

Request / Response Interface based on JSON and HTTP for XACML 3.0 Version 1.0

Committee Specification Draft 02

17 October 2013

Specification URIs

This version:

<http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csd02/xacml-json-http-v1.0-csd02.doc>
(Authoritative)
<http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csd02/xacml-json-http-v1.0-csd02.html>
<http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csd02/xacml-json-http-v1.0-csd02.pdf>

Previous version:

<http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csprd01/xacml-json-http-v1.0-csprd01.doc>
(Authoritative)
<http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csprd01/xacml-json-http-v1.0-csprd01.html>
<http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csprd01/xacml-json-http-v1.0-csprd01.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*. 22 January 2013. OASIS Standard. <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-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.

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

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

Citation format:

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

[xacml-json-http-v1.0]

Request / Response Interface based on JSON and HTTP for XACML 3.0 Version 1.0. 17 October 2013. OASIS Committee Specification Draft 02. <http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csd02/xacml-json-http-v1.0-csd02.html>.

Notices

Copyright © OASIS Open 2013. All Rights Reserved.

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

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

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

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

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

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

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

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

Table of Contents

1	Introduction.....	6
1.1	Terminology.....	6
1.2	Normative References.....	6
1.3	Non-Normative References.....	7
2	Vocabulary.....	8
3	Overview of the translation mechanisms.....	9
3.1	Assumed default values.....	9
3.2	Object names.....	9
3.3	Object cardinality.....	9
3.4	Data Types.....	9
3.4.1	Supported Data Types.....	9
3.4.2	Arrays of values.....	10
3.4.3	The xpathExpression Datatype.....	10
3.4.4	Special numeric values.....	11
3.5	Example.....	12
4	The XACML request.....	13
4.1	Class Diagram.....	13
4.2	Representation of the XACML request in JSON.....	13
4.2.1	The Request object representation.....	13
4.2.2	The Category object representation.....	14
4.2.3	The Content Object representation.....	15
4.2.4	The Attribute Object representation.....	16
4.2.5	The MultiRequests object representation.....	17
4.2.6	The RequestReference object representation.....	17
5	The XACML response.....	19
5.1	Class Diagram.....	19
5.2	Representation of the XACML response in JSON.....	19
5.2.1	The Response object representation.....	19
5.2.2	The Result object representation.....	19
5.2.3	The Status object representation.....	20
5.2.4	The StatusCode object representation.....	20
5.2.5	The Obligations object representation.....	20
5.2.6	The AssociatedAdvice object representation.....	21
5.2.7	The ObligationOrAdvice object representation.....	21
5.2.8	The AttributeAssignment object representation.....	21
5.2.9	The Attributes object representation.....	21
5.2.10	The PolicyIdentifier object representation.....	21
5.2.11	The IdReference object representation.....	21
6	Transport.....	23
6.1	Transport Security.....	23
7	IANA Registration.....	24
7.1	Media Type Name.....	24
7.2	Subtype Name.....	24

7.3 Required Parameters.....	24
7.4 Optional Parameters.....	24
7.5 Encoding Considerations.....	24
7.6 Security Considerations.....	24
7.7 Interoperability Considerations	24
7.8 Applications which use this media type	24
7.9 Magic number(s).....	24
7.10 File extension(s)	24
7.11 Macintosh File Type Code(s).....	25
7.12 Intended Usage	25
8 Examples.....	26
8.1 Request Example	26
8.2 Response Example.....	27
9 Conformance	28
Appendix A. Acknowledgements.....	29
Appendix B. Revision History	30

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] OASIS Committee Draft 03, *XACML v3.0 Multiple Decision Profile Version 1.0*, 11 March 2010. <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-multiple-v1-spec-cd-03-en.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/>

- 45 **[XML]** Bray, Tim, et.al. eds, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*,
46 W3C Recommendation 26 November 2008, available at
47 <http://www.w3.org/TR/2008/REC-xml-20081126/>
- 48 **[XMLDatatypes]** Biron, Paul et al. Eds, *XML Schema Part 2: Datatypes Second Edition*, W3C
49 Recommendation 28 October 2004, available at
50 <http://www.w3.org/TR/xmlschema-2/>
- 51 **[XPATH]** James Clark and Steve DeRose, XML Path Language (XPath), Version 1.0, W3C
52 Recommendation 16 November 1999. Available at: <http://www.w3.org/TR/xpath>

53 **1.3 Non-Normative References**

- 54 **[XACMLREST]** R. Sinnema, *REST Profile of XACML v3.0 Version 1.0*, 24 April 2012
55 [https://www.oasis-open.org/committees/download.php/45829/xacml-rest-v1.0-](https://www.oasis-open.org/committees/download.php/45829/xacml-rest-v1.0-wd02.doc)
56 [wd02.doc](https://www.oasis-open.org/committees/download.php/45829/xacml-rest-v1.0-wd02.doc).
- 57 **[HTTP]** *Hypertext Transfer Protocol*. June 1999. IETF RFC 2616.
58 <http://tools.ietf.org/html/rfc2616>
- 59 **[HTTPS]** *HTTP over TLS*. May 2000. IETF RFC 2818. <http://tools.ietf.org/html/rfc2818>
- 60
- 61 **[BASE64]** *The Base16, Base32, and Base64 Data Encodings*. October 2006. IETF RFC
62 4648. <http://tools.ietf.org/html/rfc4648>
- 63

64 **2 Vocabulary**

65 **{Non-normative}**

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

68 3 Overview of the translation mechanisms

69 3.1 Assumed default values

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

73 The user should refer to the XACML 3.0 specification document for a normative definition of the request
74 and response elements.

75 3.2 Object names

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

78 3.3 Object cardinality

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

81 The class diagram in 4.1. Class Diagram states the cardinality and relationship between objects.
82

83 3.4 Data Types

84 This section defines how data-types are represented and handled in the JSON representation. Chapter
85 10, section 10.2.7 in the XACML 3.0 specification as well as section A.2 list the data-types that are
86 defined in XACML. These are listed in the table below in section 3.4.1. It lists the shorthand value that
87 MAY be used when creating a XACML attribute in the JSON representation.

88 3.4.1 Supported Data Types

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

91 It is also possible to omit for certain XACML data types the JSON property `dataType` when it can safely
92 be inferred from the value of the attribute.

XACML data type identifier	JSON shorthand type code	Mapping / Inference Rule
http://www.w3.org/2001/XMLSchema#string	string	JavaScript "String"
http://www.w3.org/2001/XMLSchema#boolean	boolean	JavaScript "Boolean"
http://www.w3.org/2001/XMLSchema#integer	integer	JavaScript "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	JavaScript "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

- 93 For all of the XACML data types that cannot be inferred from the value, the following MUST be observed:
- 94 • The JSON `DataType` property MUST be specified and the value expressed in the XACML string
- 95 representation of the value.
- 96 • JavaScript code may choose to parse the XACML string values into internal numeric
- 97 representations for internal use, such as for `DateTime` or `*Duration` values, but the JSON
- 98 transport representation must always express the value in the XACML string representation of the
- 99 XACML data type.

100 3.4.2 Arrays of values

101 In the case of an array of values, and if the `DataType` member is not specified, it may not be possible to

102 infer the `DataType` until all the values have been inspected.

103 For example, an array that contains integers except for the last value which is a double e.g. [4,3,5,2.5] is

104 in fact an array of double values.

105 An array of values that are all integer is inferred to be an array of values of the integer datatype rather

106 than double.

107 3.4.3 The `xpathExpression` Datatype

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

109 the following properties:

Attribute	Type	Mandatory/Optional	Default value
<code>XPathCategory</code>	URI	Mandatory	None
<code>Namespaces</code>	Array of	Optional	None

	NamespaceDeclaration		
XPath	String	Mandatory	None

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

113 A NamespaceDeclaration object contains the following properties:

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

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

118 The Namespaces array MUST contain a NamespaceDeclaration object for each of the namespace
 119 prefixes used by the XPath expression. The Namespaces array MAY contain additional
 120 NamespaceDeclaration objects for namespace prefixes that are not used by the XPath expression. There
 121 SHALL NOT be two or more NamespaceDeclaration objects for the same namespace prefix.

122 3.4.3.1 Example

123 {Non-normative}

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

- 126 • As XML:

```
127
128 <Attribute xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
129   AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector">
130   <AttributeValue xmlns:md="urn:example:med:schemas:record"
131     XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
132     DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
133   >md:record/md:patient/md:patientDoB</AttributeValue>
134 </Attribute>
```

- 135 • As JSON:

```
136   "Attribute": {
137     "Id" : "urn:oasis:names:tc:xacml:3.0:content-selector",
138     "DataType" : "xpathExpression",
139     "Value" : {
140       "XPathCategory" : "urn:oasis:names:tc:xacml:3.0:attribute-category:resource",
141       "Namespaces" : [{
142         "Namespace" : "urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
143       }],
144       {
145         "Prefix" : "md",
146         "Namespace" : "urn:example:med:schemas:record"
147       }
148     },
149     "XPath" : "md:record/md:patient/md:patientDoB"
150   }
}
```

151 3.4.4 Special numeric values

152 The following special numeric values MUST also be handled

- 153 • JavaScript NaN -> "NaN"

- 154 • JavaScript positive infinity -> "INF"
- 155 • JavaScript negative infinity -> "-INF"
- 156 • JavaScript positive zero -> 0
- 157 • JavaScript negative zero -> 0 (-0 is a valid text representation, but the sign will be ignored by
- 158 XACML in comparisons, per XML #double defined in [XMLDatatypes](#))

3.5 Example

{Non-normative}

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

Equivalent examples	
Attribute representation explicitly stating the data-type	Attribute representation omitting the data-type
<pre>"Attribute": { "Id" : "document-id" "DataType": "integer" "Value" : 123 }</pre>	<pre>"Attribute": { "Id" : "document-id" "value" : 123 }</pre>

- 162
- 163 In the latter example where the JSON `DataType` property is omitted, the JSON translation must use the
- 164 closest data type, in this case integer.

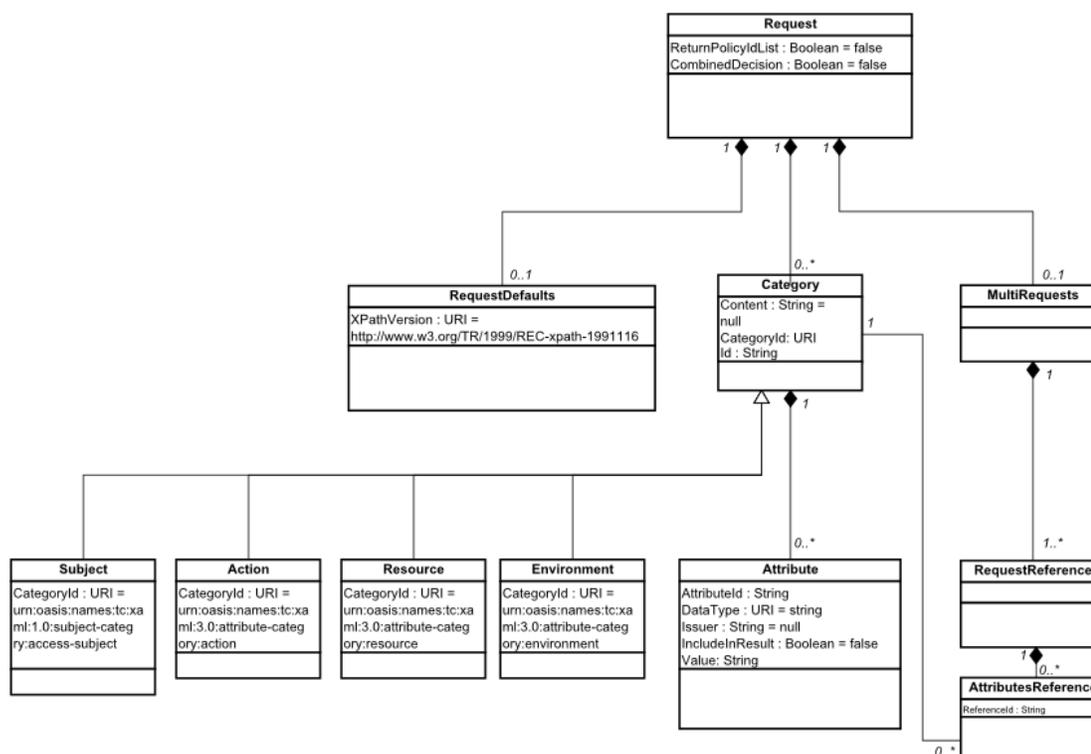
165 4 The XACML request

166 4.1 Class Diagram

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

169 The key differences are:

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



173
174

175 4.2 Representation of the XACML request in JSON

176 4.2.1 The Request object representation

177 The JSON object name for the request will be `Request`

178 The `Request` object contains the following properties:

- 179 • `ReturnPolicyIdList` of type Boolean
- 180 • `CombinedDecision` of type Boolean
- 181 • `XPathVersion` of type String

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

Attribute	Type	Default value
ReturnPolicyIdList	Boolean	False. The ReturnPolicyIdList can be omitted in the JSON representation.
CombinedDecision	Boolean	False. The ReturnPolicyIdList can be omitted in the JSON representation.
XPathVersion	String	http://www.w3.org/TR/1999/REC-xpath-19991116 . The XPathVersion can be omitted in the JSON representation.

184

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

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

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

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

196 4.2.1.1 Example

197 {Non-normative}

```
198 "Request": {
199     " XPathVersion" : "http://www.w3.org/TR/1999/REC-xpath-19991116"
200 }
201 }
```

202

203 4.2.2 The Category object representation

204 The JSON `Category` object contains the following properties:

Attribute	Type	Mandatory/Optional	Default value
CategoryId	anyURI	Mandatory	None – the identifier used in the XML representation shall be used in its JSON representation except where shorthand notations have been defined.
Id	String	Optional	The Id property is optional in the JSON representation. There is no default, assumed, value 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	The value of the Content property must be escaped or encoded as explained in 4.2.3.

--	--	--	--

205

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

- 207 • Attribute: this is an array of Attribute objects as defined in 4.2.4 The Attribute Object
208 representation

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

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

211 4.2.2.1 Default Category objects

212 To simplify the JSON representation, this profile also defines four optional default objects that are
213 semantically equivalent to the `Category` object. These default objects assume a default value for the
214 `CategoryId` property so that it need not be explicitly written. The following table summarizes these four
215 objects and the default values

Name	Default value for the child <code>Category</code> property
Subject	urn:oasis:names:tc:xacml:1.0:subject-category:access-subject
Action	urn:oasis:names:tc:xacml:3.0:attribute-category:action
Resource	urn:oasis:names:tc:xacml:3.0:attribute-category:resource
Environment	urn:oasis:names:tc:xacml:3.0:attribute-category:environment

216 In order to facilitate the creation of multiple decisions, `Subject`, `Action`, `Resource`, and `Environment`
217 can also be an array instead of single-valued.

218 4.2.2.2 Example

219 {Non-normative}

```

220 {
221   "Request": {
222     "Category": [{
223       "CategoryId": "custom-category",
224       "Attribute": [...]
225     },
226     {
227       "CategoryId": "another-custom-cat",
228       "Attribute": [...]
229     }
230   ]
231   "Subject": {
232     "Attribute": [...]
233   }
234 }
235 }

```

236

237 4.2.3 The Content Object representation

238 There are two possible ways to represent the XML content of a XACML request in the JSON
239 representation: [XML escaping](#) or [Base64 encoding](#). Both ways are exclusive one of another.

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, 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	
Issuer	String	Optional	Null
DataType	URI	Optional	The <code>DataType</code> value can be omitted in the JSON representation. Its default value will be <code>http://www.w3.org/2001/XMLSchema#string</code> unless it can be safely assumed according to the rules set in 3.4.1 Supported Data . In the case of an array of values, the <code>DataType</code> cannot be inferred from the values in the array and the <code>DataType</code> JSON property must be specified.
IncludeInResult	Boolean	Optional	False.

276 4.2.4.1 Example

277 {Non-normative}

```
278     "Attribute": [{
279         "Id": "urn:oasis:names:tc:xacml:2.0:subject:role",
280         "Value" : ["manager","administrator"]
281     }]
```

282 4.2.5 The MultiRequests object representation

283 The `MultiRequests` object is optional in the JSON representation of XACML. Its purpose is to support
284 the Multiple Decision Profile [\[XACMLMDP\]](#).

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

287 4.2.6 The RequestReference object representation

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

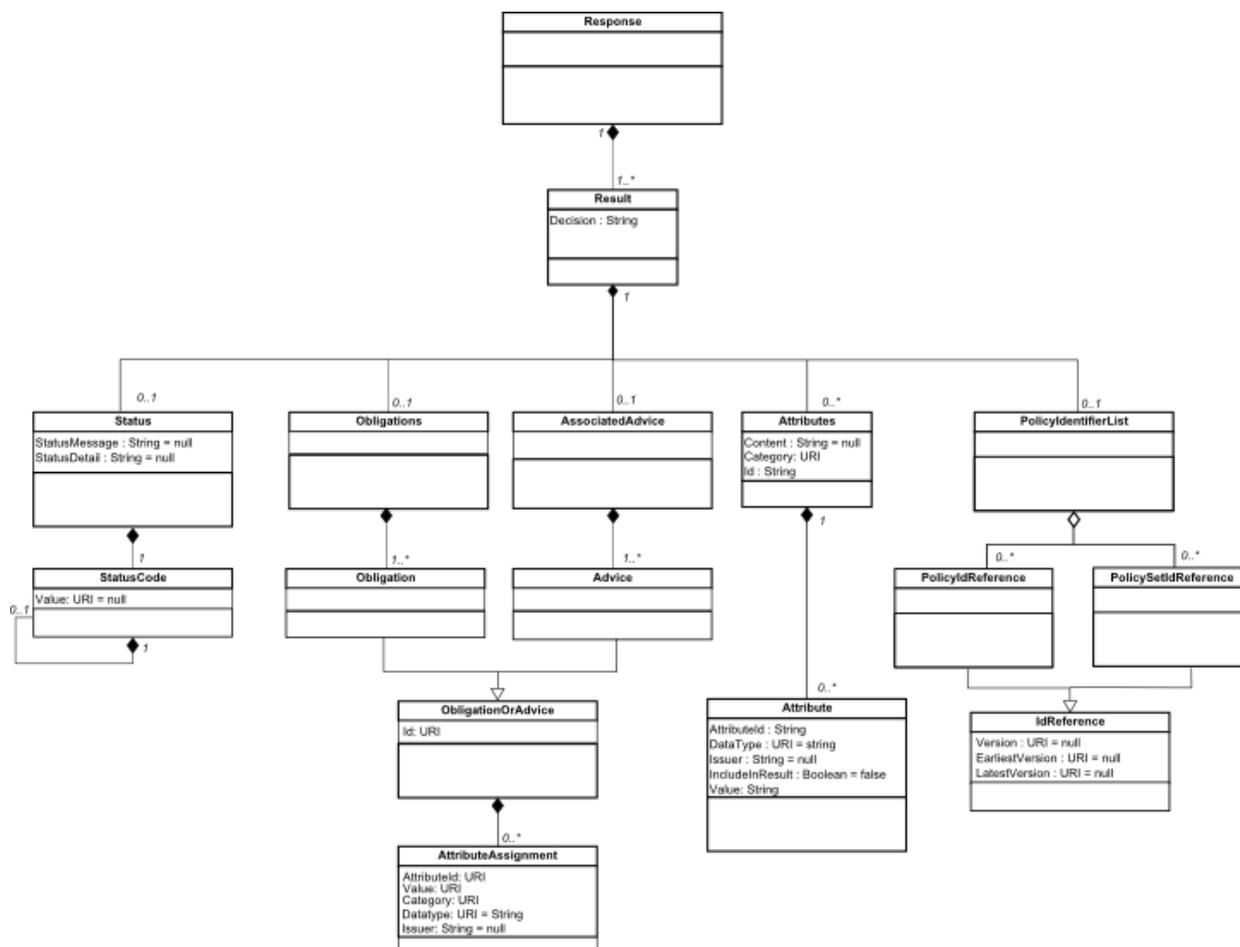
290 4.2.6.1 Non-normative example

291 {

```
292 MultiRequests : {
293     "RequestReference": [{
294         "ReferenceId" : ["foo1","bar1"]
295     },
296     {
297         "ReferenceId" : ["foo2","bar1"]
298     },
299     {
300         "ReferenceId" : ["foo3","bar1"]
301     }]
302 }
303 }
```

304 5 The XACML response

305 5.1 Class Diagram



306

307 5.2 Representation of the XACML response in JSON

308 5.2.1 The Response object representation

309 The `Response` property in its JSON representation will contain an array of `Result` objects. The `Result`
 310 object representation is detailed hereafter. The array **MUST** contain at least one `Result` object and is
 311 unbounded.

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

314 5.2.2 The Result object representation

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

Property name	Type	Mandatory/Optional	Default value
Decision	String	Mandatory	None – in addition there are only 4 valid values which are “Permit”, “Deny”, “NotApplicable”, and

			"Indeterminate". The values are case-sensitive.
--	--	--	---

316 In addition to the aforementioned properties, the `Result` object also contains the following objects:

- 317 • `Status`: this object is optional.
- 318 • `Obligations`: this object is optional.
- 319 • `AssociatedAdvice`: this object is optional.
- 320 • `Attributes`: this object is optional. It can be single-valued or an array of [Category](#) objects.
- 321 • `PolicyIdentifierList`: this object is optional.

322 5.2.3 The Status object representation

323 The `Status` object in JSON will contain the following properties:

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

324 In addition to the above properties, the `Status` object in JSON also contains a `StatusCode` object
325 detailed hereafter.

326 `StatusDetail` can contain arbitrary XML as well. In the case that `StatusDetail` does contain XML,
327 the XML content must be escaped using the same technique as specified in 4.2.3 The Content Object
328 representation.

329 5.2.4 The StatusCode object representation

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

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

331 In addition, the `StatusCode` object may contain a sequence of `StatusCode` objects – hence potentially
332 creating a recursive nesting of `StatusCode` objects.

333 5.2.4.1 Example

334 {Non-normative}

```
335 {
336   "Response": [{
337     "Decision": "Permit"
338     "Status": {
339       "StatusCode": {
340         "Value" : "http://foo.bar"
341       }
342     }
343   }]
344 }
```

345 5.2.5 The Obligations object representation

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

348 **5.2.6 The AssociatedAdvice object representation**

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

351 **5.2.7 The ObligationOrAdvice object representation**

352 The `ObligationOrAdvice` object contains the following properties in its JSON representation:

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

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

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

358 **5.2.8 The AttributeAssignment object representation**

359 The `AttributeAssignment` object contains the following properties in its JSON representation:

Property name	Type	Mandatory/Optional	Default value
AttributeId	URI	Mandatory	None.
Value	Variable	Mandatory	None
Category	URI	Optional	None
DataType	URI	Optional	String
Issuer	String	Optional	None

360

361 **5.2.9 The Attributes object representation**

362 The JSON representation of the `Attributes` object in a XACML response respects the representation
363 defined in 4.2.2 The `Category` object representation.

364 **5.2.10 The PolicyIdentifier object representation**

365 The `PolicyIdentifier` object contains 2 properties in its JSON representation:

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

366

367 **5.2.11 The IdReference object representation**

368 The `IdReference` object representation contains the following properties in its JSON representation:

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.

369

370 6 Transport

371 The XACML request represented in its JSON format MAY be carried from a PEP to a PDP via an HTTP
372 **[HTTP]** POST request.

373 HTTP Headers which may be used are:

- 374 • Content-Type: application/json
- 375 • Accept: application/json

376 The REST profile of XACML [XACMLREST] defines means of sending a XACML request to a PDP and
377 how a response is returned.

378 6.1 Transport Security

379 **{Non-normative}**

380 The use of SSL/TLS **Error! Reference source not found.** is RECOMMENDED to protect requests and
381 responses as they are transferred across the network.

382 7 IANA Registration

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

384 7.1 Media Type Name

385 application

386 7.2 Subtype Name

387 xacml+json

388 7.3 Required Parameters

389 None.

390 7.4 Optional Parameters

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

395 7.5 Encoding Considerations

396 Same as for application/xml [\[RFC4627\]](#).

397 7.6 Security Considerations

398 Per their specification, application/xacml+json typed objects do not contain executable content.
399 XACML requests and responses contain information which integrity and authenticity are important.
400 To counter potential issues, the publisher may use the transport layer's security mechanisms to secure
401 xacml+json typed objects when they are in transit. For instance HTTPS, offer means to ensure the
402 confidentiality, authenticity of the publishing party and the protection of the request / response in transit.

403 7.7 Interoperability Considerations

404 XACML 3.0 uses the urn:oasis:names:tc:xacml:3.0:core:schema:wd-17 XML namespace URI. XACML
405 2.0 uses the urn:oasis:names:tc:xacml:2.0:policy XML namespace URI.

406 7.8 Applications which use this media type

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

410 7.9 Magic number(s)

411 Per [\[RFC4627\]](#), this section is not applicable.

412 7.10 File extension(s)

413 Per [\[RFC4627\]](#), .json.

414 **7.11 Macintosh File Type Code(s)**

415 Text

416 **7.12 Intended Usage**

417 Common

418 8 Examples

419 {Non-normative}

420 8.1 Request Example

421 {Non-normative}

422 The following is a sample XACML request expressed in JSON.

```
423 {
424     "Request" : {
425         "Subject" : {
426             "Attribute": [
427                 {
428                     "Id" : "subject-id",
429                     "Value" : "Andreas"
430                 },
431                 {
432                     "Id" : "location",
433                     "Value" : "Gamla Stan"
434                 }
435             ]
436         },
437         "Action" : {
438             "Attribute":
439                 {
440                     "Id" : "action-id",
441                     "Value" : "http://www.xacml.eu/buy",
442                     "DataType" : "anyURI"
443                 }
444         },
445         "Resource" : {
446             "Attribute": [
447                 {
448                     "Id" : "book-title",
449                     "Value" : "Learn German in 90 days"
450                 },
451                 {
452                     "Id" : "currency",
453                     "Value" : "SEK"
454                 },
455                 {
456                     "Id" : "price",
457                     "Value" : 123.34
458                 }
459             ]
460         }
461     }
462 }
```

```
460         }
461     }
462 }
```

463 8.2 Response Example

464 {Non-normative}

465 The following is a sample XACML response expressed in JSON.

```
466 {
467     "Response" : [{
468         "Decision" : "Permit"
469     }
470 ]
471 }
```

472 **9 Conformance**

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

476 **Appendix A. Acknowledgements**

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

479 **Participants:**

480 Steven Legg, ViewDS
481 Rich Levinson, Oracle
482 Hal Lockhart, Oracle
483 Bill Parducci,
484 Erik Rissanen, Axiomatics
485 Anil Saldhana, Red Hat
486 Remon Sinnema, EMC
487 Danny Thorpe, Dell
488 Paul Tyson, Bell Helicopters
489

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: <ul style="list-style-type: none"> • 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.