



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

Committee Specification Draft ~~0203~~ /
Public Review Draft ~~0203~~

~~17 October 2013~~

15 May 2014

Specification URIs

This 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>

Previous version:

<http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csprd02/xacml-json-http-v1.0-csprd02.doc>
(Authoritative)
<http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csprd02/xacml-json-http-v1.0-csprd02.html>
<http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csprd02/xacml-json-http-v1.0-csprd02.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 previously titled *Request / Response Interface based on JSON and HTTP for XACML 3.0 Version 1.0.*](#)

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 <https://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 (<https://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 Profile of XACML 3.0 Version 1.0. 17 October 2013. Edited by David Brossard. 15 May 2014. OASIS Committee Specification Draft 0203 / Public Review Draft 03. http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/csprd03/xacml-json-http-v1.0-csprd03.html02-.](#) [Latest version: http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/xacml-json-http-v1.0.html.](#)

Notices

Copyright © OASIS Open 2013³⁴. 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 <https://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	Objects.....	9
3.2.1	Object names.....	9
3.2.2	Object order.....	9
3.2.3	Object cardinality.....	9
3.3	Data Types.....	9
3.3.1	Supported Data Types.....	9
3.3.2	Arrays of values.....	11
3.3.3	The xpathExpression Datatype.....	11
3.3.4	Special numeric values.....	12
3.4	Example.....	12
4	The XACML request.....	14
4.1	Class Diagram.....	14
4.2	Representation of the XACML request in JSON.....	14
4.2.1	The Request object representation.....	14
4.2.2	The Category object representation.....	15
4.2.3	The Content Object representation.....	17
4.2.4	The Attribute Object representation.....	18
4.2.5	The MultiRequests object representation.....	19
4.2.6	The RequestReference object representation.....	20
5	The XACML response.....	21
5.1	Class Diagram.....	21
5.2	Representation of the XACML response in JSON.....	21
5.2.1	The Response object representation.....	21
5.2.2	The Result object representation.....	21
5.2.3	The Status object representation.....	22
5.2.4	The MissingAttributeDetail object.....	22
5.2.5	The StatusCode object representation.....	23
5.2.6	The Obligations object representation.....	24
5.2.7	The AssociatedAdvice object representation.....	24
5.2.8	The ObligationOrAdvice object representation.....	24
5.2.9	The AttributeAssignment object representation.....	24
5.2.10	The Attributes object representation.....	25
5.2.11	The PolicyIdentifier object representation.....	25
5.2.12	The IdReference object representation.....	25
6	Transport.....	26
6.1	Transport Security.....	26

7	IANA Registration	27
7.1	Media Type Name	27
7.2	Subtype Name	27
7.3	Required Parameters.....	27
7.4	Optional Parameters.....	27
7.5	Encoding Considerations.....	27
7.6	Security Considerations.....	27
7.7	Interoperability Considerations	27
7.8	Applications which use this media type	27
7.9	Magic number(s).....	27
7.10	File extension(s)	27
7.11	Macintosh File Type Code(s).....	28
7.12	Intended Usage	28
8	Examples.....	29
8.1	Request Example	29
8.2	Response Example.....	30
9	Conformance	31
Appendix A.	Acknowledgements.....	32
Appendix B.	Revision History	33

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 | [\[XACML30\]](#) OASIS Standard, "eXtensible Access Control Markup Language (XACML)
46 | [Version 3.0](#)", April 2010. [spec-en.doc](http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-
47 | <a href=)
- 48 | **[XML]** Bray, Tim, et.al. eds, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*,
49 | W3C Recommendation 26 November 2008, available at
50 | <http://www.w3.org/TR/2008/REC-xml-20081126/>
- 51 | **[XMLDatatypes]** Biron, Paul et al. Eds, *XML Schema Part 2: Datatypes Second Edition*, W3C
52 | Recommendation 28 October 2004, available at
53 | <http://www.w3.org/TR/xmlschema-2/>
- 54 | **[XPATH]** James Clark and Steve DeRose, XML Path Language (XPath), Version 1.0, W3C
55 | Recommendation 16 November 1999. Available at: <http://www.w3.org/TR/xpath>
- 56 | [\[IEEE754\]](#) Institute of Electrical and Electronics Engineers, "Standard for Floating-Point
57 | [Arithmetic](#)", IEEE Standard 754, August 2008.
58 |

59 | 1.3 Non-Normative References

- 60 | **[XACMLREST]** R. Sinnema, *REST Profile of XACML v3.0 Version 1.0*, 24 April 2012
61 | [wd02.doc](https://www.oasis-open.org/committees/download.php/45829/xacml-rest-v1.0-
62 | <a href=).
- 63 | **[HTTP]** *Hypertext Transfer Protocol*. June 1999. IETF RFC 2616.
64 | <http://tools.ietf.org/html/rfc2616>
- 65 | **[HTTPS]** *HTTP over TLS*. May 2000. IETF RFC 2818. <http://tools.ietf.org/html/rfc2818>
66 |
- 67 | **[BASE64]** *The Base16, Base32, and Base64 Data Encodings*. October 2006. IETF RFC
68 | 4648. <http://tools.ietf.org/html/rfc4648>
69 |

70 **2 Vocabulary**

71 **{Non-normative}**

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

74 3 Overview of the translation mechanisms

75 3.1 Assumed default values

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

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

81 [3.2 Objects](#)

82 ~~3.1.13.2.1~~ Object names

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

85 [The following XML elements and attributes have been renamed:](#)

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

95 [3.2.2 Object order](#)

96 [The order of the objects and values in XACML does not matter. Therefore, the order of objects and](#)
97 [values in the serialized form \(JSON\) does not matter.](#)

98 ~~3.1.23.2.3~~ Object cardinality

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

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

102 ~~3-23.3~~ Data Types

103 This section defines how data-types are represented and handled in the JSON representation. Chapter
104 10, section 10.2.7 in the XACML 3.0 specification as well as section A.2 list the data-types that are
105 defined in XACML. These are listed in the table below in section 3.3.1. It lists the shorthand value that
106 MAY be used when creating a XACML attribute in the JSON representation.

107 ~~3.2.13.3.1~~ Supported Data Types

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

110 It is also possible to omit for certain XACML data types the JSON property `DataType` when it can safely
111 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 "JSON String"
http://www.w3.org/2001/XMLSchema#boolean	boolean	JavaScript "JSON Boolean"
http://www.w3.org/2001/XMLSchema#integer	integer	JavaScript "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	JavaScript "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

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

- 113
- 114 • The JSON `DataType` property MUST be specified and the value expressed in the XACML string representation of the value.
 - 115 • [JavaScript Implementation-specific \(e.g. Javascript\)](#) code may choose to parse the XACML string values into internal numeric representations for internal use, such as for `DateTime` or `*Duration` values, but the JSON transport representation must always express the value in the [serialized](#) XACML string representation of the XACML data type.
- 116
- 117
- 118

119 **3.2.23.3.2 Arrays of values**

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

122 Inference for an array of values works according to the inference rules as set in 3.3.1. For example, an
123 array that contains integers except for the last value which is a double e.g. [4,3,5,2.5] is in fact an array of
124 double values.

125 An array of values that are all integer is inferred to be an array of values of the integer datatype rather
126 than double.

127 . If a given data type cannot be inferred and there is no `DataType` member specified then the array of
128 values will be considered as an array of string.

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

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

133 **3.2.33.3.3 The xpathExpression Datatype**

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

Attribute	Type	Mandatory/Optional	Default value
XPathCategory	URI	Mandatory	<u>None. The shorthand notation defined in 4.2.2.1. Shorthand notation for standard XACML categories can be used as values here.</u>
Namespaces	Array of NamespaceDeclaration	Optional	None
XPath	String	Mandatory	None

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

139 A NamespaceDeclaration object contains the following properties:

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

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

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

148 **3.2.3-13.3.3.1 Example**

149 {Non-normative}

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

- 152 • As XML:

```
153 <Attribute xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"  
154   AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector">  
155   <AttributeValue xmlns:md="urn:example:med:schemas:record"  
156     XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"  
157     DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"  
158     >md:record/md:patient/md:patientDoB</AttributeValue>  
159 </Attribute>
```

- 161 • As JSON:

```
162   {"Attribute": {  
163     "id": "AttributeId": "urn:oasis:names:tc:xacml:3.0:content-  
164     selector",  
165     "DataType": "xpathExpression",  
166     "Value": {  
167       "XPathCategory":  
168       "urn:oasis:names:tc:xacml:3.0:attribute-category:resource",  
169       "Namespaces": [{  
170         "Namespace":  
171         "urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"  
172       }],  
173       {  
174         "Prefix": "md",  
175         "Namespace": "urn:example:med:schemas:record"  
176       }],  
177       "XPath": "md:record/md:patient/md:patientDoB"  
178     }  
179   }  
180 }
```

180 3.2.43.3.4 Special numeric values

181 The following special numeric values [are not supported by the profile. Should the request contain such](#)
182 [values, the Policy Decision Point MUST reply with an Indeterminate with a status value of](#)
183 [urn:oasis:names:tc:xacml:1.0:status:syntax-error as defined in Appendix B, section 8 of](#)
184 [\[XACML30\]also be handled.](#)

185 [Additional behavior of the PDP when returning urn:oasis:names:tc:xacml:1.0:status:syntax-](#)
186 [error is specified in sections 5.57 and B.8 of \[XACML30\]JavaScript.](#)

- 187 • [IEEE 754-2008 NaN](#) → `("NaN")`
- 188 • [JavaScript|IEEE 754-2008 positive infinity](#) → `("INF")`
- 189 • [JavaScript|IEEE 754-2008 negative infinity](#) → `("-INF")`
- 190 • [JavaScript positive zero](#) → `0`
- 191 • [JavaScript negative zero](#) → `0` (`-0` is a valid text representation, but the sign will be ignored by
192 XACML in comparisons, per XML #double defined in)
- 193 • [IEEE 754-2008 negative zero \(-0\)](#)

194 3-33.4 Example

195 {Non-normative}

196 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": { "IdAttributeId" : "document-id" "DataType" : "integer" "Value" : 123 +}} </pre>	<pre> {"Attribute": { "IdAttributeId" : "document- id" "Value" : 123 +}} </pre>

197

198

199

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

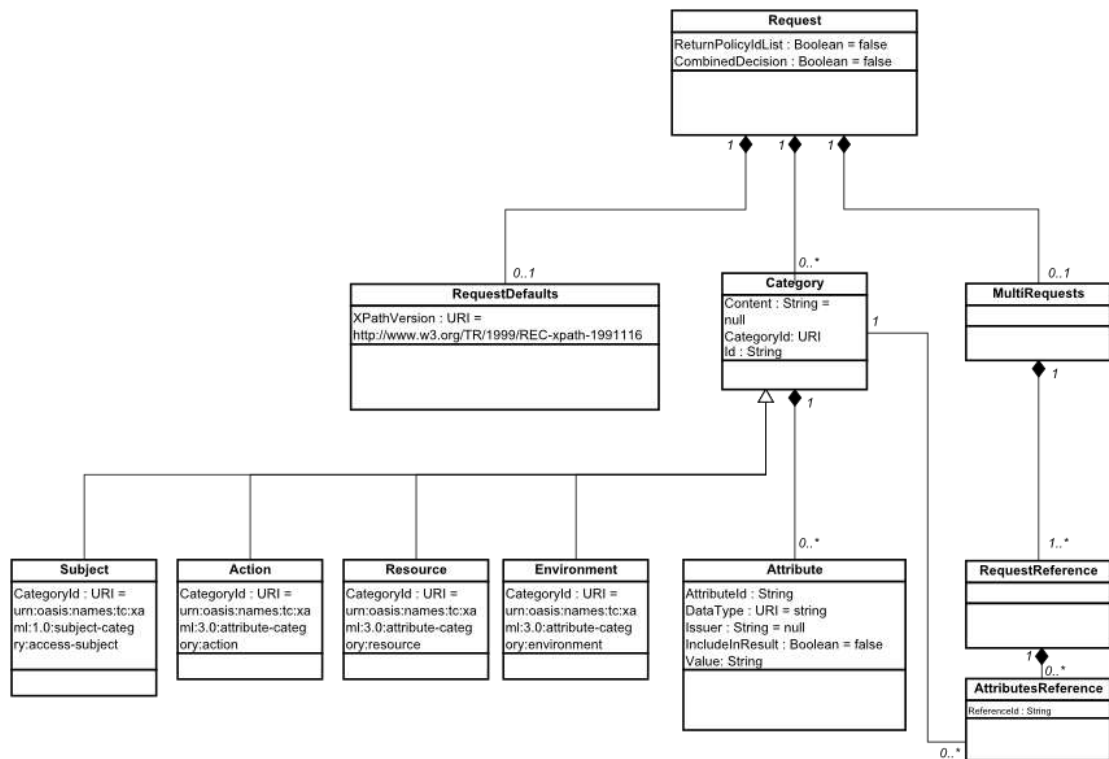
200 4 The XACML request

201 4.1 Class Diagram

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

204 The key differences are:

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



208
209

210 4.2 Representation of the XACML request in JSON

211 4.2.1 The Request object representation

212 The JSON object name for the request **WILLMUST** be `Request_`.

213 The `Request` object contains the following properties:

- 214 • `ReturnPolicyIdList` of type Boolean
- 215 • `CombinedDecision` of type Boolean
- 216 • `XPathVersion` of type String

217 These properties are represented as members. The JSON representation assumes the following default
 218 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. There is no default value. The attribute is optional. It is REQUIRED if the XACML request contains XPath expressions.

219
 220 In addition to these properties, the Request element also contains the following objects:

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

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

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

232 4.2.1.1 Example

```
233 {Non-normative}
234 {"Request": {
235   "XPathVersion": "http://www.w3.org/TR/1999/REC-xpath-19991116"
236 }
237 }
238
```

239 4.2.2 The Category object representation

240 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 MUST be used in its JSON representation except where shorthand notations have been defined – see 4.2.2.1 <u>Shorthand notation for standard XACML categories.</u>
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	None . The value of the <code>Content</code> property must be escaped or encoded as explained in 4.2.3.

241

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

- 243 • Attribute: this is an array of `Attribute` objects as defined in 4.2.4 The Attribute Object
244 representation

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

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

248 [4.2.2.1 Shorthand notation for standard XACML categories](#)

249 The following table defines a shorthand notation for the standard categories defined in [\[XACML30\]](#).

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

250 [The shorthand notation MAY be used as described in section 4.2.2.2 and section 4.2.2.](#)

251 [4.2.2.14.2.2.2 Default Category objects](#)

252 To simplify the JSON representation, this profile also defines ~~four~~ optional default objects that are
253 semantically equivalent to the `Category` object. These default objects assume a default value for the
254 `CategoryId` property so that it need not be explicitly written. [The object names correspond to the short
255 names as defined in 4.2.2.1 Shorthand notation for standard XACML categories](#)~~The following table
256 summarizes these four 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

257 ~~In order to facilitate~~[Note that JSON does not allow for the duplication of objects that bear the creation of
258 multiple decisions, `Subject`, `Action`, `Resource`, same name, e.g. "AccessSubject" and
259 "AccessSubject". Consequently, the optional default objects \(based on 4.2.2.1 Shorthand notation for](#)

260 [standard XACML categoriesEnvironment](#)) can also be an array instead of single-valued [in order to](#)
261 [cater for multiple decision requests as defined in \[XACMLMDP\]](#).

262 4.2.2.24.2.2.3 Example

```
263 {Non-normative}
264 {
265     "Request": {
266         "Category": [{
267             "CategoryId": "custom-category",
268             "Attribute": [...]
269         }],
270         {
271             "CategoryId": "another-custom-cat",
272             "Attribute": [...]
273         }
274     }],
275     "AccessSubject":{
276         "Attribute": [...]
277     }
278 }
279 },
280     "Action":[
281         {
282             "Attribute": [...]
283         }
284     ],
285     }
286 }
287 }
```

288 4.2.3 The Content Object representation

289 There are two possible ways to represent the XML content of a XACML request in the JSON
290 representation: XML escaping or Base64 encoding. Both ways are exclusive one of another.

291 [In both cases, any XML content sent in a JSON request MUST include all Namespace definitions needed](#)
292 [to parse that Content.](#)

293 4.2.3.1 XML Escaping

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

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

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

300 4.2.3.2 Base64 Encoding

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

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 <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 3.3.1 Supported Data Types. In the case of an array of values, the <code>DataType</code> cannot be inferred from the values inference works as described in 3.3.2. Arrays of values the array and the <code>DataType</code> JSON property must be specified.
IncludeInResult	Boolean	Optional	False.

341 4.2.4.1 Example

342 {Non-normative}

```
343     {"Attribute": [{
344         "AttributeId": "urn:oasis:names:tc:xacml:2.0:subject:role",
345         "Value": ["manager", "administrator"]
346     }]}
```

347 4.2.5 The MultiRequests object representation

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

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

352 4.2.6 The RequestReference object representation

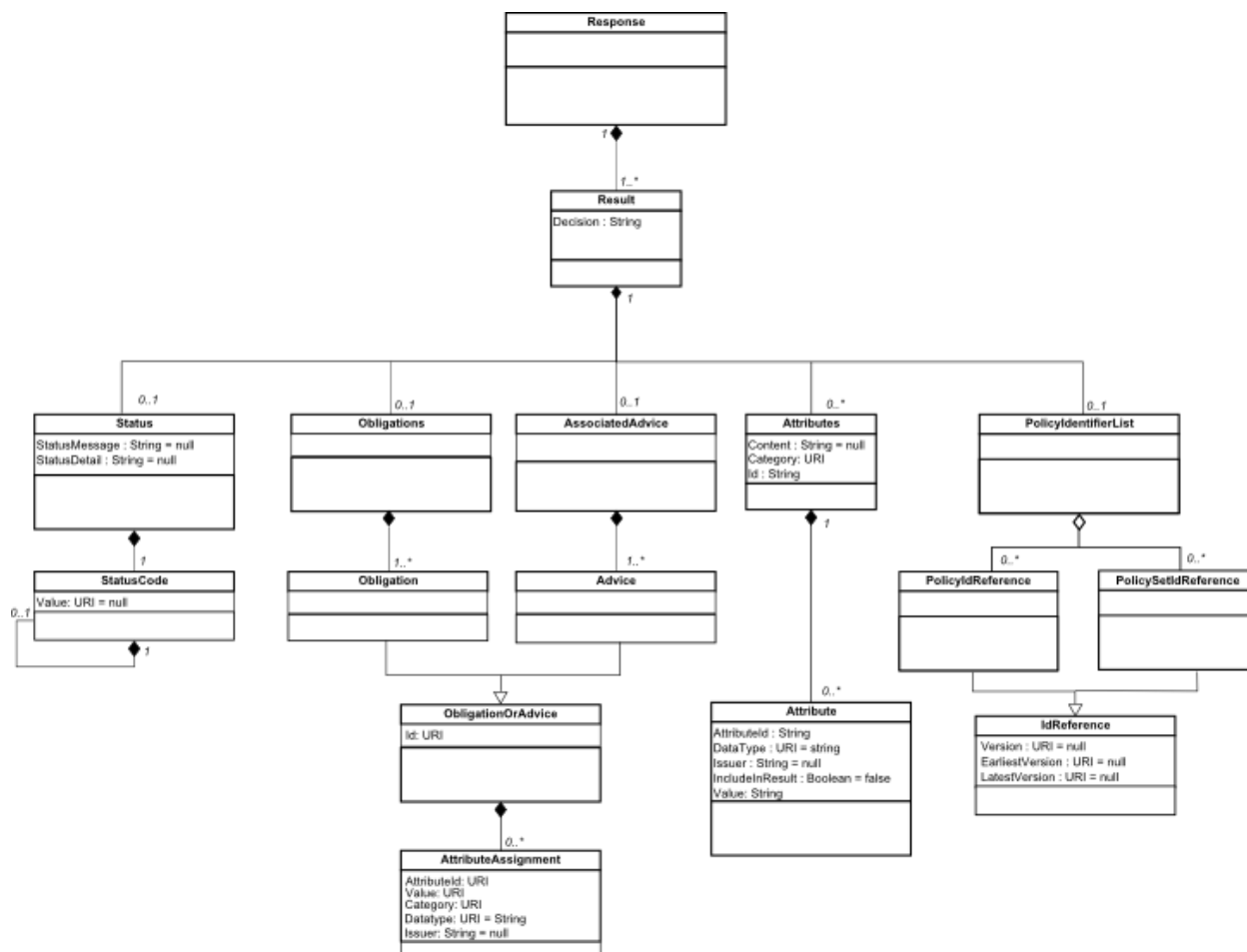
353 The RequestReference object contains a single property called ReferenceId which is an array of
354 string. Each ReferenceId value must be the value of a Category object Id property.

355 4.2.6.1 Non-normative example

```
356 {  
357 | "MultiRequests": {  
358 |     "RequestReference": [{  
359 |         "ReferenceId": ["foo1", "bar1"]  
360 |     },  
361 |     {  
362 |         "ReferenceId": ["foo2", "bar1"]  
363 |     },  
364 |     {  
365 |         "ReferenceId": ["foo3", "bar1"]  
366 |     }  
367 |     ]  
368 | }
```

369 5 The XACML response

370 5.1 Class Diagram



371

372 5.2 Representation of the XACML response in JSON

373 5.2.1 The Response object representation

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

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

379 5.2.2 The Result object representation

380 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 <code>"Permit"</code> , <code>"Deny"</code> , <code>"NotApplicable"</code> , <code>"</code>

			and <code>"Indeterminate"</code> . The values are case-sensitive.
--	--	--	---

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

- 382 • `Status`: this object is optional.
- 383 • `Obligations`: this object is optional.
- 384 • `AssociatedAdvice`: this object is optional.
- 385 • `AttributesCategory`: this object is optional. It can be single-valued or an array of `Category`
- 386 objects.
- 387 • `PolicyIdentifierList`: this object is optional.

388 **5.2.3 The Status object representation**

389 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.

390 In addition to the above properties, the `Status` object in JSON also contains a `StatusCode` object
 391 detailed hereafter. [The `StatusCode` object is optional.](#)

392 `StatusDetail` **can**[MAY](#) contain arbitrary XML as well. In the case that `StatusDetail` does contain
 393 XML, the XML content must be escaped using the same technique as specified in 4.2.3 The Content
 394 Object representation.

395 [StatusDetail MAY contain an array of `MissingAttributeDetail` object.](#)

396 **5.2.4 The MissingAttributeDetail object**

397 [The `MissingAttributeDetail` object in JSON contains the following properties:](#)

Property name	Type	Mandatory / Optional	Default value
Attributeld	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 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 4.2.2.1 Shorthand notation for standard XACML categories.

398

399 **5.2.45.2.5 The StatusCode object representation**

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

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

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

403 **5.2.4.15.2.5.1 Example**

404 {Non-normative}

```

405 {
406   "Response": [{
407     "Decision": "Permit"
408     "Status":{
409       "StatusCode":{
410         "Value": "http://foo.bar"
411       }
412     }
413   }]
414 }

```

415 **5.2.55.2.6 The Obligations object representation**

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

418 **5.2.65.2.7 The AssociatedAdvice object representation**

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

421 **5.2.75.2.8 The ObligationOrAdvice object representation**

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

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

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

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

428 **5.2.85.2.9 The AttributeAssignment object representation**

429 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. The shorthand notation defined in Shorthand notation for standard XACML categoriesNone may be used.
DataType	URI	Optional	The default value depends on the inference rules defined in Supported Data TypesString.
Issuer	String	Optional	None

430

431 **5.2.95.2.10 The Attributes object representation**

432 The JSON representation of the `Attributes` object in a XACML response [respects is identical to](#) the
433 representation defined in 4.2.2 The Category object representation.

434 **5.2.105.2.11 The PolicyIdentifier object representation**

435 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

436

437 **5.2.115.2.12 The IdReference object representation**

438 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 <code>PolicyIdReference</code> or <code>PolicySetIdReference</code> .
Version	String	Optional	None.

439

440 6 Transport

441 The XACML request represented in its JSON format MAY be carried from a PEP to a PDP via an HTTP
442 ~~[HTTP] POST request.~~ [request as defined in the REST profile of XACML \[XACMLREST\].](#)

443 HTTP Headers which may be used are:

- 444 • Content-Type: application/json
- 445 • Accept: application/json

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

448 6.1 Transport Security

449 **{Non-normative}**

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

452 7 IANA Registration

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

454 7.1 Media Type Name

455 application

456 7.2 Subtype Name

457 xacml+json

458 7.3 Required Parameters

459 None.

460 7.4 Optional Parameters

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

465 7.5 Encoding Considerations

466 Same as for application/xml [RFC4627].

467 7.6 Security Considerations

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

473 7.7 Interoperability Considerations

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

476 7.8 Applications which use this media type

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

480 7.9 Magic number(s)

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

482 7.10 File extension(s)

483 Per [RFC4627], .json.

484 **7.11 Macintosh File Type Code(s)**

485 Text

486 **7.12 Intended Usage**

487 Common

488 8 Examples

489 {Non-normative}

490 8.1 Request Example

491 {Non-normative}

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

```
493 {
494   "Request": {
495     "AccessSubject": {
496       "Attribute": [
497         {
498           "AttributeId": "subject-id",
499           "Value": "Andreas"
500         },
501         {
502           "AttributeId": "location",
503           "Value": "Gamla Stan"
504         }
505       ]
506     },
507     "Action": {
508       "Attribute": {
509         "AttributeId": "action-id",
510         "Value":
511           "http://www.xacml.euexample.com/buy",
512         "DataType": "anyURI"
513       }
514     },
515     "Resource": {
516       "Attribute": [
517         {
518           "AttributeId": "book-title",
519           "Value": "Learn German in 90 days"
520         },
521         {
522           "AttributeId": "currency",
523           "Value": "SEK"
524         },
525         {
526           "AttributeId": "price",
527           "Value": 123.34
528         }
529       ]
530     }
531   }
532 }
```

```
531         }
532     }
533 }
```

534 8.2 Response Example

535 **{Non-normative}**

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

```
537 {
538     "Response": [{
539         "Decision": "Permit"
540     }
541 ]
542 }
```

543 **9 Conformance**

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

547 **Appendix A. Acknowledgements**

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

550 **Participants:**

551 Steven Legg, ViewDS
552 Rich Levinson, Oracle
553 Hal Lockhart, Oracle
554 Bill Parducci,
555 Erik Rissanen, Axiomatics
556 Anil Saldhana, Red Hat
557 Remon Sinnema, EMC
558 Danny Thorpe, Dell
559 Paul Tyson, Bell Helicopters

560

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.
WD16	17 March 2014	David Brossard	<ul style="list-style-type: none"> • 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

			<p>XPathVersion to be in accordance with [XACML30].</p> <ul style="list-style-type: none"> • Added the MissingAttributeValue object definition.
WD17	14 April 2014	David Brossard	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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.