M
PolicyIssuer	O
PolicyPatternMatchReference	O
Request	M
RequestAttribute	M
RequestDefaults	O
RequestEntity	M
RequestEntityReference	O
RequestReference	O
Response	M
Result	M
ResultEntity	M
Rule	M
Select	O
SharedVariableDefinition	O
SharedVariableReference	O
ShortId	M
ShortIdSet	M
ShortIdSetReference	M
Status	M
StatusCode	M
StatusDetail	O
StatusMessage	O
Target	M
Value	M
VariableDefinition	M
VariableReference	M

# 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 (<https://www.oasis-open.org/committees/xacml/ipr.php>).

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.

[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.

[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.

[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]

RFC2256, 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]

RFC2798, 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>

[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.

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <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>

[RFC8174]

Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[Schematron]

ISO/IEC 19757-3:2025, Information technology — Document Schema Definition Languages (DSDL) - Part 3: Rule-based validation using Schematron, Edition 4, September 2025. Online: <https://www.iso.org/standard/85625.html>.

[UAX15]

Mark Davis, Martin Dürst, Unicode Standard Annex #15: Unicode Normalization Forms, Unicode 5.1, <https://unicode.org/reports/tr15/>

[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>

[XACMLv1.0]

OASIS Standard, Extensible access control markup language (XACML) Version 1.0. 18 February 2003, <https://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf>

[XACMLv1.1]

OASIS Committee Specification, Extensible access control markup language (XACML) Version 1.1. 7 August 2003, <https://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf>

[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/>

[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/>

[XPath]

XML Path Language (XPath) 3.1, W3C Recommendation 21 March 2017, <https://www.w3.org/TR/xpath-31/>

[XPathFunc]

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 1.0, parts 1 and 2. Available at: <https://www.w3.org/TR/xmlschema-1/> and <https://www.w3.org/TR/xmlschema-2/>

[XS11]

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/>

[XSLT]

XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999, <https://www.w3.org/TR/xslt/>

[XSV]

W3C, XML Schema Versioning namespace, August 2007. Online: <https://www.w3.org/2007/XMLSchema-versioning/>

## 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.

[Sloman94]

Sloman, M. Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Volume 2, part 4. Plenum Press. 1994.

---

## Annex C XACML identifiers (normative)

This section defines standard identifiers for commonly used entities.

### C.1 XACML namespaces

XACML is defined using this identifier:

urn:oasis:names:tc:xacml:4.0:core:schema

---

## Annex D XML Schema (normative)

This section includes the XML Schema for the XACML syntax defined in this specification, more particularly in section 5 (i.e. obtained by applying the ACAL-to-XML mapping rules):

```
<xs:schema xmlns:xacml="urn:oasis:names:tc:xacml:4.0:core:schema" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:vc="http://www.w3.org/ns/vc#">
  <xs:annotation>
    <xs:documentation>
      XACML v4.0 (XML Representation of ACAL Version 1.0) Core schema.
      Copyright (c) OASIS Open 2026. All Rights Reserved.
      For more license and copyright information,
      and complete status, please see Annex A of XACML v4.0 (XML Representation of ACAL
      Version 1.0) specification, which contains the License, Document Status and Notices.
    </xs:documentation>
  </xs:annotation>
  <xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="https://www.w3.org/2001/xml.xsd"/>
  <xs:element name="Request" type="xacml:RequestType">
    <xs:key name="Request_ShortIdSetReference">
      <xs:selector xpath="xacml:ShortIdSetReference"/>
      <xs:field xpath=".//>
    </xs:key>
    <xs:unique name="Request_RequestEntity_Id">
      <xs:selector xpath="xacml:RequestEntity"/>
      <xs:field xpath="@Id"/>
    </xs:unique>
    <xs:keyref name="Request_RequestEntityReference" refer="xacml:Request_RequestEntity_Id">
      <xs:selector xpath="xacml:MultiRequests/xacml:RequestReference/xacml:RequestEntityReference"/>
      <xs:field xpath="@Id"/>
    </xs:keyref>
  </xs:element>
  <xs:complexType name="RequestType">
    <xs:sequence>
      <xs:element ref="xacml:ShortIdSetReference" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="xacml:RequestDefaults" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="xacml:RequestEntity" maxOccurs="unbounded"/>
      <xs:element ref="xacml:MultiRequests" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="ReturnPolicyIdList" type="xs:boolean" use="optional" default="false"/>
  </xs:complexType>

```

```

<xs:attribute name="CombinedDecision" type="xs:boolean" use="optional" default="false"/>
<xs:assert vc:minVersion="1.1" test="every $elt in element(*, xacml:DefaultsType) satisfies (every $following in $elt/following-sibling::*)[1] = $elt">
</xs:complexType>
<xs:element name="RequestDefaults" type="xacml:DefaultsType" abstract="true"/>
<xs:element name="Response" type="xacml:ResponseType">
  <xs:key name="Response_ShortIdSetReference">
    <xs:selector xpath="xacml:ShortIdSetReference"/>
    <xs:field xpath=". "/>
  </xs:key>
</xs:element>
<xs:complexType name="ResponseType">
  <xs:sequence>
    <xs:element ref="xacml:ShortIdSetReference" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:Result" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="MediaType">
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Za-z0-9][A-Za-z0-9!#$&lt;!--_+]{0,63}[A-Za-z0-9!#$&lt;!--_+]{0,63}">
      <xs:annotation>
        <xs:documentation xml:lang="en">Pattern based on the ABNF syntax from section
          4.2 of RFC 6838 (with the recommendation that &lt;type-name&gt; and
          &lt;subtype-name&gt; SHOULD be limited to 64 characters).
          The standard MediaType values defined in XACML core specification should be used
          - 'application/xml' for XML document (default);
          - 'application/json' for JSON object.
          For other cases, you may use
        </xs:documentation>
      </xs:annotation>
    </xs:pattern>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ContentEncodingType">
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-z0-9]+(-[a-z0-9]+)*">
      <xs:annotation>
        <xs:documentation xml:lang="en">
          No special encoding is needed for XML/JSON data.
          However, if other (custom) content types are used and require specific encoding
          the standard identifiers specified in [Annex D.11](#d11-content-encodings)
          SHOULD be used whenever applicable, e.g. 'base64' or else one of the standard [IANA]
          Encodings* names registered at
          IANA](https://www.iana.org/assignments/transfer-encodings/transfer-encodings.xhtml)
          Or else, if a custom content encoding is really needed, as suggested by
          RFC 2045 Section 6.3, define a new one prefixed with `x-`,
          e.g. `x-my-new-encoding` .
        </xs:documentation>
      </xs:annotation>
    </xs:pattern>
  </xs:restriction>
</xs:simpleType>
<xs:element name="Body" type="xs:anyType">
  <xs:annotation>
    <xs:documentation>
      Content body. For safety/security reasons, in production, ACAL implementers shall add filtering
      to prevent possible security issues when allowing any XML content like this.
    </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="Content" type="xacml:ContentType"/>
<xs:complexType name="ContentType">
  <xs:sequence>
    <xs:element ref="xacml:Body"/>
  </xs:sequence>
  <xs:attribute name="MediaType" type="xacml:MediaType" use="optional" default="application/xml"/>
  <xs:attribute name="Encoding" type="xacml:ContentEncodingType" use="optional"/>
</xs:complexType>
<xs:element name="Result" type="xacml:ResultType">
  <xs:key name="Result_ResultEntity_Category">
    <xs:selector xpath="xacml:ResultEntity"/>
    <xs:field xpath="@Category"/>
  </xs:key>
  <xs:key name="Result_ApplicablePolicyReference">
    <xs:selector xpath="xacml:ApplicablePolicyReference"/>
  </xs:key>
</xs:element>

```

```

<xs:field xpath="@Id"/>
</xs:key>
</xs:element>
<xs:complexType name="ResultType">
  <xs:sequence>
    <xs:element ref="xacml>Status" minOccurs="0"/>
    <xs:element ref="xacml:Notice" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:ResultEntity" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:ApplicablePolicyReference" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="Decision" type="xacml:DecisionType" use="required"/>
</xs:complexType>
<xs:element name="ResultEntity" type="xacml:ResultEntityType">
  <xs:key name="ResultEntity_Attribute_AttributeId">
    <xs:selector xpath="xacml:Attribute"/>
    <xs:field xpath="@AttributeId"/>
  </xs:key>
</xs:element>
<xs:complexType name="ResultEntityType">
  <xs:sequence>
    <xs:element ref="xacml:Attribute" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="Category" type="xacml:IdentifierType" use="required"/>
  <xs:attribute name="Id" type="xacml:LocalIdentifierType" use="optional"/>
</xs:complexType>
<xs:element name="ApplicablePolicyReference" type="xacml:ExactMatchIdReferenceType">
  <xs:annotation>
    <xs:documentation xml:lang="en">Policy reference to an applicable policy used in a Result</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:complexType name="ExactMatchIdReferenceType">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      IdReferenceType with a fixed Version (to be used for ApplicablePolicyReferences in a Result), a
      Version pattern matching.
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="xacml:IdReferenceType">
      <xs:attribute name="Version" type="xacml:VersionType" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:simpleType name="DecisionType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Permit"/>
    <xs:enumeration value="Deny"/>
    <xs:enumeration value="Indeterminate"/>
    <xs:enumeration value="NotApplicable"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="Status" type="xacml:StatusType"/>
<xs:complexType name="StatusType">
  <xs:sequence>
    <xs:element ref="xacml:StatusCode"/>
    <xs:element ref="xacml:StatusMessage" minOccurs="0"/>
    <xs:element ref="xacml:StatusDetail" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="StatusCode" type="xacml:StatusCodeType"/>
<xs:complexType name="StatusCodeType">
  <xs:sequence>
    <xs:element ref="xacml:StatusCode" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="Value" type="xacml:IdentifierType" use="required"/>
</xs:complexType>
<xs:element name="StatusMessage" type="xs:string"/>
<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>
<xs:element name="MissingAttributeDetail" type="xacml:MissingAttributeDetailType"/>
<xs:complexType name="MissingAttributeDetailType">
  <xs:sequence>

```

```

<xs:element ref="xacml:Value" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="Category" type="xacml:IdentifierType" use="required"/>
<xs:attribute name="AttributeId" type="xacml:IdentifierType" use="required"/>
<xs:attribute name="DataType" type="xacml:IdentifierType" use="required"/>
<xs:attribute name="Issuer" type="xs:Name" use="optional"/>
<xs:assert vc:minVersion="1.1" test="not(xacml:Value/@ DataType)"/>
</xs:complexType>
<xs:element name="RequestEntity" type="xacml:RequestEntityType">
  <xs:key name="RequestEntity_RequestAttribute_AttributeId">
    <xs:selector xpath="xacml:RequestAttribute"/>
    <xs:field xpath="@AttributeId"/>
  </xs:key>
</xs:element>
<xs:complexType name="RequestEntityType">
  <xs:sequence>
    <xs:element ref="xacml:Content" minOccurs="0"/>
    <xs:element ref="xacml:RequestAttribute" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="Category" type="xacml:IdentifierType" use="required"/>
  <xs:attribute name="Id" type="xacml:LocalIdentifierType" use="optional"/>
</xs:complexType>
<xs:element name="Attribute" type="xacml:AttributeType"/>
<xs:complexType name="AttributeType">
  <xs:sequence>
    <xs:element ref="xacml:Value" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="AttributeId" type="xacml:IdentifierType" use="required"/>
  <xs:attribute name="Issuer" type="xs:Name" use="optional"/>
  <xs:attribute name="DataType" type="xacml:IdentifierType" use="optional" default="urn:oasis:names:tc:acal:1.0:data-type:string">
    <xs:assert vc:minVersion="1.1" test="not(xacml:Value/@ DataType)"/>
  </xs:attribute>
</xs:complexType>
<xs:element name="RequestAttribute" type="xacml:RequestAttributeType"/>
<xs:complexType name="RequestAttributeType">
  <xs:complexContent mixed="false">
    <xs:extension base="xacml:AttributeType">
      <xs:attribute name="IncludeInResult" type="xs:boolean" use="optional" default="false"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="MultiRequests" type="xacml:MultiRequestsType"/>
<xs:complexType name="MultiRequestsType">
  <xs:sequence>
    <xs:element ref="xacml:RequestReference" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:assert vc:minVersion="1.1" test="every $elt in xacml:RequestReference satisfies (every $following in $elt/following-sibl
    <xs:annotation>
      <xs:documentation xml:lang="en">
        ACAL constraint on RequestReference property: {OCL}
        self-&gt;isUnique(RequestEntityReference-&gt;collect(Id)-&gt;asSet())
      </xs:documentation>
    </xs:annotation>
  </xs:assert>
</xs:complexType>
<xs:element name="RequestReference" type="xacml:RequestReferenceType"/>
<xs:complexType name="RequestReferenceType">
  <xs:sequence>
    <xs:element ref="xacml:RequestEntityReference" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="RequestEntityReference" type="xacml:RequestEntityReferenceType"/>
<xs:complexType name="RequestEntityReferenceType">
  <xs:attribute name="Id" type="xacml:LocalIdentifierType" use="required"/>
</xs:complexType>
<xs:element name="Notice" type="xacml:NoticeType">
  <xs:unique name="Notice_AttributeAssignment_AttributeId-Category">
    <xs:selector xpath="xacml:AttributeAssignment"/>
    <xs:field xpath="@AttributeId"/>
    <xs:field xpath="@Category"/>
  </xs:unique>
</xs:element>
<xs:complexType name="NoticeType">
  <xs:sequence>
    <xs:element ref="xacml:AttributeAssignment" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="Id" type="xacml:IdentifierType" use="required"/>

```

```

<xs:attribute name="IsObligation" type="xs:boolean" use="optional" default="false"/>
</xs:complexType>
<xs:element name="AttributeAssignment" type="xacml:AttributeAssignmentType"/>
<xs:complexType name="AttributeAssignmentType">
  <xs:complexContent>
    <xs:extension base="xacml:AttributeType">
      <xs:attribute name="Category" type="xacml:IdentifierType" use="optional"/>
      <xs:assert vc:minVersion="1.1" test="not(xacml:Value/@DataType)" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="NoticeExpression" type="xacml:NoticeExpressionType">
  <xs:unique name="NoticeExpression_AttributeAssignmentExpression_AttributeId-Category">
    <xs:selector xpath="xacml:AttributeAssignmentExpression"/>
    <xs:field xpath="@AttributeId"/>
    <xs:field xpath="@Category"/>
  </xs:unique>
</xs:element>
<xs:complexType name="NoticeExpressionType">
  <xs:sequence>
    <xs:element ref="xacml:Condition" minOccurs="0"/>
    <xs:element ref="xacml:AttributeAssignmentExpression" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="Id" type="xacml:IdentifierType" use="required"/>
  <xs:attribute name="IsObligation" type="xs:boolean" use="optional"/>
  <xs:attribute name="AppliesTo" type="xacml:EffectType" use="optional">
    <xs:annotation>
      <xs:documentation xml:lang="en">
        Undefined AppliesTo means it applies to both Permit and Deny.
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:element name="AttributeAssignmentExpression" type="xacml:AttributeAssignmentExpressionType"/>
<xs:complexType name="AttributeAssignmentExpressionType">
  <xs:sequence>
    <xs:element ref="xacml:Expression">
      <xs:annotation>
        <xs:documentation xml:lang="en">
          If the Expression is a Value (literal) without a defined DataType, it is
          assumed to be String (urn:oasis:names:tc:acal:1.0:data-type:string) by default.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="AttributeId" type="xacml:IdentifierType" use="required"/>
  <xs:attribute name="Category" type="xacml:IdentifierType" use="optional"/>
  <xs:attribute name="Issuer" type="xs:Name" use="optional"/>
</xs:complexType>
<xs:simpleType name="EffectType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Permit"/>
    <xs:enumeration value="Deny"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="PolicyIssuer" type="xacml:EntityType">
  <xs:unique name="PolicyIssuer_Attribute_AttributeId-Issuer">
    <xs:selector xpath="xacml:Attribute"/>
    <xs:field xpath="@AttributeId"/>
    <xs:field xpath="@Issuer"/>
  </xs:unique>
</xs:element>
<xs:complexType name="EntityType">
  <xs:sequence>
    <xs:element ref="xacml:Content" minOccurs="0"/>
    <xs:element ref="xacml:Attribute" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:assert vc:minVersion="1.1" test="xacml:Content or xacml:Attribute"/>
</xs:complexType>
<xs:simpleType name="LocalIdentifierType">
  <xs:restriction base="xs:string">
    <xs:pattern value="_*[A-Za-z][A-Za-z_0-9]*([-_.]*[A-Za-z_0-9]*)*"/>
    <xs:annotation>
      <xs:documentation xml:lang="en">
        Pattern agreed with ALFA working group, except the hyphen is allowed as
        separator in addition to the dot:
      </xs:documentation>
    </xs:annotation>
  </xs:restriction>
</xs:simpleType>

```

```

        &lt;name&gt;.&lt;name&gt;.&lt;name&gt;...
        or
        &lt;name&gt;-&lt;name&gt;-&lt;name&gt;...
        where &lt;name&gt; is at least a letter possibly preceded by underscore(s) and/
        letter(s)/digit(s)/underscore(s)
    </xs:documentation>
</xs:annotation>
</xs:pattern>
</xs:restriction>
</xs:simpleType>
<xs:element name="PolicyPatternMatchReference" type="xacml:PatternMatchIdReferenceType"/>
<xs:complexType name="PatternMatchIdReferenceType">
    <xs:complexContent>
        <xs:extension base="xacml:IdReferenceType">
            <xs:attribute name="Version" type="xacml:VersionMatchType" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:element name="PolicyReference" type="xacml:PolicyReferenceType"/>
<xs:complexType name="PolicyReferenceType">
    <xs:complexContent>
        <xs:extension base="xacml:PatternMatchIdReferenceType">
            <xs:sequence>
                <xs:element ref="xacml:Expression" minOccurs="0" maxOccurs="unbounded">
                    <xs:annotation>
                        <xs:documentation xml:lang="en">
                            Optional argument(s) to the referenced
                            Policy. The arguments SHALL match the
                            Policy's Parameter definitions in the same order
                            of declaration.
                            If one of the argument Expressions is a Value element, it SHALL
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:sequence>
            <xs:assert vc:minVersion="1.1" test="not(xacml:Value/@DataType)" />
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="DefaultsType" abstract="true">
    <xs:annotation>
        <xs:documentation xml:lang="en">
            ACAL Profiles may extend DefaultsType to support both PolicyDefaults and RequestDefault
            &lt;xs:complexType name="MyProfileDefaultsType"&gt;
                &lt;xs:complexContent&gt;
                    &lt;xs:extension base="xacml:DefaultsType"&gt;
                        ...My common Defaults...
                    &lt;/xs:extension&gt;
                &lt;/xs:complexContent&gt;
            &lt;/xs:complexType&gt;
        </xs:documentation>
    </xs:annotation>
    <xs:element name="MyProfilePolicyDefaults" type="MyProfileDefaultsType"
        substitutionGroup="xacml:PolicyDefaults"/&gt;
    <xs:element name="MyProfileRequestDefaults" type="MyProfileDefaultsType"
        substitutionGroup="xacml:RequestDefaults"/&gt;

```

In XSD 1.0:

```

        &lt;xs:element name="MyProfilePolicyDefaults" type="MyProfileDefaultsType"
            substitutionGroup="xacml:PolicyDefaults"/&gt;
        &lt;xs:element name="MyProfileRequestDefaults" type="MyProfileDefaultsType"
            substitutionGroup="xacml:RequestDefaults"/&gt;

```

In XSD 1.1, substitutionGroup on multiple elements is allowed, therefore we can simplify:

```

        &lt;xs:element name="MyProfileDefaults" type="MyProfileDefaultsType"
            substitutionGroup="xacml:PolicyDefaults xacml:RequestDefaults"/&gt;
    </xs:documentation>
</xs:complexType>
<xs:element name="PolicyDefaults" type="xacml:DefaultsType" abstract="true"/>
<xs:complexType name="IdReferenceType" abstract="true">
    <xs:attribute name="Id" type="xs:anyURI" use="required"/>
</xs:complexType>
<xs:simpleType name="VersionType">
    <xs:restriction base="xs:string">
        <xs:pattern value="(0|[1-9]\d*)(\.(0|[1-9]\d*)){0,3}"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="VersionMatchType">
    <xs:restriction base="xs:string">
        <xs:pattern value="(0|[1-9]\d*|\*)(\.(0|[1-9]\d*|\*|\+)){0,3}"/>
    </xs:restriction>

```

```

</xs:restriction>
</xs:simpleType>
<xs:element name="Parameter" type="xacml:ParameterType"/>
<xs:complexType name="ParameterType">
  <xs:sequence>
    <xs:element ref="xacml:Description" minOccurs="0">
      <xs:annotation>
        <xs:documentation xml:lang="en">
          Parameter description
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element ref="xacml:Expression" minOccurs="0">
      <xs:annotation>
        <xs:documentation xml:lang="en">
          Default value expression (optional), i.e. that is used as default argument if no
          value is passed to this parameter from the policy reference.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="Name" type="xacml:LocalIdentifierType" use="required"/>
  <xs:attribute name="DataType" type="xacml:IdentifierType" use="optional" default="urn:oasis:names:tc:acal:1.0:data-type:string"/>
  <xs:attribute name="IsBag" type="xs:boolean" use="optional" default="false"/>
  <xs:assert vc:minVersion="1.1" test="not(xacml:Value/@DataType)"/>
</xs:complexType>
<xs:element name="Policy" type="xacml:PolicyType">
  <xs:key name="Policy_ShortIdSetReference">
    <xs:selector xpath="xacml:ShortIdSetReference"/>
    <xs:field xpath=".//>">
  </xs:key>
  <xs:key name="Policy_Parameter_Name">
    <xs:selector xpath="xacml:Parameter"/>
    <xs:field xpath="@Name"/>
  </xs:key>
  <xs:key name="Policy_VariableDefinition_VariableId">
    <xs:selector xpath="xacml:VariableDefinition"/>
    <xs:field xpath="@VariableId"/>
  </xs:key>
  <xs:key name="Policy_PolicyId">
    <xs:selector xpath="xacml:Policy"/>
    <xs:field xpath="@PolicyId"/>
  </xs:key>
  <xs:key name="Policy_Rule_Id">
    <xs:selector xpath="xacml:Rule"/>
    <xs:field xpath="@Id"/>
  </xs:key>
</xs:element>
<xs:complexType name="PolicyType">
  <xs:sequence>
    <xs:element ref="xacml:ShortIdSetReference" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:Description" minOccurs="0"/>
    <xs:element ref="xacml:PolicyIssuer" minOccurs="0"/>
    <xs:element ref="xacml:PolicyDefaults" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:Parameter" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:VariableDefinition" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:Target" minOccurs="0"/>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="xacml:Policy"/>
      <xs:element ref="xacml:Rule"/>
      <xs:element ref="xacml:PolicyReference"/>
    </xs:choice>
    <xs:element ref="xacml:NoticeExpression" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
  <xs:attribute name="Version" type="xacml:VersionType" use="required"/>
  <xs:attribute name="CombiningAlgId" type="xacml:IdentifierType" use="required"/>
  <xs:attribute name="MaxDelegationDepth" type="xs:nonNegativeInteger" use="optional"/>
  <xs:assert vc:minVersion="1.1" test="every $elt in element(*, xacml:DefaultsType) satisfies (every $following in $elt/following-sibling::*)[1] = $elt">
    <xs:annotation>
      <xs:documentation xml:lang="en">
        The ACAL constraint on PolicyDefaults property is similar to the
        RequestDefaults Property.
      </xs:documentation>
    </xs:annotation>
  </xs:assert>

```

```

</xs:complexType>
<xs:element name="Description" type="xs:string"/>
<xs:element name="Rule" type="xacml:RuleType">
  <xs:key name="Rule_VariableDefinition_VariableId">
    <xs:selector xpath="xacml:VariableDefinition"/>
    <xs:field xpath="@VariableId"/>
  </xs:key>
</xs:element>
<xs:complexType name="RuleType">
  <xs:sequence>
    <xs:element ref="xacml:Description" minOccurs="0"/>
    <xs:element ref="xacml:VariableDefinition" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:Condition" minOccurs="0"/>
    <xs:element ref="xacml:NoticeExpression" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="Id" type="xacml:LocalIdentifierType" use="required"/>
  <xs:attribute name="Effect" type="xacml:EffectType" use="required"/>
</xs:complexType>
<xs:element name="Target" type="xacml:BooleanExpressionType"/>
<xs:element name="VariableDefinition" type="xacml:VariableDefinitionType"/>
<xs:complexType name="VariableDefinitionType">
  <xs:sequence>
    <xs:element ref="xacml:Expression">
      <xs:annotation>
        <xs:documentation xml:lang="en">
          If the Expression is a Value (literal) without a defined DataType
          identifier, it is assumed to be String (urn:oasis:names:tc:acal:1.0:data-type:s
          default.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="VariableId" type="xacml:LocalIdentifierType" use="required"/>
</xs:complexType>
<xs:element name="Expression" type="xacml:ExpressionType" abstract="true"/>
<xs:complexType name="ExpressionType" abstract="true"/>
<xs:element name="NonLiteralExpression" type="xacml:NonLiteralExpressionType" abstract="true" substitutionGroup="xacml:Expres
<xs:complexType name="NonLiteralExpressionType" abstract="true">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      NonLiteralExpressionType is an ExpressionType that is neither ValueType or
      FunctionType.
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="xacml:ExpressionType"/>
  </xs:complexContent>
</xs:complexType>
<xs:element name="VariableReference" type="xacml:VariableReferenceType" substitutionGroup="xacml:NonLiteralExpression"/>
<xs:complexType name="VariableReferenceType">
  <xs:complexContent>
    <xs:extension base="xacml:NonLiteralExpressionType">
      <xs:attribute name="VariableId" type="xacml:LocalIdentifierType" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:simpleType name="AttributeSelectorPathType">
  <xs:restriction base="xs:string">
    <xs:pattern value="\S(.*\S)?">
      <xs:annotation>
        <xs:documentation xml:lang="en">Non-empty string without leading/trailing whitespace</xs:documentation>
      </xs:annotation>
    </xs:pattern>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="BaseAttributeSelectorType" abstract="true">
  <xs:complexContent>
    <xs:extension base="xacml:NonLiteralExpressionType">
      <xs:attribute name="Path" type="xacml:AttributeSelectorPathType" use="required"/>
      <xs:attribute name="DataType" type="xacml:IdentifierType" use="optional" default="urn:oasis:names:tc:acal:1.0:data-type:>
      <xs:attribute name="MustBePresent" type="xs:boolean" use="optional" default="false"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="AttributeSelector" type="xacml:AttributeSelectorType" abstract="true" substitutionGroup="xacml:NonLiteralExp
<xs:complexType name="AttributeSelectorType" abstract="true">

```

```

<xs:complexContent>
  <xs:extension base="xacml:BaseAttributeSelectorType">
    <xs:attribute name="Category" type="xacml:IdentifierType" use="required"/>
  </xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:element name="EntityAttributeSelector" type="xacml:EntityAttributeSelectorType" abstract="true" substitutionGroup="xacml:EntityAttributeSelectorType">
<xs:complexType name="EntityAttributeSelectorType" abstract="true">
  <xs:complexContent>
    <xs:extension base="xacml:BaseAttributeSelectorType">
      <xs:sequence>
        <xs:element ref="xacml:Expression"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
</xs:element>
<xs:complexType name="NamedAttributeDesignatorType" abstract="true">
  <xs:complexContent>
    <xs:extension base="xacml:NonLiteralExpressionType">
      <xs:attribute name="AttributeId" type="xacml:IdentifierType" use="required"/>
      <xs:attribute name="DataType" type="xacml:IdentifierType" use="optional" default="urn:oasis:names:tc:acal:1.0:data-type"/>
      <xs:attribute name="Issuer" type="xs:Name" use="optional"/>
      <xs:attribute name="MustBePresent" type="xs:boolean" use="optional" default="false"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="AttributeDesignator" type="xacml:AttributeDesignatorType" substitutionGroup="xacml:NonLiteralExpressionType"/>
<xs:complexType name="AttributeDesignatorType">
  <xs:complexContent>
    <xs:extension base="xacml:NamedAttributeDesignatorType">
      <xs:attribute name="Category" type="xacml:IdentifierType" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="EntityAttributeDesignator" type="xacml:EntityAttributeDesignatorType" substitutionGroup="xacml:NonLiteralExpressionType"/>
<xs:complexType name="EntityAttributeDesignatorType">
  <xs:complexContent>
    <xs:extension base="xacml:NamedAttributeDesignatorType">
      <xs:sequence>
        <xs:element ref="xacml:Expression"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="Value" type="xacml:ValueType" substitutionGroup="xacml:ExpressionType"/>
<xs:complexType name="ValueType" mixed="true">
  <xs:complexContent mixed="true">
    <xs:extension base="xacml:ExpressionType">
      <xs:sequence>
        <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="DataType" type="xacml:IdentifierType" use="optional"/>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="Function" type="xacml:FunctionType" substitutionGroup="xacml:ExpressionType"/>
<xs:complexType name="FunctionType">
  <xs:complexContent>
    <xs:extension base="xacml:ExpressionType">
      <xs:attribute name="Id" type="xacml:IdentifierType" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="Condition" type="xacml:BooleanExpressionType"/>
<xs:complexType name="BooleanExpressionType">
  <xs:sequence>
    <xs:element ref="xacml:NonLiteralExpression"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="Apply" type="xacml:ApplyType" substitutionGroup="xacml:NonLiteralExpressionType"/>
<xs:complexType name="ApplyType">
  <xs:complexContent>
    <xs:extension base="xacml:NonLiteralExpressionType">
      <xs:sequence>
        <xs:element ref="xacml:Description" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

```

```

<xs:element ref="xacml:Expression" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      The datatype of each function argument here must be known from
      the function function signature, OR it must be explicitly speci-
    </xs:documentation>
  </xs:annotation>
</xs:element>
</xs:sequence>
<xs:attribute name="FunctionId" type="xacml:IdentifierType" use="required"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:element name="ForAny" type="xacml:QuantifiedExpressionType" substitutionGroup="xacml:NonLiteralExpression"/>
<xs:element name="ForAll" type="xacml:QuantifiedExpressionType" substitutionGroup="xacml:NonLiteralExpression"/>
<xs:element name="Map" type="xacml:QuantifiedExpressionType" substitutionGroup="xacml:NonLiteralExpression"/>
<xs:element name="Select" type="xacml:QuantifiedExpressionType" substitutionGroup="xacml:NonLiteralExpression"/>
<xs:complexType name="QuantifiedExpressionType">
  <xs:complexContent>
    <xs:extension base="xacml:NonLiteralExpressionType">
      <xs:sequence>
        <xs:element ref="xacml:NonLiteralExpression"/>
        <xs:element ref="xacml:Expression"/>
      </xs:sequence>
      <xs:attribute name="VariableId" type="xacml:LocalIdentifierType" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:simpleType name="IdentifierType">
  <xs:restriction base="xs:string">
    <xs:pattern value="[^{}]*(\{[A-Za-z][0-9A-Za-z]*(-[0-9A-Za-z]+)*\}[^{}]*)" />
    <xs:minLength value="1"/>
    <xs:whiteSpace value="collapse"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="ShortIdSet" type="xacml:ShortIdSetType">
  <xs:key name="ShortIdSetReference">
    <xs:selector xpath="xacml:ShortIdSetReference"/>
    <xs:field xpath=". />
  </xs:key>
  <xs:key name="ShortId_Name">
    <xs:selector xpath="xacml:ShortId"/>
    <xs:field xpath="@Name"/>
  </xs:key>
</xs:element>
<xs:complexType name="ShortIdSetType">
  <xs:sequence>
    <xs:element ref="xacml:ShortIdSetReference" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:ShortId" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="Id" type="xs:anyURI" use="required"/>
</xs:complexType>
<xs:element name="ShortIdSetReference" type="xs:anyURI"/>
<xs:element name="ShortId" type="xacml:ShortIdType"/>
<xs:complexType name="ShortIdType">
  <xs:attribute name="Name" type="xacml:ShortIdNameType" use="required"/>
  <xs:attribute name="Value" type="xacml:ShortIdValueType" use="required"/>
</xs:complexType>
<xs:simpleType name="ShortIdNameType">
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Za-z][0-9A-Za-z]*(-[0-9A-Za-z]+)*" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ShortIdValueType">
  <xs:restriction base="xs:string">
    <xs:annotation>
      <xs:documentation xml:lang="en">
        ShortId Value type. Such value is an alternating sequence of URI characters and {ShortId
        curly braces} (reminder: the curly brace is not a valid URI character), so the
        ^uc*(\{s\}uc*)*$

        where:
        - s is the pattern for a ShortId name, copied from the ShortIdNameType definition (which
          [A-Za-z][0-9A-Za-z]*(-[0-9A-Za-z]+)*
        - uc is the pattern for a valid URI character based on RFC 3986 (cf. the ABNF definition
          which excludes curly braces: [#-;=?\[_a-zA-Z~]
      </xs:documentation>
    </xs:annotation>
  </xs:restriction>
</xs:simpleType>

```

```

The minLength=1 restriction prevents the empty string, i.e. there is at least one uc or
An XSD pattern      is matched against the entire string, so the ^ and $ anchors are
</xs:documentation>

</xs:annotation>
<xs:pattern value="[^#-;=?-\[\]_a-z~]*(\{[A-Za-z][0-9A-Za-z]*(-[0-9A-Za-z]+)*\}[!#-;=?-\[\]_a-z-]*)*"/>
<xs:minLength value="1"/>
</xs:restriction>
</xs:simpleType>
<xs:element name="SharedVariableDefinition" type="xacml:SharedVariableDefinitionType">
  <xs:key name="SharedVariableDefinition_ShortIdSetReference">
    <xs:selector xpath="xacml:ShortIdSetReference"/>
    <xs:field xpath=". "/>
  </xs:key>
  <xs:key name="SharedVariableDefinition_Parameter_Name">
    <xs:selector xpath="xacml:Parameter"/>
    <xs:field xpath="@Name"/>
  </xs:key>
</xs:element>
<xs:complexType name="SharedVariableDefinitionType">
  <xs:sequence>
    <xs:element ref="xacml:Description" minOccurs="0"/>
    <xs:element ref="xacml:ShortIdSetReference" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:Parameter" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="xacml:Expression">
      <xs:annotation>
        <xs:documentation xml:lang="en">
          If the Expression is a Value, the DataType is
          'urn:oasis:names:tc:acal:1.0:data-type:string' by default, if not otherwise
          defined.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="Id" type="xs:anyURI" use="required"/>
  <xs:attribute name="Version" type="xacml:VersionType" use="required"/>
</xs:complexType>
<xs:element name="SharedVariableReference" type="xacml:SharedVariableReferenceType"/>
<xs:complexType name="SharedVariableReferenceType">
  <xs:complexContent>
    <xs:extension base="xacml:NonLiteralExpressionType">
      <xs:sequence>
        <xs:element ref="xacml:Expression" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation xml:lang="en">
              Optional argument(s) to the referenced
              SharedVariableDefinition. The arguments SHALL match the
              SharedVariableDefinition's Parameter definitions in the same order
              of declaration.
              If one of the argument Expressions is a Value element, it SHALL
              set the DataType attribute value.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="Id" type="xs:anyURI" use="required"/>
      <xs:attribute name="Version" type="xacml:VersionMatchType" use="optional"/>
      <xs:assert vc:minVersion="1.1" test="not(xacml:Value/@DataType)"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="Bundle" type="xacml:BundleType">
  <xs:key name="Bundle_ShortIdSet_Id">
    <xs:selector xpath="xacml:ShortIdSet"/>
    <xs:field xpath="@Id"/>
  </xs:key>
  <xs:key name="Bundle_SharedVariableDefinition_Id">
    <xs:selector xpath="xacml:SharedVariableDefinition"/>
    <xs:field xpath="@Id"/>
  </xs:key>
  <xs:key name="Bundle_Policy_PolicyId">
    <xs:selector xpath="xacml:Policy"/>
    <xs:field xpath="@PolicyId"/>
  </xs:key>
</xs:element>
<xs:complexType name="BundleType">
  <xs:sequence>

```

```

<xs:element ref="xacml:ShortIdSet" minOccurs="0" maxOccurs="unbounded"/>
<xs:element ref="xacml:SharedVariableDefinition" minOccurs="0" maxOccurs="unbounded"/>
<xs:element ref="xacml:Policy" minOccurs="0" maxOccurs="unbounded"/>
<xs:element ref="xacml:PolicyReference" minOccurs="0"/>
</xs:sequence>
<xs:assert vc:minVersion="1.1" test="not(xacml:PolicyReference) or xacml:Policy">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      If there is a PolicyRef, there must be at least one Policy element .
    </xs:documentation>
  </xs:annotation>
</xs:assert>
</xs:complexType>
</xs:schema>

```

---

## Annex E 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 code repository. Go to Github Actions on the 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

The following tools are required:

- Pandoc;
- Pandoc-include filter.

Either install them on your system or, if you have Docker installed already, simply use the following shell alias:

```
$ alias pandoc='docker run --rm --volume "$(pwd):/data" pandoc/extral'
```

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 code 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-xml-v4.0-csd01.md`) to an output file `/tmp/acal-core-xml-v4.0-csd01.html`:

```
$ pandoc -s --verbose --embed-resources -f gfm+definition_lists -c styles/markdown-styles-v1.7.3a.css -F pandoc-include -M lang
```

Note this command generates a document which may require 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, the command line is the following (different -t and -H arguments, and output goes to file `/tmp/acal-core-xml-v4.0-csd01.pdf`):

```
$ pandoc -s --embed-resources -f gfm+definition_lists -c styles/markdown-styles-v1.7.3a.css -F pandoc-include -H pandoc/custom-include.html -o output.html
```

## 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

Substantial contributions to this document from the following individuals 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.)

XACML 4.0 differs from XACML 3.0 in the following ways:

- 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.
  - `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.
- `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 a `Body` element (content body which is similar to former `Content` value) and two attributes: `MediaType` (Content media type which indicates whether it is XML, JSON or some other content type) and `Encoding` (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 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.
- Changed `Decision` element to an XML attribute in `Result`.
- 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 code repository.

Revision	Date	Editor	Changes Made
specname-v1.0-wd01	2024-08-09	Steven Legg	Initial working draft
	2024-09-09	Steven Legg	Changed <code>targets</code> to a single child expression. Removed the <code>&lt;AnyOf&gt;</code> , <code>&lt;AllOf&gt;</code> and <code>&lt;Match&gt;</code> elements. Removed <code>targets</code> from rules.
	2024-09-28	Cyril Dangerville	Removed the only-one-applicable <i>combining algorithm</i> . Embedded diagrams.
	2024-09-30	Cyril Dangerville	Added support for definition lists.
	2024-10-22	Steven Legg	Reformatted indented paragraphs.
	2025-04-16	Steven Legg	Made MustBePresent, CombinedDecision and ReturnPolicyIdList X
	2025-04-17	Steven Legg	Restructured the type hierarchy for policy references.
	2025-07-15	Steven Legg	Merged <code>obligations</code> and <code>advice</code> into <code>notices</code> .
	2025-07-23	Steven Legg	Added support for <i>short identifiers</i> .
	2025-07-25	Steven Legg	Added example two showing <i>short identifiers</i> . Grouping <i>short identifiers</i> in sets rather than collections. Replaced ShortIdentifier with ShortId. Replaced SIDCollection with ShortIdSet.