JSON Profile of XACML 3.0 Version 1.0

Candidate OASIS Standard 01

12 October 2017

Specification URIs

This version:

http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/cos01/xacml-json-http-v1.0-cos01.doc (Authoritative)
http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/cos01/xacml-json-http-v1.0-cos01.html
http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/cos01/xacml-json-http-v1.0-cos01.pdf

Previous version:

http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csprd03/xacml-json-http-v1.0-csprd03.doc (Authoritative)
http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csprd03/xacml-json-http-v1.0-csprd03.html
http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csprd03/xacml-json-http-v1.0-csprd03.pdf

Latest version:

http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/xacml-json-http-v1.0.doc (Authoritative)
http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/xacml-json-http-v1.0.html
http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/xacml-json-http-v1.0.pdf

Technical Committee:

OASIS eXtensible Access Control Markup Language (XACML) TC

Chairs:

Hal Lockhart (hal.lockhart@oracle.com), Oracle
Bill Parducci (bill@parducci.net), Individual

Editor:

David Brossard (david.brossard@axiomatics.com), Axiomatics AB

Related work:

This specification is related to:

- *eXtensible Access Control Markup Language (XACML) Version 3.0*. Edited by Erik Rissanen. 22 January 2013. OASIS Standard. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

Abstract:

The aim of this profile is to propose a standardized interface between a policy enforcement point and a policy decision point using JSON. The decision request and response structure is specified in the core XACML specification. This profile leverages it.

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://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml#technical.

TC members should send comments on this specification 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/.

This Candidate OASIS Standard is provided under the RF on Limited Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (https://www.oasis-open.org/committees/xacml/ipr.php).

Citation format:

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

[xacml-json-v1.0]

JSON Profile of XACML 3.0 Version 1.0. Edited by David Brossard. 12 October 2017. Candidate OASIS Standard 01. http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/cos01/xacml-json-http-v1.0-cos01.html. Latest version: http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/xacml-json-http-v1.0.html.

Notices

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

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

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

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

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

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

Table of Contents

1 Introduction

[All text is normative unless otherwise labeled]

{Non-normative}

The XACML architecture promotes a loose coupling between the component that enforces decisions, the policy enforcement point (PEP), and the component that decides based on XACML policies, the policy decision point (PDP).

The XACML standard defines the format of the request and the response between the PEP and the PDP. As the default representation of XACML is XML and is backed by a schema, the request and response are typically expressed as XML elements or documents. Depending on the PDP implementation, the request and response could be embedded inside a SOAP message or even a SAML assertion as described in the SAML profile of XACML.

With the rise in popularity of APIs and its consumerization, it becomes important for XACML to be easily understood in order to increase the likelihood it will be adopted.

This profile aims at defining a JSON format for the XACML request and response. It also defines the transport between client (PEP) and service (PDP).

In writing this document, the authors have kept three items in mind:

1. Equivalence: a XACML request and response expressed in XML need not be strictly equivalent in structure to a XACML request expressed in JSON so long as the meaning remains the same and so long as the JSON and XML requests would lead to the same response (decision, obligation, and advice).
2. Lossless behavior: it MUST be possible to translate XACML requests and responses between XML and JSON representations in either direction at any time without semantic loss.
3. Transport-agnostic nature: the JSON representation MUST contain all the information the XACML request and/or response contains: this means the transport layer cannot convert XACML decisions into HTTP codes, e.g. HTTP 401 for a Deny decision.

1.1 Terminology

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

1.2 Normative References

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.

[RFC4627] D. Crockford, *The application/json Media Type for JavaScript Object Notation (JSON)*, http://tools.ietf.org/html/rfc4627, IETF RFC 4627, July 2006.

[XACMLMDP] *XACML v3.0 Multiple Decision Profile Version 1.0*. Latest version. http://docs.oasis-open.org/xacml/3.0/multiple/v1.0/xacml-3.0-multiple-v1.0.html

[ECMA262] S. Bradner, *ECMAScript Language*, http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf, Standard ECMA 262, June 2011.

[NAMESPACES] Bray, Tim, et.al. eds, Namespaces in XML 1.0 (Third Edition), W3C Recommendation 8 December 2009, available at http://www.w3.org/TR/2009/REC-xml-names-20091208/

[XACML30] *eXtensible Access Control Markup Language (XACML) Version 3.0*. Latest version. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html

[XML] Bray, Tim, et.al. eds, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, W3C Recommendation 26 November 2008, available at http://www.w3.org/TR/2008/REC-xml-20081126/

[XMLDatatypes] Biron, Paul et al. Eds, *XML Schema Part 2: Datatypes Second Edition*, W3C Recommendation 28 October 2004, available at http://www.w3.org/TR/xmlschema-2/

[XPATH] James Clark and Steve DeRose, XML Path Language (XPath), Version 1.0, W3C Recommendation 16 November 1999. Available at: http://www.w3.org/TR/xpath

[IEEE754] Institute of Electrical and Electronics Engineers, "Standard for Floating-Point Arithmetic", IEEE Standard 754, August 2008.

1.3 Non-Normative References

[XACMLREST] *REST Profile of XACML v3.0 Version 1.0*. Edited by Rémon Sinnema. Latest version: http://docs.oasis-open.org/xacml/xacml-rest/v1.0/xacml-rest-v1.0.html.

[HTTP] *Hypertext Transfer Protocol*. June 1999. IETF RFC 2616. http://tools.ietf.org/html/rfc2616

[HTTPS] *HTTP over TLS*. May 2000. IETF RFC 2818. http://tools.ietf.org/html/rfc2818

[BASE64] *The Base16, Base32, and Base64 Data Encodings*. October 2006. IETF RFC 4648. http://tools.ietf.org/html/rfc4648

2 Vocabulary

{Non-normative}

XML introduces the notion of elements. The equivalent notion in JSON is an object. XML introduces the notion of attributes. The equivalent notion in JSON is a member.

3 Overview of the translation mechanisms

3.1 Assumed default values

To avoid bloating the JSON request and response, certain parts of a request and response have default values which can then be omitted. As an example, the default value for the data-type of an attribute value is `String` (http://www.w3.org/2001/XMLSchema#string).

The user should refer to the XACML 3.0 specification document **[XACML30]** for a normative definition of the request and response elements.

3.2 Objects

3.2.1 Object names

Unless otherwise stated, JSON object names MUST match the XACML XML element and/or attribute names exactly, including case.

The following XML elements and attributes have been renamed:

- The name of the XACML XML `Attributes` element has been changed in JSON to the `Category` object. It makes more sense to call the parent element that way since it represents an instance of a category from a XACML sense.
- The `AttributeValue` element in the XML representation no longer exists. The information it bears in XML is moved to the parent `Attribute` object in the JSON representation. A `Value` property has been introduced in the JSON `Attribute` object to bear the information contained in the XML `AttributeValue` element as specified in Section 4. The XACML request.
- The `AdviceId` and the `ObligationId` attributes of the `<Advice/>` and the `<Obligation/>` XML elements respectively have been renamed to `Id` in JSON.

3.2.2 Object order

The order of the objects and values in XACML does not matter. Therefore, the order of objects and values in the serialized form (JSON) does not matter.

3.2.3 Object cardinality

When in the XACML specification, an object (XML element) can occur more than once (e.g. 0..* or 1..*), the JSON equivalent MUST use an array of objects.

The class diagram in Section 4.1. Class Diagram states the cardinality and relationship between objects.

3.3 Data Types

This section defines how data-types are represented and handled in the JSON representation. Chapter 10, section 10.2.7 in the XACML 3.0 specification as well as section A.2 list the data-types that are defined in XACML. These are listed in the table below in section 3.3.1. It lists the shorthand value that MAY be used when creating a XACML attribute in the JSON representation.

3.3.1 Supported Data Types

The full XACML data type URI can also be used in JSON as the JSON shorthand type codes are a convenience, not a replacement.

It is also possible to omit the JSON property `DataType` for certain XACML data types when it can safely be inferred from the value of the attribute as shown in Table 1.

107 *Table 1. JSON shorthand and rules of inference for XACML data types.*

XACML data type identifier	JSON shorthand type code	Mapping / Inference Rule
http://www.w3.org/2001/XMLSchema#string	string	JSON "String"
http://www.w3.org/2001/XMLSchema#boolean	boolean	JSON "Boolean"
http://www.w3.org/2001/XMLSchema#integer	integer	JSON "Number" with no fractional portion and within the integer range defined by the XML schema in **[XMLDatatypes].**
http://www.w3.org/2001/XMLSchema#double	double	JSON "Number" with fractional portion or out of integer range as defined in **[XMLDatatypes].**
http://www.w3.org/2001/XMLSchema#time	time	None – inference must fail.
http://www.w3.org/2001/XMLSchema#date	date	None – inference must fail.
http://www.w3.org/2001/XMLSchema#dateTime	dateTime	None – inference must fail.
http://www.w3.org/2001/XMLSchema#dayTimeDuration	dayTimeDuration	None – inference must fail.
http://www.w3.org/2001/XMLSchema#yearMonthDuration	yearMonthDuration	None – inference must fail.
http://www.w3.org/2001/XMLSchema#anyURI	anyURI	None – inference must fail.
http://www.w3.org/2001/XMLSchema#hexBinary	hexBinary	None – inference must fail.
http://www.w3.org/2001/XMLSchema#base64Binary	base64Binary	None – inference must fail.
urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name	rfc822Name	None – inference must fail.
urn:oasis:names:tc:xacml:1.0:data-type:x500Name	x500Name	None – inference must fail.
urn:oasis:names:tc:xacml:2.0:data-type:ipAddress	ipAddress	None – inference must fail.
urn:oasis:names:tc:xacml:2.0:data-type:dnsName	dnsName	None – inference must fail.
urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression	xpathExpression	None – inference must fail

108 For all of the XACML data types that cannot be inferred from the value, the following MUST be observed:

109 • The JSON `DataType` property MUST be specified and the value expressed in the XACML string
110 representation of the value.

111 • Implementation-specific (e.g. Javascript) code may choose to parse the XACML string values into
112 internal numeric representations for internal use, such as for `DateTime` or duration
113 (`dayTimeDuration`, `yearMonthDuration`) values, but the JSON transport representation
114 must always express the value in the serialized XACML string representation of the XACML data
115 type.

3.3.2 Arrays of values

In the case of an array of values, and if the `DataType` member is not specified, it may not be possible to infer the `DataType` until all the values have been inspected.

Inference for an array of values works according to the inference rules as set in Section 3.3.1. If a given data type cannot be inferred and there is no `DataType` member specified then the array of values will be considered as an array of string.

If an array of values contains integers and doubles only (excluding non-numerical values), then the inference will make the array an array of double.

Any other combination of values will make the inference fail and the array will be considered as an array of string.

3.3.3 The xpathExpression Datatype

Values of the `xpathExpression` data-type are represented as JSON objects. Each such object contains the following properties:

Table 2 - Properties of the xPathExpression Datatype

Attribute	Type	Mandatory/Optional	Default value
XPathCategory	URI	Mandatory	None. The shorthand notation defined in section 4.2.2.1 can be used as values here.
Namespaces	Array of NamespaceDeclaration	Optional	None
XPath	String	Mandatory	None

The `XPath` property contains the XPath expression **[XPATH]** from the XACML value. The `Namespaces` property contains namespace declarations for interpreting qualified names **[NAMESPACES]** in the XPath expression.

A `NamespaceDeclaration` object contains the following properties:

Table 3 - Properties of the NamespaceDeclaration Datatype

Attribute	Type	Mandatory/Optional	Default value
Prefix	String	Optional	None
Namespace	URI	Mandatory	None

Each NamespaceDeclaration object describes a single XML namespace declaration **[NAMESPACES]**. The Prefix property contains the namespace prefix and the Namespace property contains the namespace name. In the case of a namespace declaration for the default namespace the Prefix property SHALL be absent.

The Namespaces array MUST contain a NamespaceDeclaration object for each of the namespace prefixes used by the XPath expression. The Namespaces array MAY contain additional NamespaceDeclaration objects for namespace prefixes that are not used by the XPath expression. There SHALL NOT more than one NamespaceDeclaration objects for the same namespace prefix.

3.3.3.1 Example

{Non-normative}

This example shows the XML representation of an XACML attribute with a value of the `xpathExpression` data-type and its corresponding representation in JSON.

- As XML:

```
<Attribute xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
```

```
        AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector">
        <AttributeValue xmlns:md="urn:example:med:schemas:record"
        XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
        DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
          >md:record/md:patient/md:patientDoB</AttributeValue>
    </Attribute>
```

- As JSON:

```
        {"Attribute": {
            "AttributeId": "urn:oasis:names:tc:xacml:3.0:content-
selector",
            "DataType": "xpathExpression",
            "Value": {
                "XPathCategory":
"urn:oasis:names:tc:xacml:3.0:attribute-category:resource",
                "Namespaces": [{
                    "Namespace":
"urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
                },
                {
                    "Prefix": "md",
                    "Namespace": "urn:example:med:schemas:record"
                }],
                "XPath": "md:record/md:patient/md:patientDoB"
            }
        }}
```

3.3.4 Special numeric values

The following special numeric values are not supported by the profile. Should the request contain such values, the Policy Decision Point MUST reply with an Indeterminate with a status value of `urn:oasis:names:tc:xacml:1.0:status:syntax-error` as defined in Appendix B, section 8 of **[XACML30]**.

Additional behavior of the PDP when returning `urn:oasis:names:tc:xacml:1.0:status:syntax-error` is specified in sections 5.57 and B.8 of **[XACML30]**.

- IEEE 754-2008 NaN ("NaN")
- IEEE 754-2008 positive infinity ("INF")
- IEEE 754-2008 negative infinity ("-INF")
- IEEE 754-2008 negative zero (-0)

3.4 Example

{Non-normative}

The example below illustrates possible notations and the behavior of the JSON interpreter:

Table 4 - Equivalent examples

Representation explicitly stating the data-type	Representation omitting the data-type
{"Attribute": { "AttributeId" : "document-id" "DataType" : "integer" "Value" : 123 }}	{"Attribute": { "AttributeId": "document-id" "Value" : 123 }}

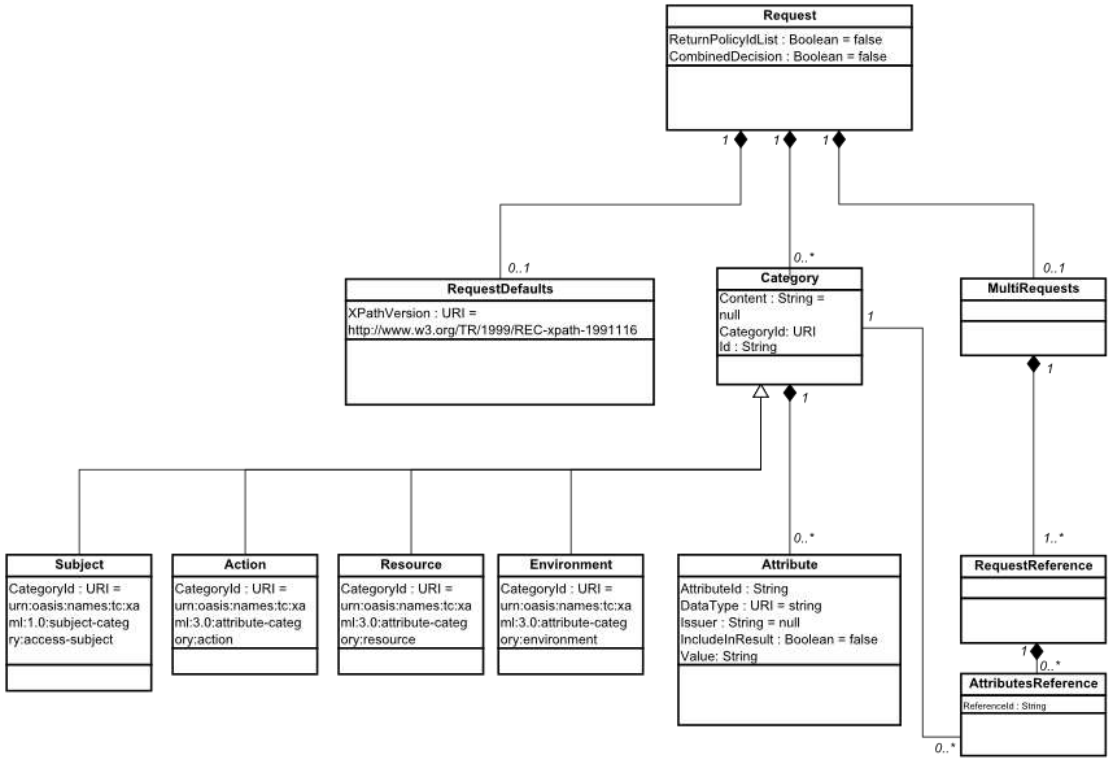
4 The XACML request

4.1 Class Diagram

The following class diagram represents the XACML request structure for the JSON representation. It is not a representation of the XACML request as expressed in XML.

The key differences are:

- The `AttributeValue` element in the XML representation no longer exists. The information it bears in XML is moved to the parent `Attribute` object in the JSON representation.
- There are 4 new objects for attributes belonging to the most commonly used categories.



4.2 Representation of the XACML request in JSON

4.2.1 The Request object representation

The JSON object name for the request MUST be `Request`.

The `Request` object contains the following properties:

- `ReturnPolicyIdList` of type Boolean
- `CombinedDecision` of type Boolean
- `XPathVersion` of type String

These properties are represented as members. The JSON representation assumes the following default values:

Table 5 - Properties of the Request object

Attribute	Type	Default value
`ReturnPolicyIdList`	Boolean	False. `ReturnPolicyIdList` can be omitted in the JSON representation.
`CombinedDecision`	Boolean	False. `ReturnPolicyIdList` can be omitted in the JSON representation.
`XPathVersion`	String	There is no default value. The attribute is optional. It is REQUIRED if the XACML request contains XPath expressions.

In addition to these properties, the Request element also contains the following objects:

- Category: this is represented as a JSON array of `Category` objects; the `Category` object corresponds to the XML `Attributes` element. Just like the `Attributes` element is specific to a given attribute category, the `Category` object in JSON is specific to a given category.
- MultiRequests: this is an optional object and can be omitted. It serves to support the Multiple Decision Profile **[XACMLMDP]**.

The representation of these objects is elicited in the following relevant sections.

Note that, in the XACML XML schema, the XML Request element contains a `RequestDefaults` element. To simplify things and since the `RequestDefaults` element contained a single element `XPathVersion` with a single value, the `RequestDefaults` element was flattened into a single JSON property called `XPathVersion` as mentioned in the above table.

4.2.1.1 Example

{Non-normative}

```
{"Request": {
      "XPathVersion": "http://www.w3.org/TR/1999/REC-xpath-19991116"
      }
}
```

4.2.2 The Category object representation

The JSON `Category` object contains the following properties:

Table 6 - Properties of the Category object

Attribute	Type	Mandatory/Optional	Default value
CategoryId	anyURI	Mandatory	None – the identifier used in the XML representation MUST be used in its JSON representation except where shorthand notations have been defined – see section 4.2.2.1.
Id	String	Optional	The `Id` property is optional in the JSON representation. No default value is assumed for the `Id` in JSON. If there is a value specified in the XML representation, it must also be specified

			in the JSON representation.
Content	String	Optional	None. The value of the `Content` property must be escaped or encoded as explained in section 4.2.3.

In addition to these properties, the `Category` object also contains:

- Attribute: this is an array of `Attribute` objects as defined in section 4.2.4, The Attribute Object representation.

The `Category` object is the equivalent of the `<Attributes/>` element in the XACML XML representation.

The structure and default values for the aforementioned are elicited in the following relevant sections.

4.2.2.1 Shorthand notation for standard XACML categories

The following table defines a shorthand notation for the standard categories defined in **[XACML30]**.

Table 7 - Shorthand notation for standard XACML categories

Identifier	Short name
urn:oasis:names:tc:xacml:3.0:attribute-category:resource	Resource
urn:oasis:names:tc:xacml:3.0:attribute-category:action	Action
urn:oasis:names:tc:xacml:3.0:attribute-category:environment	Environment
urn:oasis:names:tc:xacml:1.0:subject-category:access-subject	AccessSubject
urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject	RecipientSubject
urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject	IntermediarySubject
urn:oasis:names:tc:xacml:1.0:subject-category:codebase	Codebase
urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine	RequestingMachine

The shorthand notation MAY be used as described in sections 4.2.2.2 and 4.2.2.

4.2.2.2 Default Category objects

To simplify the JSON representation, this profile also defines optional default objects that are semantically equivalent to the `Category` object. These default objects assume a default value for the `CategoryId` property so that it need not be explicitly written. The object names correspond to the short names as defined in section 4.2.2.1.

Note that JSON does not allow for the duplication of objects that bear the same name, e.g. "AccessSubject" and "AccessSubject". Consequently, the optional default objects (based on section 4.2.2.1) can also be an array instead of single-valued in order to cater for multiple decision requests as defined in **[XACMLMDP]**.

4.2.2.3 Example

{Non-normative}

```
{
    "Request": {
        "Category": [{
            "CategoryId": "custom-category",
            "Attribute": […]
```

```
          },
          {
               "CategoryId": "another-custom-cat",
               "Attribute": […]
               }
          }],
        "AccessSubject":{
               "Attribute": […]
        },
        "Action":[{
               "Attribute": […]
        },
        {
               "Attribute": […]
        }]
    }
}
```

4.2.3 The Content Object representation

There are two possible ways to represent the XML content of a XACML request in the JSON representation: XML escaping or Base64 encoding. The request parser must determine whether XML escaping or Base 64 encoding is used. There are no attributes or parameters in the JSON request to indicate which is used.

In both cases, any XML content sent in a JSON request MUST include all Namespace definitions needed to parse that Content.

4.2.3.1 XML Escaping

The JSON `Content` object data-type is a string which MUST be null or contain an XML payload per the XACML specification.

XML Content must be escaped before being inserted into the JSON request. JSON dictates double quotes (") be escaped using a backslash (\). This profile therefore follows this behavior.

In addition, since the XML content could itself contain backslashes and possibly the sequence \", it is important to also escape backslashes.

4.2.3.2 Base64 Encoding

In the case of Base64 encoding, the XML content shall be converted to its Base64 representation as per **[BASE64]**.

4.2.3.3 Example

{Non-normative}

The following is an example using XML escaping as defined in 4.2.3.1.

```
{"Request":
{"AccessSubject": {
      "Content": "<?xml version=\"1.0\"?><catalog><book
id=\"bk101\"><author>Gambardella, Matthew</author><title>XML Developer's
Guide</title><genre>Computer</genre><price>44.95</price><publish_date>2000-
10-01</publish_date><description>An in-depth look at creating applications
with XML.</description></book></catalog>"
```

```
}}}
```

The following is an example using Base64 encoding as defined in 4.2.3.2.

```
{"Request":
{
      "AccessSubject":{
            "Content":
"PD94bWwgdmVyc2lvbj0iMS4wIj8+DQo8Y2F0YWxvZz48Ym9vayBpZD0iYmsxMDEiPjxhdXRob3I+
R2FtYmFyZGVsbGEsIE1hdHRoZXc8L2F1dGhvcj48dGl0bGU+WE1MIERldmVsb3BlcidzIEd1aWRlP
C90aXRsZT48Z2VucmU+Q29tcHV0ZXI8L2dlbnJlPjxwcmljZT40NC45NTwvcHJpY2U+PHB1Ymxpc2
hfZGF0ZT4yMDAwLTEwLTAxPC9wdWJsaXNoX2RhdGU+PGRlc2NyaXB0aW9uPkFuIGluLWRlcHRoIG
vb2sgYXQgY3JlYXRpbmcgYXBwbGljYXRpb25zIHdpdGggWE1MLjwvZGVzY3JpcHRpb24+PC9ib29r
PjwvY2F0YWxvZz4="
      }
}}
```

4.2.4 The Attribute Object representation

The JSON `Attribute` object contains an array of `Attribute` objects. The `Attribute` object contains the following properties:

Table 8 - Properties of the Attribute Object

Property name	Type	Mandatory/Optional	Default value
AttributeId	URI	Mandatory	None – the identifier used in the XML representation of a XACML attribute shall be used in its JSON representation
Value	Either of String, Boolean, Number (which maps to either a XACML integer or double as defined in Supported Data Types), Object, Array of String, Array of Boolean, Array of Number, Array of Object, or a mixed Array of String and Number where the String values represent a numerical value.	Mandatory	None – the value must be specified.
Issuer	String	Optional	Null
DataType	URI	Optional	The `DataType` value can be omitted in the JSON representation. Its default value will be `http://www.w3.org/2001/XMLSchema#string` unless it can be safely assumed according to the rules set in 3.3.1 Supported Data Types. In the case of an array of values, inference works as described in section 3.3.2.
IncludeInResult	Boolean	Optional	False.

4.2.4.1 Example

{Non-normative}

```
{"Attribute": [{
    "AttributeId": "urn:oasis:names:tc:xacml:2.0:subject:role",
    "Value": ["manager","administrator"]
}]}
```

4.2.5 The MultiRequests object representation

The `MultiRequests` object is optional in the JSON representation of XACML. Its purpose is to support the Multiple Decision Profile **[XACMLMDP]**.

The `MultiRequests` object contains an array of `RequestReference` objects. There must be at least one `RequestReference` object inside the `MultiRequests` object.

4.2.6 The RequestReference object representation

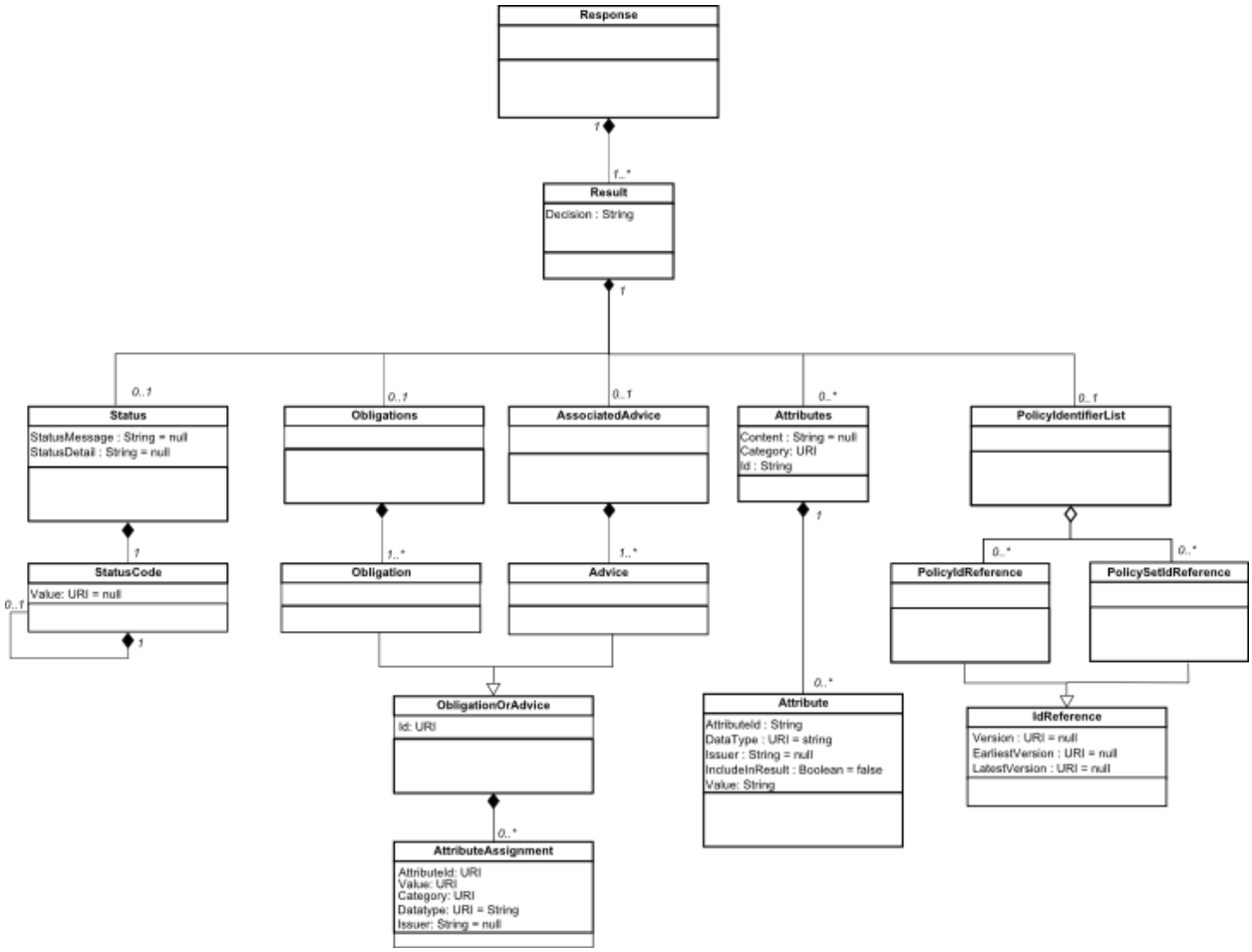
The `RequestReference` object contains a single property called `ReferenceId` which is an array of string. Each `ReferenceId` value must be the value of a `Category` object `Id` property.

4.2.6.1 Non-normative example

```
{
"MultiRequests": {
            "RequestReference": [{
                "ReferenceId": ["foo1","bar1"]
            },
            {
                "ReferenceId": ["foo2","bar1"]
            },
            {
                "ReferenceId": ["foo3","bar1"]
            }]
      }
}
```

5 The XACML response

5.1 Class Diagram



5.2 Representation of the XACML response in JSON

5.2.1 The Response object representation

The `Response` property MAY contain an array of `Result` objects. The array MUST contain at least one `Result` object and is unbounded.The `Result` object representation is detailed hereafter.

The JSON representation effectively eliminates the nesting of Response and Result as introduced in XACML's XML schema. The notion of an array of values is used to convey the nesting.

5.2.2 The Result object representation

The `Result` object in JSON will contain the following properties:

Table 9 - Properties of the Result object

Property name	Type	Mandatory/Optional	Default value
Decision	String	Mandatory	None – in addition there are only 4 valid values:

			"Permit", "Deny", "NotApplicable", and "Indeterminate". The values are case-sensitive.

Additionally, the `Result` object also contains the following objects:

- Status: this object is optional.
- Obligations: this object is optional.
- AssociatedAdvice: this object is optional.
- Category: this object is optional. It can be single-valued or an array of `Category` objects.
- PolicyIdentifierList: this object is optional.

5.2.3 The Status object representation

The `Status` object should contain the following properties:

Table 10 - Properties of the Status object

Property name	Type	Mandatory/Optional	Default value
StatusMessage	String	Optional	None.
StatusDetail	String	Optional	None.

In addition to the above properties, the `Status` object in JSON also contains a `StatusCode` object detailed hereafter. The `StatusCode` object is optional.

`StatusDetail` MAY contain arbitrary XML in which case the XML content must be escaped using the same technique as specified in section 4.2.3, The Content Object representation.

`StatusDetail` MAY contain an array of `MissingAttributeDetail` object.

5.2.4 The MissingAttributeDetail object

The `MissingAttributeDetail` object in JSON contains the following properties:

Table 11 - Properties of the MissingAttributeDetail object

Property name	Type	Mandatory / Optional	Default value
AttributeId	URI	Mandatory	None – the identifier used in the XML representation of a XACML attribute shall be used in its JSON representation
Value	Either of String, Boolean, Number (which maps to either a XACML integer or double as defined in Supported Data Types), Object, Array of String, Array of Boolean, Array of Number, Array of Object, or a mixed Array of String and Number where the String values represent a numerical value.	Optional	None – the value must be specified.
Issuer	String	Optional	Null
DataType	URI	Optional	The `DataType` value can be omitted in the JSON representation. Its default value will be `http://www.w3.org/2001/XMLSchema#string` unless it can be safely assumed according to the rules set in section 3.3.1 Supported Data Types. In the case of an array of values, inference works as described in section 3.4.2.
Category	URI	Mandatory	Note that the shorthand notation for default XACML 3.0 categories may be used. See section 4.2.2.1.

380

381 5.2.5 The StatusCode object representation

382 The `StatusCode` object in JSON contains the following properties:

383 *Table 12 - Properties of the StatusCode object*

Property name	Type	Mandatory/Optional	Default value
Value	URI	Optional	urn:oasis:names:tc:xacml:1.0:status:ok.

384 In addition, the `StatusCode` object may contain a `StatusCode` object – hence potentially creating a
385 recursive nesting of `StatusCode` objects.

386 5.2.5.1 Example

387 {Non-normative}

```
{
    "Response": [{
        "Decision": "Permit"
        "Status":{
           "StatusCode":{
              "Value": "http://example.com"
           }
        }
       }]
}
```

5.2.6 The Obligations object representation

The `Obligations` property in the JSON representation is simply an array of `ObligationOrAdvice` objects. The `ObligationOrAdvice` object is detailed hereafter.

5.2.7 The AssociatedAdvice object representation

The `AssociatedAdvice` property in the JSON representation is simply an array of `ObligationOrAdvice` objects. The `Advice` object is detailed hereafter.

5.2.8 The ObligationOrAdvice object representation

The `ObligationOrAdvice` object contains the following properties:

Table 13 - Properties of the ObligationOrAdvice object

Property name	Type	Mandatory/Optional	Default value
Id	URI	Mandatory	None.

Note that the `ObligationOrAdvice` object maps to either an `Advice` or an `Obligation` element in the XACML XML representation. While in the XML representation, each element has an attribute called `AdviceId` and `ObligationId` respectively, in the JSON representation, the naming has been harmonized to `Id`.

The `ObligationOrAdvice` object contains an unbounded array of `AttributeAssignment` objects.

5.2.9 The AttributeAssignment object representation

The `AttributeAssignment` object contains the following properties:

Table 14 - Properties of the AttributeAssignment object

Property name	Type	Mandatory/Optional	Default value
AttributeId	URI	Mandatory	None.
Value	Variable	Mandatory	None
Category	URI	Optional	None. The shorthand notation defined in Shorthand notation for standard XACML categories may be used.
DataType	URI	Optional	The default value depends on the inference rules defined in Supported Data Types.
Issuer	String	Optional	None

5.2.10 The Attributes object representation

The JSON representation of the `Attributes` object in a XACML response is identical to the representation defined in section 4.2.2 The Category object representation.

5.2.11 The PolicyIdentifier object representation

The `PolicyIdentifier` object contains 2 properties:

Table 15 - Properties of the PolicyIdentifier object

Property name	Type	Mandatory/Optional	Default value
PolicyIdReference	Array of IdReference	Optional	None.
PolicySetIdReference	Array of IdReference	Optional	None

5.2.12 The IdReference object representation

The `IdReference` object representation contains the following properties:

Table 16 - Properties of the IdReference object

Property name	Type	Mandatory/Optional	Default value
Id	URI	Mandatory	Represents the value stored inside the XACML XML `PolicyIdReference` or `PolicySetIdReference`.
Version	String	Optional	None.

6 Transport

The XACML request represented in its JSON format MAY be carried from a PEP to a PDP via an HTTP **[HTTP]** request as defined in the REST profile of XACML [XACMLREST].

HTTP Headers which may be used are:

- Content-Type: application/xacml+json
- Accept: application/xacml+json

6.1 Transport Security

{Non-normative}

The use of SSL/TLS **[HTTPS]** is RECOMMENDED to protect requests and responses as they are transferred across the network.

7 IANA Registration

The following section defines the information required by IANA when applying for a new media type.

7.1 Media Type Name

application

7.2 Subtype Name

xacml+json

7.3 Required Parameters

None.

7.4 Optional Parameters

version: The version parameter indicates the version of the XACML specification. Its range is the range of published XACML versions. As of this writing that is: 1.0, 1.1, 2.0, and 3.0. These and future version identifiers are of the form x.y, where x and y are decimal numbers with no leading zeros, with x being positive and y being non-negative.

7.5 Encoding Considerations

Same as for application/xml **[RFC4627].**

7.6 Security Considerations

Per their specification, application/xacml+json typed objects do not contain executable content.

XACML requests and responses contain information which integrity and authenticity are important.

To counter potential issues, the publisher may use the transport layer's security mechanisms to secure

xacml+json typed objects when they are in transit. For instance HTTPS, offer means to ensure the confidentiality, authenticity of the publishing party and the protection of the request/response in transit.

7.7 Interoperability Considerations

XACML 3.0 uses the `urn:oasis:names:tc:xacml:3.0:core:schema:wd-17` XML namespace URI. XACML 2.0 uses the `urn:oasis:names:tc:xacml:2.0:policy` XML namespace URI.

7.8 Applications which use this media type

Potentially any application implementing XACML, as well as those applications implementing specifications based on XACML or those applications requesting an authorization decision from a XACML implementation.

7.9 Magic number(s)

Per **[RFC4627]**, this section is not applicable.

7.10 File extension(s)

Per **[RFC4627]**, .json.

7.11 Macintosh File Type Code(s)

Text

7.12 Intended Usage

Common

8 Examples

{Non-normative}

8.1 Request Example

{Non-normative}

The following is a sample XACML request expressed in JSON.

```
{
      "Request": {
            "AccessSubject": {
                  "Attribute": [
                        {
                              "AttributeId": "subject-id",
                              "Value": "Andreas"
                        },
                        {
                              " AttributeId": "location",
                              "Value": "Gamla Stan"
                        }
                  ]
            },
            "Action": {
                  "Attribute":
                        {
                              "AttributeId": "action-id",
                              "Value": "http://example.com/buy",
                              "DataType": "anyURI"
                        }
            },
            "Resource": {
                  "Attribute": [
                        {
                              "AttributeId": "book-title",
                              "Value": "Learn German in 90 days"
                        },
                        {
                              "AttributeId": "currency",
                              "Value": "SEK"
                        },
                        {
                              "AttributeId": "price",
                              "Value": 123.34
                        }
                  ]
```

```
                }
        }
}
```

8.2 Response Example

{Non-normative}

The following is a sample XACML response expressed in JSON.

```
{
        "Response": [{
                        "Decision": "Permit"
                }
        ]
}
```

9 Conformance

An implementation may conform to this profile if and only if both the XACML request and the response are correctly encoded into JSON as previously described in sections 3 through 5 and follows the transport requirements as specified in section 6.

Appendix A. Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Steven Legg, ViewDS
Rich Levinson, Oracle
Hal Lockhart, Oracle
Bill Parducci,
Erik Rissanen, Axiomatics
Anil Saldhana, Red Hat
Remon Sinnema, EMC
Danny Thorpe, Dell
Paul Tyson, Bell Helicopters

Appendix B. Revision History

Revision	Date	Editor	Changes Made
WD 01	2 Jul 2012	David Brossard	Initial working draft
WD 02	9 Jul 2012	David Brossard	Integrated comments from XACML list. Enhanced the section on data-types. Added a class diagram for clarity. Changed tense to present. Removed overly explicit comparisons with XML representation.
WD 03	19 Jul 2012	David Brossard	Started work on the XACML response
WD 04	20 Aug 2012	David Brossard	Finalized work on the XACML response, added a note on HTTPS. Restructured the document to extract paragraphs common to the Request and Response section.
WD 05	20 Sep 2012	David Brossard	Took in comments from the XACML TC list (technical comments and typographical corrections)
WD 06	29 Oct 2012	David Brossard	Removed the Non-normative section in the appendix. Completed the conformance section. Added non-normative tags where needed. Also added a sample response example. Added the section on IANA registration.
WD07	15 Nov 2012	David Brossard	Removed the XPathExpression from the supported DataTypes. Fixed the examples as per Steven Legg's email. Fixed the XML encoding of XML content as per conversations on the XACML TC list.
WD08	27 Nov 2012	David Brossard	Fixed the Base64 encoding section as per Erik Rissanen's comments
WD09	24 Dec 2012	David Brossard	Addressed comments and fixed errors as per emails sent on the XACML TC list in December.
WD10	4 Feb 2013	David Brossard	Fixed the IANA registration section. Fixed inconsistent DataType spelling. DataType is always the XACML attribute and JSON property name. Data type refers to the English notion. Fixed the status XML content encoding to be consistent with the Request XML encoding technique. Fixed a non-normative section label. Fixed the formatting of JSON property names. Fixed the XACML to JSON data type inference by adding references to the relevant XML data types.

WD11	5 Feb 2013	David Brossard	Fixed the AttributeAssignment section
WD12	10 May 2013	David Brossard	Reinserted a section on the xpathExpression data type. Fixed the PolicyIdReference section (missing value). Fixed the Response example. Simplified the XPathVersion / RequestDefaults Renamed Attributes → Category Removed unnecessary nesting in Response → Result Renamed Attributes to Category
WD13	14 June 2013	David Brossard	Fixed the final issue re. Category vs. Attributes.
WD14	12 July 2013	David Brossard	Cleaned up the documents and comments.
WD15	02 September 2013	David Brossard	Fixed document based on feedback from Steven Legg: • The naming of Attributes vs. Category in section 5.2.2 • Fixed the name of ObligationOrAdvice in section 5.2.6 Also fixed subjective line in introduction based on email xacml-comment from David Webber.
WD16	17 March 2014	David Brossard	• Fixed issues with special numerical values: based on input from the XACML TC, special values (NaN, Inf, -0) are now excluded • Rewrote section 3.4.2 and added reference to 3.4.1 • Added a section defining the shorthand notation for standard XACML categories • Added normative reference to XACML 3.0 standard • Added optional category objects for all default categories in XACML 3.0 instead of the 4 most common ones only. • Updated example in 4.2.4.1 • Fixed the Transport section to reference the REST profile. • Fixed broken samples • Added references to IEEE 754-2008 rather than Javascript for the special numerical values • Fixed the Content section to include the namespaces requirement • Fixed the default value for

			XPathVersion to be in accordance with **[XACML30]**. • Added the MissingAttributeValue object definition.
WD17	14 April 2014	David Brossard	• Updated the profile title per conversation on the XACML TC list • Updated section 3.2.1 on object names in JSON • Fixed broken reference to 3.3.1 in 3.3.2 • Updated the inference rule for double and integers to remove any doubt as to the potential datatypes • Fixed wording in section 4.2.1 (much like vs. just like) • Simplified the wording of section 4.2.2.2 • Updated the example in section 4.2.2.3 • Changed the shorthand name subject to access-subject to be consistent • Added the Indeterminate behavior for invalid numerical values • Fixed the base 64 encoding example in section 4.2.3.3. • Fixed the examples (wrong attribute names, missing parents, missing curly braces) • Changed the MS Word quotes into proper quotes
WD18	22 April 2014	David Brossard	• Changed the shorthand names to use Title Case instead. resource becomes Resource, access-subject becomes AccessSubject, and so on. • Updated the XPathCategory so that one can use the category shorthand notation as a valid value instead.
WD19	23 October	David Brossard	• Introduced formatting changes based on feedback received on xacml-comment • Fixed section 6 content-type and accept • Fixed the wording on StatusCode • Added captions to tables

545