XACML v3.0 Hierarchical Resource Profile Version 1.0

Committee Draft 01

16 April 2009

Specification URIs:
This Version:
http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cd-1-en.pdf
http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cd-1-en.doc (Authoritative)
http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cd-1-en.html

Previous Version:
N/A

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, WD 11

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

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

{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 hierachies.

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

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

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

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

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

1.1 Glossary

DAG

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

Hierarchical resource

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

Hierarchy

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

Multi-rooted disjoint hierarchy

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

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

Multi-rooted hierarchy

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

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

Node

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

Single-rooted hierarchy

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

1.1.1 Comparison of hierarchical structures

The following table is intended to capture the salient features of the hierarchical structures used in this 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.**

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

1.2 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]**.

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

```
Example code listings appear like this.
```

In descriptions of syntax, elements in angle brackets ("<", ">") are to be replaced by appropriate values, square brackets ("[", "]") enclose optional elements, elements in quotes are literal components, and "*" indicates that the preceding element may occur zero or more times.

1.3 Normative References

[ISO10181-3] ISO/IEC JTC 1, *Information technology -- Open Systems Interconnection -- Security frameworks for open systems: Access control framework,* ISO/IEC 10181-3:1996, 1996.

[RFC1034] P. Mockapetris, *DOMAIN NAMES – CONCEPTS AND FACILITIES,* IETF RFC 1034, November 1987, http://www.ietf.org/rfc/rfc1034.txt

[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.

[RFC2396] T. Berners-Lee, et al., Uniform Resource Identifiers (URI): Generic Syntax, http://www.ietf.org/rfc/rfc2396.txt, IETF RFC 2396, August 1998.

[RFC3198] A. Westerinen, et al., *Terminology for Policy-Based Management,* http://www.ietf.org/rfc/rfc3198.txt, IETF RFC 3198, November 2001.

[MULTIPLE] E. Rissanen, ed., *XACML v3.0 Multiple Resource Profile Version 1.0, Working draft 5*, 5 April 2009,FIXME URL

[XACML] E. Rissanen, ed., *eXtensible Access Control Markup Language (XACML) Version 3.0,* Working draft 11, 5 April 2009,FIXME URL

[XPath] *XML Path Language (XPath),* Version 1.0, W3C Recommendation 16, November 1999. Available at http://www.w3.org/TR/xpath

1.4 Non-Normative References

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

2 Representing the identity of a node

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

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

2.1 Nodes in XML documents

{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:2.0:profile:hierarchical:xml-node-id

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

2.2 Nodes in hierarchical resources identified by URIs

{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:URI-node-id.

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

<scheme> “://” <authority> “/” <pathname>

File system resources SHALL use the “file:” scheme. If the resource is identified with a standard <scheme> specified in **[RFC2396]** or in a related standard for a registered URI scheme which is hierachical with slashes, then that scheme SHALL be used. Otherwise the URI SHALL use the “file:” scheme.

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

<root name> [“/” <node name>]*

- The sequence of <root name> and <node name> values SHALL correspond to the individual hierarchical component names of ancestors of the represented ***node*** along the path from a <root> ***node*** to the represented ***node***.
- The components of the <pathname> portion of the URI SHALL be specified using the canonical form for such path components at the <authority>.
- In accordance with **[RFC2396]**, the separator character between hierarchical components of the <pathname> portion of the URI SHALL be the character “/“. Sequences of the “/“ character SHALL be resolved to a single “/“. ***Node*** identities SHALL NOT terminate with the “/“ character.
- All <pathname> values SHALL be absolute.

- If there is more than one fully resolved, absolute path from a <root> at the <authority> to the represented ***node***, then a separate resource attribute with `AttributeId` "urn:oasis:names:tc:xacml:1.0:resource:resource-id" and `DataType` http://urn:oasis:names:tc:xacml:1.0:data-type:anyURI SHALL be present in the Request Context for each such path.

Implementation note: the scheme name of the URI should be checked to determine it is an expected scheme before parsing the URI into its hierarchical components.

Also note that the notion of parsing the syntax of a URI is controversial, see for example [URIOpacity].

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

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

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

3 Requesting access to a node

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

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

3.1 Nodes in an XML document

{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:xml-node-req

The attributes with `AttributeId`s 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 supported for use in resources represented as XML documents, the following URIs SHALL be used as identifiers for the functionality they represent:

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

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

- A `<Content>` element that contains the entire XML document instance of which the requested ***node*** is a part.
- An `<Attribute>` element with an `AttributeId` of "urn:oasis:names:tc:xacml:1.0:resource:resource-id" and a `DataType` of "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression". The `<AttributeValue>` of this `<Attribute>` SHALL be an XPath expression whose context node SHALL be the `<Content>` element in the "urn:oasis:names:tc:xacml:3.0:attribute-category:resource" attribute category. This XPath expression SHALL evaluate to a nodeset containing the single node in the `<Content>` element that is the ***node*** to which access is requested. This `<Attribute>` MAY specify an `Issuer`.
- An `<Attribute>` element with an `AttributeId` of "urn:oasis:names:tc:xacml:2.0:resource:resource-parent" and a `DataType` of "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression". The `<AttributeValue>` of this `<Attribute>` SHALL be an XPath expression; the context node for this XPath expression SHALL be the `<Content>` element in the "urn:oasis:names:tc:xacml:3.0:attribute-category:resource" attribute category. This XPath expression SHALL evaluate to a nodeset containing the single node in the `<Content>` element that is the immediate parent of the ***node*** represented in the "resource-id" attribute. This `<Attribute>` MAY specify an `Issuer`.
- For each node in the XML document instance that is an ancestor of the ***node*** represented by the "resource-id" attribute, an `<Attribute>` element with an `AttributeId` of

"urn:oasis:names:tc:xacml:2.0:resource:resource-ancestor" and a `DataType` of "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression". The `<AttributeValue>` of this `<Attribute>` SHALL be an XPath expression; the context node for this XPath expression SHALL be the `<Content>` element in the "urn:oasis:names:tc:xacml:3.0:attribute-category:resource" attribute category. This XPath expression SHALL evaluate to a nodeset containing the single node in the `<Content>` element that is the respective ancestor of the ***node*** represented in the "resource-id" attribute. For each "resource-parent" attribute, there SHALL be a corresponding "resource-ancestor" attribute. This `<Attribute>` MAY specify an `Issuer`.

- For each node in the XML document instance that is an ancestor of the ***node*** represented by the "resource-id" attribute, and for the "resource-id" ***node*** itself, an `<Attribute>` element with an `AttributeId` of "urn:oasis::names:tc:xacml:2.0:resource:resource-ancestor-or-self" and a `DataType` of "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression". The `<AttributeValue>` of this `<Attribute>` SHALL be an XPath expression; the context node for this XPath expression SHALL be the `<Content>` element in the "urn:oasis:names:tc:xacml:3.0:attribute-category:resource" attribute category. This XPath expression SHALL evaluate to a nodeset containing the single node in the `<Content>` element that is the respective ancestor of the ***node*** represented in the "resource-id" attribute, or that is the "resource-id" ***node*** itself. For each "resource-parent" and "resource-id" attribute, there SHALL be a corresponding "resource-ancestor-or-self" attribute. This `<Attribute>` MAY specify an `Issuer`.

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

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

3.2 Nodes in hierarchical resources identified by URIs

{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:URI-node-id.

The resource SHALL be identified by means of a hierarchical URI (or URIs) as described in section 2.2. 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 hiearchy in any way.

All that is required is that the Context Handler be able to determine what hierarchies exist, what are the recource-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 members.
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 <AttributeValue> of this <Attribute> MAY be of any XACML datatype This <Attribute> MAY specify an Issuer.
- For each ancestor of the ***node*** specified in the "resource-id" attribute or attributes, and for each representation of that ancestor ***node***, an <Attribute> element with AttributeId "urn:oasis:names:tc:xacml:2.0:resource:resource-ancestor". The <AttributeValue> of this <Attribute> SHALL be an identifier of the ancestor ***node***. The DataType of the <AttributeValue> of this <Attribute> MAY be of any XACML datatype This <Attribute> MAY specify an Issuer.
- For each ancestor of the ***node*** specified in the "resource-id" attribute or attributes, and for each representation of that ancestor ***node***, and for each representation of the "resource-id" ***node*** itself, an <Attribute> element with AttributeId "urn:oasis:names:tc:xacml:2.0:resource:resource-ancestor-or-self". The <AttributeValue> of this <Attribute> SHALL be an identifier of the ancestor ***node*** or of the "resource-id" ***node*** itself. The DataType of the <AttributeValue> of this <Attribute MAY be of any XACML datatype. This <Attribute> MAY specify an Issuer. Additional attributes MAY be included in the <Attributes> element.

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

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

```
// Define a class for "Resource Hierarchy identifier" node
public class ResHierId(int res, int hier)

// Define Sets to collect nodes in:
selfNodes = new HashSet<ResHierId>();
parentNodes = new HashSet<ResHierId>();
ancestorNodes = new HashSet<ResHierId>();
ancestorOrSelfNodes = new HashSet<ResHierId>();

// Define number of resources, hierarchies and 1-based 2-d array
int nRes=4, mHier=5; // example hierarchy dims
int[][] ijResource = new int[nRes+1][mHier+1];

// Define method to collect nodes
collectAncestorNodes(int iRes) {
  for (int j = 1; j<mHier+1; j++){
    int mDag = 1; m=j; iDepth = 0;
    if (ijResource[0][j] != 0){
      while ((m<mHier) && (ijResource[0][m+1] == ijResource[0][j])){
        mDag++; m++;
    } }
    walkUpHierarchyDag(iRes, j, mDag, iDepth);
    j=j+mDag-1; // skip columns handled by mDag
} }

walkUpHierarchyDag(int iRes, int j, int mDag, int iDepth){
  // for each instance of self in Dag subrow
  for (int k=1; k<mDag+1; k++){
    int m = j+k-1; // m is column in big matrix
    int iResCurrent = iRes; // iResCurrent is 1-based row-id
    if (ijResource[iResCurrent][m] != 0){
      ResHierId rhId = new ResHierId(iResCurrent,m);
      if (iDepth == 0){
        selfNodes.add(rhId);
        ancestorOrSelfNodes.add(rhId);
      }
      else if (iDepth == 1){
        parentNodes.add(rhId);
        ancestorNodes.add(rhId);
        ancestorOrSelfNodes.add(rhId);
      }
      else {
        ancestorNodes.add(rhId);
        ancestorOrSelfNodes.add(rhId);
      }
      if (iResCurrent != ijResource[iResCurrent][m]) {
        // Set the new current node as parent of current node
        iResCurrent = ijResource[iResCurrent][m];
        iDepth++;
        walkUpHierarchyDag(iResCurrent, j, mDag, iDepth);
      }
      else { } // found root on this path - done
    }
    else { } // zero means node not used - done
} }
```

Note the following:

- The matrix, ijResource[nRes+1][mHier+1] represents a collection of nRes resources, each of which may belong to any of mHier single-parent hierarchies, or a mix of hierarchies and DAGS.
- DAGs are represented by multiple columns, where the width of the DAG is mDag, which is equal to the maximum number of parents that a single node in the DAG currently has. It is assumed

that the matrix has been prepared such that all columns within a single DAG are adjacent. Each DAG has a unique “DAG-id”, which is present in row 0 of each column of the DAG. By contrast, single-parent hierarchies (single column) have a zero in row 0.

- The matrix is generally sparse, is initialized to all zeroes, and single-parent hierarchies and DAGs are built by assigning the row number (effectively resource-id) of the parent of the resource to the cell in resource’s row, effectively making the row a collection of potential hierarchies and DAGs that the resource can belong to. The root of a hierarchy is indicated by the row element pointing to the current row, a self-reference.
- The 2-d array is “one-based” in that column 0 and row 0 are not used so that resources and hierarchies may be identified as running from 1->nRes and 1->mHier.
- Once the matrix is built, the ancestors for a resource may be collected by passing the row number of the resource to the collectAncestorNodes(iRes) method. For each hierarchy and DAG in the matrix, the recursive walkUpHierarchyOrDag(res-id, hier-id, dag-width, depth) method is called, which will collect all the ancestors of either a hierarchy or DAG.
- The collected ancestors are stored in 4 sets: one each for self, parent, ancestor, and ancestor-or-self.
- This algorithm is intended to be a model only and does not represent any specific implementation strategy, except to clearly identify a concrete framework for identifying all the resources and hierarchies and DAGs that are potentially covered by this profile.

4 Stating policies that apply to nodes

{Non-normative}

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

4.1 Policies applying to nodes in an XML document or with ancestor attributes

{Non-normative}

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

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

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

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

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

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

4.2 Policies applying only to nodes in XML documents

{Non-normative}

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

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

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

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

4.3 Policies applying only to nodes identified with URIs

{Non-normative}

For ***hierarchical resources*** that are not represented as XML document instances, and where the URI representation of ***nodes*** specified in Section 2.2 of this Profile is used, the following functions described

in the XACML 3.0 Specification **[XACML]** MAY be used to state policies that apply to one or more ***nodes*** in that resource.

urn:oasis:names:tc:xacml:1.0:function:anyURI-equal

urn:oasis:names:tc:xacml:2.0:function:regexp-uri-match

5 New attribute identifiers

{Optional}

5.1 document-id

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

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

5.2 resource-parent

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

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

5.3 resource-ancestor

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

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

5.4 resource-ancestor-or-self

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

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

6 New profile identifiers

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

Section 2.1: Nodes in XML documents

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

Section 2.2: Nodes in resources that are not XML documents

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

Section 3.1: Nodes in an XML document

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

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

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

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

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

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

- 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

7 Conformance

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

7.1 Nodes in XML documents

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

- urn:oasis:names:tc:xacml:2.0:profile:hierarchical:xml-node-id

7.2 Nodes in hierarchical resources identified by URIs

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

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

7.3 Nodes in hierarchical resources identified by ancestor attributes

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

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

A. Acknowledgements

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

Participants:

Anthony Nadalin
Bill Parducci
Daniel Engovatov
Erik Rissanen
Hal Lockhart
Michiharu Kudo
Michael McIntosh
Ron Jacobson
Seth Proctor
Steve Anderson
Tim Moses

B. Revision History

[optional; should not be included in OASIS Standards]

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.