



XACML v3.0 Hierarchical Resource Profile Version 1.0

Committee Specification 01

10 August 2010

Specification URIs:

This Version:

<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cs-01-en.pdf>
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cs-01-en.doc> (Authoritative)
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cs-01-en.html>

Previous Version:

<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cd-03-en.pdf>
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cd-03-en.doc> (Authoritative)
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cd-03-en.html>

Latest Version:

<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-en.pdf>
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-en.doc> (Authoritative)
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-en.html>

Technical Committee:

OASIS eXtensible Access Control Markup Language (XACML) TC

Chair(s):

Bill Parducci, <bill@parducci.net>
Hal Lockhart, Oracle <hal.lockhart@oracle.com>

Editor(s):

Erik Rissanen, Axiomatics AB <erik@axiomatics.com>
Rich Levinson, Oracle <rich.levinson@oracle.com>
Hal Lockhart, Oracle <hal.lockhart@oracle.com>

Related work:

This specification replaces or supercedes:

- [Hierarchical resource profile of XACML v2.0](#)

This specification is related to:

- [eXtensible Access Control Markup Language \(XACML\) Version 3.0, CD 03](#)

Declared XML Namespace(s):

None

Abstract:

This document provides a profile for the use XACML with resources that are structured as hierarchies. The profile addresses resources represented as nodes in XML documents or represented in some non-XML way. The profile covers identifying nodes in a hierarchy, requesting access to nodes in a hierarchy, and specifying policies that apply to nodes in a hierarchy.

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” or “Latest Approved 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>.

The non-normative errata page for this specification is located at <http://www.oasis-open.org/committees/xacml/>.

Notices

Copyright © OASIS® 2010. 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 names "OASIS" and "XACML" are trademarks 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/who/trademark.php> for above guidance.

Table of Contents

1	Introduction.....	5
1.1	Glossary	6
1.1.1	Comparison of hierarchical structures	8
1.2	Terminology.....	8
1.3	Normative References.....	9
1.4	Non-Normative References	9
2	Representing the identity of a node.....	10
2.1	Nodes in XML documents.....	10
2.2	Nodes in hierarchical resources identified by URIs.....	10
2.2.1	Alternative URI-reference representation for XML documents	11
2.3	Nodes in hierarchical resources identified by ancestor attributes	12
3	Requesting access to a node	13
3.1	Nodes in an XML document.....	13
3.2	Nodes in hierarchical resources identified by URIs.....	13
3.3	Nodes in hierarchical resources identified by ancestor attributes	14
3.3.1	Pseudo-code for Nodes in hierarchical resources identified by ancestor attributes (non-normative)	15
4	Stating policies that apply to nodes.....	17
4.1	Policies applying to nodes with ancestor attributes.....	17
4.2	Policies applying only to nodes in XML documents	17
4.3	Policies applying only to nodes identified with URIs	17
5	New attribute identifiers.....	19
5.1	content-selector	19
5.2	document-id.....	19
5.3	resource-parent	19
5.4	resource-ancestor.....	19
5.5	resource-ancestor-or-self.....	19
6	New profile identifiers	20
7	Conformance.....	21
7.1	Nodes in XML documents.....	21
7.2	Nodes in hierarchical resources identified by URIs.....	21
7.3	Nodes in hierarchical resources identified by ancestor attributes	21
A.	Acknowledgements	22
B.	Revision History	23

1 Introduction

{Non-normative}

It is often the case that a resource is organized as a hierarchy. Examples include file systems, XML documents, and organizations. This Profile specifies how XACML can provide access control for a resource that is organized as a hierarchy.

Why are resources organized as hierarchies special? First of all, policies over hierarchies frequently apply the same access controls to entire sub-trees of the hierarchy. Being able to express a single policy constraint that will apply to an entire sub-tree of **nodes** in the hierarchy, rather than having to specify a separate constraint for each **node**, increases both ease of use and the likelihood that the policy will correctly reflect the desired access controls. Another special characteristic of **hierarchical resources** is that access to one **node** may depend on the value of another **node**. For example, a medical patient might be granted access to the “diagnosis” **node** in a XML document medical record only if the patient's name matches the value in the “patient name” **node**. Where this is the case, the requested **node** can not be processed in isolation from the rest of the **nodes** in the hierarchy, and the PDP must have access to the values of other **nodes**. Finally, the identity of **nodes** in a hierarchy often depends on the position of the **node** in the hierarchy; there also may be multiple ways to describe the identity of a single **node**. In this Profile, a resource organized as a hierarchy may be

- a “(rooted) tree” (a hierarchy with a single root),
- a “Directed Acyclic Graph” or “DAG” (a hierarchy with multiple roots, but a DAG may not have cycles; (also, a DAG may be expanded to an equivalent set of disjoint hierarchies, a fact, which is useful to know when conceptualizing the hierarchical properties of the DAG)),
- or a “polyarchy” (a “forest”, which is a disjoint set of trees, which when applied to a collection of resources may be designed to become a polyarchy, because each disjoint tree is layed on the same collection of resources, and nodes from disjoint trees, in general, may refer to the same resource, and as a result, with respect to the resource, merge to become a single node, which organizes the resources as a polyarchy; note also, that by jumping from one disjoint tree to another while on an intersecting node, that the polyarchy may contain cycles, which are not possible with the DAG).

All such resources are called **hierarchical resources** in this Profile. An XML document is always structured as a “tree”. Other types of **hierarchical resources**, such as files in a file system that supports links, may be structured as a “forest”.

In this Profile, the **nodes** in a **hierarchical resource** are treated as individual resources. An authorization decision that permits access to an interior **node** does not imply that access to its descendant **nodes** is permitted. An authorization decision that denies access to an interior **node** does not imply that access to its descendant **nodes** is denied.

There are three types of facilities specified in this Profile for dealing with **hierarchical resources**:

- Representing the identity of a **node**.
- Requesting access to a **node**.
- Stating policies that apply to one or more **nodes**.

Support for each of these facilities is optional.

This Profile addresses three ways of representing a **hierarchical resource**.

- In the first way, the hierarchy of which the **node** is a part is represented as an XML document that is included in the Request, and the requested resource is represented as a **node** in that document.
- In the second way, the resource must be a part of one or more singly rooted hierarchies. The resource is identified using a hierarchical URI which reflects the resource’s place in these hierarchies.

- 48 • In the third way, the resource may be a part of one or more singly or multiply rooted hierarchies.
49 The parent and other ancestor nodes of the resource are identified as attributes in the request.
50 The naming of the resource (or its ancestors) has no significance in terms of describing the
51 structure of the hierarchy.

52 Note that the actual target resource in the first case need not be part of an XML document - it is merely
53 represented that way in the Request. Likewise, the target resource in the second case might actually be
54 part of an XML document, but is being represented in some other way in the Request.

55 Facilities for dealing with resources represented as **nodes** in XML documents can make use of the fact
56 that the XML document itself is included in the decision request. **[XPath]** expressions can be used to
57 reference **nodes** in this document in a standard way, and can provide unique representations for a given
58 **node** in the document. These facilities are not available for **hierarchical resources** that are not
59 represented as XML documents. Other means must be provided in the case of such non-XML resources
60 for determining the location of the requested **node** in the hierarchy. In some cases this can be done by
61 including the **node**'s position in the hierarchy as part of the **node**'s identifier. In other cases, a **node** may
62 have more than one normative identity, such as when the pathname of a file in a file system can include
63 hard links. In such cases, the XACML PDP's Context Handler may need to supply the identities of all the
64 **node**'s ancestors. For all these reasons, the facilities for dealing with **nodes** in XML documents differ
65 from the facilities for dealing with **nodes** in other **hierarchical resources**.

66 In dealing with a **hierarchical resource**, it may be useful to request authorization decisions for multiple
67 **nodes** in the resource in a single decision request. Ways to make such requests are specified in another
68 Profile – the Multiple resource profile of XACML v3.0 **[MULTIPLE]**. That Profile also provides a way to
69 return a single authorization decision when access to multiple **nodes** in a hierarchy is requested.
70 Readers of this Profile are encouraged to become familiar with the Multiple resource profile of XACML.
71 This Profile may be considered to be layered on top of the multiple resource profile, which in turn is
72 layered on top of the behavior specified in the core XACML specification **[XACML]**. The functionality in
73 this Profile MAY, however, be layered directly on the functionality in the core XACML specification.

74 This Profile for **hierarchical resources** assumes that all requests for access to multiple **nodes** in a
75 **hierarchical resource** **[MULTIPLE]** have been resolved to individual requests for access to a single
76 **node**.

77 1.1 Glossary

78 DAG

79 A Directed Acyclic Graph (**DAG**), which may also be characterized as a **multi-rooted hierarchy**.

80 Hierarchical resource

81 A resource that is organized as a tree or (Directed Acyclic Graph (DAG) of individual resources
82 called **nodes**.

83 Hierarchy

84 A general term that applies to all the types of hierarchical representations that are used in this
85 specification to represent the organization of a collection of resource. This includes a **single-**
86 **rooted hierarchy**, a **multi-rooted hierarchy**, and a **multi-rooted disjoint hierarchy**.

87 Multi-rooted disjoint hierarchy

88 A “hierarchy” that has multiple top level “root” **nodes**, each of which is top **node** of a **single-**
89 **rooted hierarchy**, which in general, contains subtrees that overlap with subtrees of the other
90 **single-rooted hierarchies**, that are topped by the other top level root **nodes**, where all the
91 **nodes** that were in each original **single-rooted hierarchy** retain their identity as having been and
92 remaining as a member of that original hierarchy. Because of this retention of identity within
93 original **single-rooted hierarchy**, there are no restrictions with respect to cycles or otherwise as
94 to the layout of the **single-rooted hierarchies** with respect to each other. This structure is also
95 know as a “polyarchy”. It is also known as a “forest”, or “disjoint set of trees”, with the logical
96 physical characteristic that each “set of overlapping **nodes**” from multiple hierarchies that
97 identifies a specific single resource, actually contains a “set of individual distinct identifiers” any of
98 which can be used to identify that single resource within the **multi-rooted disjoint hierarchy**.

99
100 A specific example of this type of structure may begin with a set of resources that have been
101 identified and organized within a **single-rooted hierarchy** by having one of a set of hierarchical
102 URIs (considered to be a distinct hierarchical namespace) assigned to each resource as
103 described in section 2.2. One may then for a totally independent purpose apply another set of
104 hierarchical URIs (section 2.2) to a set of resources that may include part or all of the first set,
105 and may include new members that were not included in the first set. Note that any **multi-rooted**
106 **hierarchy (DAG)** may be represented in this manner.
107 However, the **multi-rooted disjoint hierarchy** (polyarchy) has no constraints on the additional
108 **single-rooted hierarchies** that are laid down, and therefore, can be used to create more
109 complex structures that may include cycles that cannot be represented by a **DAG**. Note also, that
110 the use of URIs is a convenience and not a necessity for implementation of this structure.

111 **Multi-rooted hierarchy**

112 A “hierarchy” that has multiple top level “root” **nodes**, each of which is top **node** of a **single-**
113 **rooted hierarchy**, which in general, contains subtrees that overlap with subtrees of the other
114 **single-rooted hierarchies**, that are topped by the other top level root **nodes**. This type of
115 “hierarchy” is also know as a Directed Acyclic Graph (**DAG**). In general, multiple **single-rooted**
116 **hierarchies** may be laid across a set of resources for organization purposes. The **DAG**
117 properties constrain the layout options somewhat, in that within the layout of the multiple
118 overlapping hierarchies, there may not be contained any cycles, i.e. where one could follow a
119 path from any particular **node** that eventually returns to that same particular **node**.

120
121 A specific example of this type of structure may begin with a set of resources that have been
122 identified and organized within a **single-rooted hierarchy** by having one of a set of hierarchical
123 URIs (considered to be a distinct hierarchical namespace) assigned to each resource as
124 described in section 2.2. One may then for a totally independent purpose apply another set of
125 hierarchical URIs (section 2.2) to a set of resources that may include part or all of the first set,
126 and may include new members that were not included in the first set. Note that any **multi-rooted**
127 **hierarchy (DAG)** may be represented in this manner.
128 However, there are constraints on the 2nd and additional **single-rooted hierarchies** that are laid
129 down, specifically, that no cycles are allowed to be produced when the new edges are added to
130 the DAG for the additional hierarchies.

131 **Node**

132 An individual resource that is part of a **hierarchical resource**.

133 **Single-rooted hierarchy**

134 A “hierarchy” that has one top level “root” **node** and each member of the hierarchy can have only
135 one parent **node**. Examples of resources that fit this model include a single XML document, and
136 any **hierarchical resource** that is organized as a single hierarchy, such as typical organization
137 charts, or the individual components within an overall assembly, where the finished assembled
138 entity represents the top root node.

139 **1.1.1 Comparison of hierarchical structures**

140 The following table is intended to capture the salient features of the hierarchical structures used in this
 141 document:

	Single-Rooted Hierarchy (XML document)	Multi-Rooted Hierarchy (DAG)	Multi-Rooted Disjoint Hierarchy (polyarchy)
Number of root nodes	1	n>=1	n>=1
Maximum number of parent nodes	1	m>=1	m>=1
Is original hierarchical membership retained	Yes	No	Yes
Are navigation cycles allowed	No	No	Yes, by shifting to at least one different original hierarchy along cyclic path, if such paths exist.
Are there restrictions whether a specific existing node is allowed to be made a child of current node	Yes	Yes, if adding the new node will create a cycle.	No, however, each new connection made must identify a specific hierarchy included in current node, or begin a new hierarchy.

142 The situation with “cycles” is that there seems, in general, little point to purposely trying to create such a
 143 cycle, however, if such a cycle should happen to occur as a result of the difference in semantics of **two**
 144 **single-rooted hierarchies** that are being applied to the set of resources, whereby, for example, if in one
 145 hierarchy node “a” is the parent of node “b”, while in a 2nd hierarchy node “b” was the parent of node “a”
 146 then such a construct would not be allowed by the DAG, but would be allowed by the polyarchy. As a
 147 result, the polyarchy may be regarded as more general than the DAG, because the layouts possible with
 148 a polyarchy are a superset of those possible with a DAG on the same set of resources.

149

150 **1.2 Terminology**

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

154 The phrase **{Optional}** means that the described functionality is optional for compliant XACML
 155 implementations, but, if the functionality is claimed as being supported according to this Profile, then it
 156 SHALL be supported in the way described.

157 `Example code listings appear like this.`

158 In descriptions of syntax, elements in angle brackets (“<”, “>”) are to be replaced by appropriate values,
 159 square brackets (“[”, “]”) enclose optional elements (but are taken as literal when within quotes), elements
 160 in quotes are literal components, backslash-quote (“\”) is a literal quote character within a literal
 161 component, and an unquoted asterisk, (*), indicates that the preceding element may occur zero or more
 162 times, whereas an asterisk in quotes, (“*”), is a literal asterisk.

163 1.3 Normative References

- 164 [ISO10181-3] ISO/IEC JTC 1, *Information technology -- Open Systems Interconnection --*
165 *Security frameworks for open systems: Access control framework*, ISO/IEC
166 10181-3:1996, 1996.
- 167 [RFC1034] P. Mockapetris, *DOMAIN NAMES – CONCEPTS AND FACILITIES*, IETF RFC
168 1034, November 1987, <http://www.ietf.org/rfc/rfc1034.txt>
- 169 [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
170 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- 171 [RFC3986] T. Berners-Lee, et al., *Uniform Resource Identifiers (URI): Generic Syntax*,
172 <http://www.ietf.org/rfc/rfc3986.txt>, IETF RFC 3986, January 2005.
- 173 [RFC3198] A. Westerinen, et al., *Terminology for Policy-Based Management*,
174 <http://www.ietf.org/rfc/rfc3198.txt>, IETF RFC 3198, November 2001.
- 175 [MULTIPLE] OASIS Committee Specification 01, XACML v3.0 Multiple Decision Profile
176 Version 1.0, August 2010, [http://docs.oasis-open.org/xacml/3.0/xacml-3.0-](http://docs.oasis-open.org/xacml/3.0/xacml-3.0-multiple-v1-spec-cs-01-en.doc)
177 [multiple-v1-spec-cs-01-en.doc](http://docs.oasis-open.org/xacml/3.0/xacml-3.0-multiple-v1-spec-cs-01-en.doc)
- 178 [XACML] OASIS Committee Specification 01, eXtensible access control markup language
179 (XACML) Version 3.0. August 2010. [http://docs.oasis-open.org/xacml/3.0/xacml-](http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.doc)
180 [3.0-core-spec-cs-01-en.doc](http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.doc)
- 181 [XPath] *XML Path Language (XPath)*, Version 1.0, W3C Recommendation 16, November
182 1999. Available at <http://www.w3.org/TR/xpath>

183 1.4 Non-Normative References

- 184 [URIOpacity] Ian Jacobs, et al., *Architecture of the World Wide Web, Volume One*, section
185 2.5, W3C Recommendation 15 December 2004,
186 <http://www.w3.org/TR/webarch/#uri-opacity>

187

2 Representing the identity of a node

188 In order for XACML policies to apply consistently to **nodes** in a **hierarchical resource**, it is necessary for
189 the **nodes** in that resource to be represented in a consistent way. If a policy refers to a **node** using one
190 representation, but a request refers to the **node** using a different representation, then the policy will not
191 apply, and security may be compromised.

192 The following sections describe RECOMMENDED representations for **nodes** in **hierarchical resources**.
193 Alternative representations of **nodes** in a given resource are permitted so long as all Policy
194 Administration Points and all Policy Enforcement Points that deal with that resource have contracted to
195 use the alternative representation.

2.1 Nodes in XML documents

197 {Optional}

198 The following URI SHALL be used as the identifier for the functionality specified in this Section of this
199 Profile. This identifier represents metadata about this specification and implementations implementing this
200 specification. This identifier MAY be used to describe capabilities of an implementation or to make other
201 references to this specification

- 202 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:xml-node-id

203 The identity of a **node** in a resource that is represented as an XML document instance SHALL be an
204 XPath expression that evaluates to exactly that one **node** in the copy of the resource that is contained in
205 the <Content> element of the <Attributes> element with the resource category of the <Request>.

206 Note: one possible XPath expression template for representation of node identifiers in an XML document
207 or as part of a URI-reference is described in section 2.2.1.

2.2 Nodes in hierarchical resources identified by URIs

209 {Optional}

210 The following URI SHALL be used as the identifier for the functionality specified in this Section of this
211 Profile. This identifier represents metadata about this specification and implementations implementing this
212 specification. This identifier MAY be used to describe capabilities of an implementation or to make other
213 references to this specification

- 214 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:URI-node-id.

215 The identity of a **node** in a **hierarchical resource** that is not represented as an XML document instance
216 MAY be represented as a URI that conforms to [RFC3986] and which has a hierarchical structure where
217 the ancestors are delimited by slashes. (According to [RFC2396] URI schemes may be non-hierarchical,
218 e.g. mailto:, hierarchical without slashes, e.g. urn: or hierarchical using slashes, e.g. http:). Hierarchical
219 URIs with slashes are of the following generic form.

220 <scheme> “.” [“/” <authority>] [“/” <pathname>]

221 File system resources SHALL use the “file:” scheme. If the resource is identified with a standard
222 <scheme> specified in [RFC3986] or in a related standard for a registered URI scheme which is
223 hierarchical with slashes, then that scheme SHALL be used. Otherwise the URI SHALL use the “file:”
224 scheme.

225 The <pathname> portion of the URI SHALL be of the form

226 <root name> [“/” <node name>] *

- 227 • The sequence of <root name> and <node name> values SHALL correspond to the individual
228 hierarchical component names of ancestors of the represented **node** along the path from a <root>
229 **node** to the represented **node**.

- 230 • The components of the <pathname> portion of the URI SHALL be specified using the canonical form
231 for such path components at the <authority>.
 - 232 • In accordance with [RFC3986], the separator character between hierarchical components of the
233 <pathname> portion of the URI SHALL be the character “/”. Sequences of the “/” character SHALL
234 be resolved to a single “/”. **Note** identities SHALL NOT terminate with the “/” character.
 - 235 • All <pathname> values SHALL be absolute.
 - 236 • If there is more than one fully resolved, absolute path from a <root> at the <authority> to the
237 represented **node**, then a separate resource attribute with AttributeId
238 “urn:oasis:names:tc:xacml:1.0:resource:resource-id” and DataType
239 http://urn:oasis:names:tc:xacml:1.0:data-type:anyURI SHALL be present in the Request Context for
240 each such path.
- 241 Implementation note: the scheme name of the URI should be checked to determine it is an expected
242 scheme before parsing the URI into its hierarchical components.
- 243 Also note that the notion of parsing the syntax of a URI is controversial, see for example [URIOpacity].

244 2.2.1 Alternative URI-reference representation for XML documents

245 {Optional}

246 The following URI SHALL be used as the identifier for the functionality specified in this Section of this
247 Profile:

- 248 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:URI-reference-node-id.

249 The identity of a **node** in a **hierarchical resource** that is represented as an XML document instance
250 MAY be represented as a URI-reference that conforms to [RFC3986] and which has a hierarchical
251 structure where the ancestors are delimited by slashes. Hierarchical “URI-references” with slashes
252 conform to the following five component generic form (where the “URI portion” is the first four
253 components):

254 <scheme> “.” [“/” <authority>] [“/” <pathname>] [“?” <query>] [“#” <fragment>]

255 The query portion of the URI is not used in this profile.

256 The <fragment> portion of the URI-reference MAY be used to identify explicit element, attribute, text, and
257 other nodes in an XML document when constructed as an XPath path expression using the following
258 form:

259 <fragment> = “xpointer(/” <fragment-id> “)”

260 <fragment-id> = [<doc-node-xsegment> [“/” <elem-node-xsegment>] * [“/” <end-node-xsegment>]]

261 where

262 <end-node-xsegment> = <attr-node-xsegment> |

263 <text-node-xsegment> |

264 <other-node-xsegment>

265 and

266 <doc-node-xsegment> = “*.” <doc-node-local-name> [<namespace-uri>]

267 <elem-node-xsegment> = “*.” <elem-node-local-name> [<namespace-uri>] [<position>]

268 <attr-node-xsegment> = “@” “*.” <attr-node-local-name> [<namespace-uri>]

269 <text-node-xsegment> = “text()”

270 <other-node-xsegment> = (literal xpath syntax for other node types)

271

272 <position> = “[<integer> ”]

273 <integer> = (same as result of xpath fcn: position(), i.e. integer >= 1)

274

275 <namespace-uri> = “[namespace-uri()=\\” <literal-namespace> “\”]
276 <literal-namespace> = (same as result of xpath fcn: namespace-uri())

277 Notes:

- 278 • When expressions using the above syntax are used within an actual URI-Reference, the literal forms
279 of <*-node-xsegment> items MUST be percent-encoded as described in [RFC2396]. However, the
280 decoded form is an executable XPath path expression.
- 281 • The <*-node-xsegment> items are all have a leading “*.” which selects all nodes in any namespace
282 with the <*-node-local -name> that follows. The following <namespace-uri> item is then used to
283 specify the namespace.
- 284 • When the following literal <namespace-uri> predicate appears in an expression it may be ignored or
285 removed by the policies without changing the meaning of the expression:
286 [namespace-uri()=”]
287 because when the namespace-uri() XPath function evaluates to the empty string as shown, this
288 means there is “no namespace” defined for the element, which is equivalent to an unprefix local-
289 name QName. Also the “*.” may also be ignored or removed. (Ignoring or removing is meant within
290 the context of regular expression (regexp) processing.)
291 For example the XPath segment “*.abc[namespace-uri()=”]” may be regarded as equal to “abc”.
- 292 • Policies may, in general, ignore <position> predicates for matching purposes (i.e. allow “any” position
293 value), because they usually do not represent a specific property of the node, but only provide a
294 discriminator for otherwise equal node locations within the hierarchy. For example, a list of line items
295 in a purchase order, usually does not attach any specific significance to the order in which the line
296 items appear.

297 2.3 Nodes in hierarchical resources identified by ancestor attributes

298 {Optional}

299 The following URI SHALL be used as the identifier for the functionality specified in this Section of this
300 Profile. This identifier represents metadata about this specification and implementations implementing this
301 specification. This identifier MAY be used to describe capabilities of an implementation or to make other
302 references to this specification

- 303 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:attribute-node-id.

304 The identity of a **node** in a **hierarchical resource** that is not represented as an XML document instance
305 MAY be represented by specifying its ancestors as XACML attributes in the request. In this case the node
306 and its ancestors may be identified using identifiers of any XACML datatype. There is no requirement that
307 different nodes use the same XACML datatype or that nodes in the same hierarchy use the same
308 datatype.

309 In this mode of operation, any number of hierarchies with any number of roots may be represented,
310 however, only hierarchies of which the resource is a member will be included. Hierarchies which include
311 the ancestors or descendants of the resource, but do not contain the resource are not included.

312 In this approach, considerable information is discarded. It is not possible to determine how many
313 hierarchies there are or which ancestors are in which hierarchies or the relative position of ancestors
314 other than immediate parents.

315 3 Requesting access to a node

316 In order for XACML policies to apply consistently to **nodes** in a **hierarchical resource**, it is necessary for
317 each request context that represents a request for access to a **node** in that resource to use a consistent
318 description of that **node** access. If a policy refers to certain expected attributes of a **node**, but the request
319 context does not contain those attributes, or if the attributes are not expressed in the expected way, then
320 the policy may not apply, and security may be compromised.

321 The following sections describe RECOMMENDED request context descriptions of access to **nodes** in
322 **hierarchical resources**. Alternative representations of such requests are permitted so long as all Policy
323 Administration Points and all Policy Enforcement Points that deal with that resource have contracted to
324 use the alternative representation.

325 3.1 Nodes in an XML document

326 {Optional}

327 The following URI SHALL be used as the identifier for the functionality specified in this Section of this
328 Profile. This identifier represents metadata about this specification and implementations implementing this
329 specification. This identifier MAY be used to describe capabilities of an implementation or to make other
330 references to this specification

- 331 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:xml-node-req

332 In order to request access to a resource represented as a **node** in an XML document, the request context
333 <Attributes> element in the resource category SHALL contain the following elements and XML
334 attributes:

- 335 • A <Content> element that contains the entire XML document instance of which the requested **node**
336 is a part.
- 337 • An <Attribute> element with an AttributeId of "urn:oasis:names:tc:xacml:3.0:content-selector"
338 and a DataType of "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression". The
339 <AttributeValue> of this <Attribute> SHALL be an XPath expression whose context node
340 SHALL be the <Content> element in the "urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
341 attribute category. This XPath expression SHALL evaluate to a nodeset containing the single node
342 in the <Content> element that is the **node** to which access is requested. This <Attribute> MAY
343 specify an Issuer.

344 Additional attributes MAY be included in the <Resource> element. In particular, the following attribute
345 MAY be included.

- 346 • An <Attribute> element with an AttributeId of
347 "urn:oasis:names:tc:xacml:2.0:resource:document-id" and a DataType of
348 "urn:oasis:names:tc:xacml:1.0:data-type:anyURI". The <AttributeValue> of this <Attribute>
349 SHALL be a URI that identifies the XML document of which the requested resource is a part, and of
350 which a copy is present in the <Content> element. This <Attribute> MAY specify an Issuer.

351 3.2 Nodes in hierarchical resources identified by URIs

352 {Optional}

353 The following URI SHALL be used as the identifier for the functionality specified in this Section of this
354 Profile:

- 355 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:URI-node-id.

356
357 The resource SHALL be identified by means of a hierarchical URI (or URIs) as described in section 2.2.
358 Parent and Ancestor attributes SHALL NOT be provided.

3.3 Nodes in hierarchical resources identified by ancestor attributes

{Optional}

The following URI SHALL be used as the identifier for the functionality specified in this Section of this Profile

- urn:oasis:names:tc:xacml:3.0:profile:hierarchical:attribute-node-id.

The attributes with AttributeIds of “urn:oasis::names:tc:xacml:2.0:resource:resource-parent”, “urn:oasis::names:tc:xacml:2.0:resource:resource-ancestor”, and “urn:oasis::names:tc:xacml:2.0:resource:resource-ancestor-or-self” are optional to implement. If this section of the specification is supported, the following URIs SHALL be used as identifiers for the functionality they represent:

- urn:oasis:names:tc:xacml:2.0:profile:hierarchical:non-xml-node-req:resource-parent
- urn:oasis:names:tc:xacml:2.0:profile:hierarchical:non-xml-node-req:resource-ancestor
- urn:oasis:names:tc:xacml:2.0:profile:hierarchical:non-xml-node-req:resource-ancestor-or-self

In order to request access to a **node** in a **hierarchical resource** in this mode of operation, the request context <Attributes> element SHALL NOT contain a <Content> element. The request context <Attributes> element in the resource category SHALL contain the following elements and XML attributes. Note that in this case, a node MAY have multiple parents. For example, in a file system that supports hard links, there may be multiple normative paths to a single file. Each such path MAY contain different sets of parents and ancestors.

The following discussion assumes that the Context Handler knows what hierarchies exist, how they are represented and how the nodes in them are named. There may be any number of distinct hierarchies which may be singly or multiply rooted. Individual nodes may belong to any number of hierarchies. Nodes in the hierarchies may be of a single type or multiple types. The resource-id of nodes may be of the same XACML datatype or different ones. Where they use the same datatype, say string, the naming scheme may be a single scheme or multiple schemes. A node may have a different name in every hierarchy it is in or one name in all hierarchies. A node may have multiple names in a single hierarchy of which it is a member. In general the naming scheme is not constrained to relate to the hierarchy in any way.

All that is required is that the Context Handler be able to determine what hierarchies exist, what are the resource-ids of the members and what are their relationships. Starting from this information the Context Handler SHALL perform the following steps or some process which gives equivalent results.

1. Identify all the hierarchies associated with the resources in question.
2. Drop from further consideration any hierarchies of which the node in question is not actually a member.
3. Drop from further consideration any descendants of the node.
4. In each hierarchy in turn, collect all of the identifiers for all of the nodes in each hierarchy for each of the node types described below.
5. Discard any duplicates.

- For each representation of the requested **node**, an <Attribute> element with AttributeId of “urn:oasis:names:tc:xacml:1.0:resource:resource-id”. The <AttributeValue> of this <Attribute> SHALL be an identifier of the **node** to which access is requested. The DataType of the <AttributeValue> of this <Attribute> MAY be of any XACML datatype. This <Attribute> MAY specify an Issuer.
- For each immediate parent of the **node** specified in the “resource-id” attribute or attributes, and for each representation of that parent **node**, an <Attribute> element with AttributeId “urn:oasis:names:tc:xacml:2.0:resource:resource-parent”. The <AttributeValue> of this <Attribute> SHALL be an identifier of the parent **node**. The DataType of the

407 <AttributeValue> of this <Attribute> MAY be of any XACML datatype This <Attribute>
 408 MAY specify an Issuer.

- 409 • For each ancestor of the **node** specified in the “resource-id” attribute or attributes, and for each
 410 representation of that ancestor **node**, an <Attribute> element with AttributeId
 411 “urn:oasis:names:tc:xacml:2.0:resource:resource-ancestor”. The <AttributeValue> of this
 412 <Attribute> SHALL be an identifier of the ancestor **node**. The DataType of the
 413 <AttributeValue> of this <Attribute> MAY be of any XACML datatype This <Attribute>
 414 MAY specify an Issuer.
- 415 • For each ancestor of the **node** specified in the “resource-id” attribute or attributes, and for each
 416 representation of that ancestor **node**, and for each representation of the “resource-id” **node** itself, an
 417 <Attribute> element with AttributeId “urn:oasis:names:tc:xacml:2.0:resource:resource-
 418 ancestor-or-self”. The <AttributeValue> of this <Attribute> SHALL be an identifier of the
 419 ancestor **node** or of the “resource-id” **node** itself. The DataType of the <AttributeValue> of this
 420 <Attribute> MAY be of any XACML datatype. This <Attribute> MAY specify an Issuer.
 421 Additional attributes MAY be included in the <Attributes> element.

422 3.3.1 Pseudo-code for Nodes in hierarchical resources identified by 423 ancestor attributes (non-normative)

424 This section contains pseudo-code which may be considered to represent a model by which one can
 425 represent any collection of resources that are each individually identified as belonging to one or more
 426 hierarchies and/or DAGs. An algorithm is then defined to process the collection according to the rules of
 427 section 3.3.

```

428 // Define a class for "Resource Hierarchy identifier" node
429 public class ResHierId(int res, int hier)
430
431 // Define Sets to collect nodes in:
432 selfNodes = new HashSet<ResHierId>();
433 parentNodes = new HashSet<ResHierId>();
434 ancestorNodes = new HashSet<ResHierId>();
435 ancestorOrSelfNodes = new HashSet<ResHierId>();
436
437 // Define number of resources, hierarchies and 1-based 2-d array
438 int nRes=4, mHier=5; // example hierarchy dims
439 int[][] ijResource = new int[nRes+1][mHier+1];
440
441 // Define method to collect nodes
442 collectAncestorNodes(int iRes) {
443     for (int j = 1; j<mHier+1; j++){
444         int mDag = 1; m=j; iDepth = 0;
445         if (ijResource[0][j] != 0){
446             while ((m<mHier) && (ijResource[0][m+1] == ijResource[0][j])){
447                 mDag++; m++;
448             } }
449         walkUpHierarchyDag(iRes, j, mDag, iDepth);
450         j=j+mDag-1; // skip columns handled by mDag
451     } }
452
453 walkUpHierarchyDag(int iRes, int j, int mDag, int iDepth){
454     // for each instance of self in Dag subrow
455     for (int k=1; k<mDag+1; k++){
456         int m = j+k-1; // m is column in big matrix
457         int iResCurrent = iRes; // iResCurrent is 1-based row-id
458         if (ijResource[iResCurrent][m] != 0){
459             ResHierId rhId = new ResHierId(iResCurrent,m);
460             if (iDepth == 0){
461                 selfNodes.add(rhId);
462                 ancestorOrSelfNodes.add(rhId);
463             }
464             else if (iDepth == 1){
465                 parentNodes.add(rhId);
466                 ancestorNodes.add(rhId);
467                 ancestorOrSelfNodes.add(rhId);
468             }

```



```

469     else {
470         ancestorNodes.add(rhId);
471         ancestorOrSelfNodes.add(rhId);
472     }
473     if (iResCurrent != ijResource[iResCurrent][m]) {
474         // Set the new current node as parent of current node
475         iResCurrent = ijResource[iResCurrent][m];
476         iDepth++;
477         walkUpHierarchyDag(iResCurrent, j, mDag, iDepth);
478     }
479     else { } // found root on this path - done
480 }
481 else { } // zero means node not used - done
482 } }

```

483 Note the following:

- 484 • The matrix, ijResource[nRes+1][mHier+1] represents a collection of nRes resources, each of
485 which may belong to any of mHier single-parent hierarchies, or a mix of hierarchies and DAGS.
 - 486 • DAGs are represented by multiple columns, where the width of the DAG is mDag, which is equal
487 to the maximum number of parents that a single node in the DAG currently has. It is assumed
488 that the matrix has been prepared such that all columns within a single DAG are adjacent. Each
489 DAG has a unique “DAG-id”, which is present in row 0 of each column of the DAG. By contrast,
490 single-parent hierarchies (single column) have a zero in row 0.
 - 491 • The matrix is generally sparse, is initialized to all zeroes, and single-parent hierarchies and
492 DAGs are built by assigning the row number (effectively resource-id) of the parent of the
493 resource to the cell in resource’s row, effectively making the row a collection of potential
494 hierarchies and DAGs that the resource can belong to. The root of a hierarchy is indicated by the
495 row element pointing to the current row, a self-reference.
 - 496 • The 2-d array is “one-based” in that column 0 and row 0 are not used so that resources and
497 hierarchies may be identified as running from 1->nRes and 1->mHier.
 - 498 • Once the matrix is built, the ancestors for a resource may be collected by passing the row
499 number of the resource to the collectAncestorNodes(iRes) method. For each hierarchy and DAG
500 in the matrix, the recursive walkUpHierarchyOrDag(res-id, hier-id, dag-width, depth) method is
501 called, which will collect all the ancestors of either a hierarchy or DAG.
 - 502 • The collected ancestors are stored in 4 sets: one each for self, parent, ancestor, and ancestor-
503 or-self.
 - 504 • This algorithm is intended to be a model only and does not represent any specific
505 implementation strategy, except to clearly identify a concrete framework for identifying all the
506 resources and hierarchies and DAGs that are potentially covered by this profile.
- 507

508 4 Stating policies that apply to nodes

509 {Non-normative}

510 This Section describes various ways to specify a policy predicate that can apply to multiple *nodes* in a
511 *hierarchical resource*. This is not intended to be an exhaustive list.

512 4.1 Policies applying to nodes with ancestor attributes

513 {Non-normative}

514 Resource attributes with the following `AttributeId` values, described in Section 5: New attribute
515 identifiers for *hierarchical resources* of this Profile, MAY be used to state policies that apply to one or
516 more *nodes* in any *hierarchical resource*.

517 urn:oasis:names:tc:xacml:2.0:resource:resource-parent

518 urn:oasis:names:tc:xacml:2.0:resource:resource-ancestor

519 urn:oasis:names:tc:xacml:2.0:resource:resource-ancestor-or-self

520 Note that a `<AttributeDesignator>` that refers to the “resource-parent”, “resource-ancestor”, or
521 “resource-ancestor-or-self” attribute will return a bag of values representing all normative identities of all
522 parents, ancestors, or ancestors plus the resource itself, respectively, of the resource to which access is
523 being requested. The representations of the identities of these parents, ancestors, or self will not
524 necessarily indicate the path from the root of the hierarchy to the respective parent, ancestor, or self
525 unless the representation recommended in Section 3.2: Nodes in a resource that is not an XML document
526 is used.

527 The standard XACML [XACML] bag and higher-order bag functions MAY be used to state policies that
528 apply to one or more *nodes* in any *hierarchical resource*. The *nodes* used as arguments to these
529 functions MAY be specified using a `<AttributeDesignator>` with the “resource-parent”, “resource-
530 ancestor”, or “resource-ancestor-or-self” `AttributeId` value.

531 4.2 Policies applying only to nodes in XML documents

532 {Non-normative}

533 For *hierarchical resources* that are represented as XML document instances, the following function,
534 described in the XACML 3.0 Specification [XACML] MAY be used to state policy predicates that apply to
535 one or more *nodes* in that resource.

536 urn:oasis:names:tc:xacml:3.0:function:xpath-node-match

537 The standard XACML `<AttributeSelector>` element MAY be used in policies to refer to all or
538 portions of a resource represented as an XML document and contained in the `<Content>` element of a
539 request context.

540 The standard XACML [XACML] bag and higher-order bag functions MAY be used to state policies that
541 apply to one or more *nodes* in a resource represented as an XML document. The *nodes* used as
542 arguments to these functions MAY be specified using an `<AttributeSelector>` that selects a portion
543 of the `<Content>` element of the `<Attributes>` element with the resource category.

544 4.3 Policies applying only to nodes identified with URIs

545 {Non-normative}

546 For *hierarchical resources* that are not represented as XML document instances, and where the URI
547 representation of *nodes* specified in Section 2.2 of this Profile is used, the following functions described
548 in the XACML 3.0 Specification [XACML] MAY be used to state policies that apply to one or more *nodes*
549 in that resource.

550 urn:oasis:names:tc:xacml:1.0:function:anyURI-equal
551 urn:oasis:names:tc:xacml:2.0:function:regexp-uri-match

552 5 New attribute identifiers

553 {Optional}

554 5.1 content-selector

555 The following identifier locates with an XPath expression the resource in the XML document that
556 represents the hierarchy in which the requested resource is a part. The `DataType` of this attribute **MUST**
557 be “urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression”.

558 urn:oasis:names:tc:xacml:3.0:content-selector

559 5.2 document-id

560 The following identifier indicates the identity of the XML document that represents the hierarchy of which
561 the requested resource is a part, and of which a copy is present in the `<Content>` element. Whenever
562 access to a **node** in a resource represented as an XML document is requested, one or more instances of
563 an attribute with this `AttributeId` **MAY** be provided in the `<Attributes>` element of the request
564 context. The `DataType` of these attributes **SHALL** be “urn:oasis:names:tc:xacml:1.0:data-type:anyURI”.

565 urn:oasis:names:tc:xacml:2.0:resource:document-id

566 5.3 resource-parent

567 The following identifier indicates one normative identity of one parent **node** in the tree or forest of which
568 the requested **node** is a part. Whenever access to a **node** in a **hierarchical resource** is requested, one
569 instance of an attribute with this `AttributeId` **SHALL** be provided in the `<Attributes>` element of the
570 request context for each normative representation of each **node** that is a parent of the requested **node**.

571 urn:oasis:names:tc:xacml:2.0:resource:resource-parent

572 5.4 resource-ancestor

573 The following identifier indicates one normative identity of one ancestor **node** in the tree or forest of which
574 the requested **node** is a part. Whenever access to a **node** in a **hierarchical resource** is requested, one
575 instance of an attribute with this `AttributeId` **SHALL** be provided in the `<Attributes>` element of the
576 request context for each normative representation of each **node** that is an ancestor of the requested
577 **node**.

578 urn:oasis:names:tc:xacml:2.0:resource:resource-ancestor

579 5.5 resource-ancestor-or-self

580 The following identifier indicates one normative identity of one ancestor **node** in the tree or forest of which
581 the requested **node** is a part, or one normative identity of the requested **node** itself. Whenever access to
582 a **node** in a **hierarchical resource** is requested, one instance of an attribute with this `AttributeId`
583 **SHALL** be provided in the `<Attributes>` element of the request context for each normative
584 representation of each **node** that is an ancestor of the requested **node**, and for each normative
585 representation of the requested **node** itself.

586 urn:oasis:names:tc:xacml:2.0:resource:resource-ancestor-or-self

587 **6 New profile identifiers**

588 The following URI values SHALL be used as identifiers for the functionality specified in various Sections
589 of this Profile:

590 Section 2.1: Nodes in XML documents

- 591 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:xml-node-id

592 Section 2.2: Nodes in resources that are not XML documents

- 593 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:non-xml-node-id

594 Section 3.1: Nodes in an XML document

- 595 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:xml-node-req

596 Section 3.2: Nodes in a resource that is not an XML document

- 597 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:non-xml-node-req

598 Support for the “resource-parent”, “resource-ancestor”, and “resource-ancestor-or-self” attributes is
599 optional within this Section, so these have separate identifiers:

- 600 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:non-xml-node-req:resource-parent

- 601 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:non-xml-node-req:resource-ancestor

- 602 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:non-xml-node-req:resource-ancestor-or-self

603 **7 Conformance**

604 Implementations of this profile MAY conform to any or all of the following conformance clauses.

605 **7.1 Nodes in XML documents**

606 Implementations supporting hierarchical resources as nodes in an xml document SHALL conform to
607 sections 2.1 and 3.1. The following URI identifies this functionality.

- 608 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:xml-node-id

609

610 **7.2 Nodes in hierarchical resources identified by URIs**

611 Implementations supporting hierarchical resources by means of URIs SHALL conform to sections 2.2 and
612 3.2. The following URI identifies this functionality.

- 613 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:URI-node-id

614

615 **7.3 Nodes in hierarchical resources identified by ancestor attributes**

616 Implementations supporting hierarchical resources by means of ancestor attributes SHALL conform to
617 sections 2.3 and 3.3. The following URI identifies this functionality.

- 618 • urn:oasis:names:tc:xacml:3.0:profile:hierarchical:attribute-node-id.

619

620 **A. Acknowledgements**

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

623
624 Anil Saldhana
625 Anil Tappetla
626 Anne Anderson
627 Anthony Nadalin
628 Bill Parducci
629 Craig Forster
630 David Chadwick
631 David Staggs
632 Dilli Arumugam
633 Duane DeCouteau
634 Erik Rissanen
635 Gareth Richards
636 Hal Lockhart
637 Jan Herrmann
638 John Tolbert
639 Ludwig Seitz
640 Michiharu Kudo
641 Naomaru Itoi
642 Paul Tyson
643 Prateek Mishra
644 Rich Levinson
645 Ronald Jacobson
646 Seth Proctor
647 Sridhar Muppidi
648 Tim Moses
649 Vernon Murdoch
650
651

652

B. Revision History

653 [optional; should not be included in OASIS Standards]

654

Revision	Date	Editor	Changes Made
WD 1		Erik Rissanen	Initial conversion to XACML 3.0.
WD 2	28 Dec 2007	Erik Rissanen	Conversion to the current OASIS template.
WD 3	4 Nov 2008	Erik Rissanen	Update to XACML core working draft 7.
WD 6	24 March 2009	Hal Lockhart	Added definitions provided by Rich Levinson Separated Attribute and URI modes Added conformance section
WD 8	5 April 2009	Erik Rissanen	Editorial cleanups.
WD 9		Erik Rissanen	Added non-normative pseudo-code (by Rich) for how one can collect the required attributes from a hierarchy.
WD 10	14 Dec 2009	Erik Rissanen Rich Levinson	Fixed typos. Updated URI scheme with XML node pointers. Remove ancestor attributes from XML scheme. Clarified meaning of metadata identifiers. Use the content-selector attribute instead of the resource-id in the XML scheme. Fixed 2.0 -> 3.0 typos in some identifiers.
WD 11	17 Dec 2009	Erik Rissanen Rich Levinson	Fixed typos Fixed OASIS references Updated acknowledgments
WD 12	12 Jan 2010	Erik Rissanen	Updated cross references Updated acknowledgments
WD 13	8 Mar 2010	Erik Rissanen	Updated cross references Fixed OASIS formatting issues

655

656