


Fedora XACML Policy Writing Guide

Under Construction

 This page is in the process of being edited. – DWD

With a Reference Collection of Sample Policies for Fedora

1 Introduction

XACML provides a very flexible language for expressing access control policies. This document offers guidance on writing a range of useful policies for Fedora such as

- Broad repository-wide policies for controlling access to Fedora API operations
- Specific repository-wide policies for controlling access to groups of digital objects based on various attributes
- Fine-grained object-specific policies for controlling access to individual digital objects.

This guide also provides a collection of sample XACML policies intended as reference material to help in writing custom XACML policies for Fedora. The sample policies demonstrate one possible authoring style for XACML; there are other ways to write XACML policies that have the same effect. Most of the sample repository-wide policies are authored to have a **single effect**, meaning that each policy has a single rule that either permits or denies access. This style of policy writing results in many individual policies, but each policy is atomic and uncomplicated. An alternative is to have fewer policies, each with multiple rules within. This multi-rule approach can result in more complicated policies, but is, nevertheless, appropriate for writing **object-specific policies** in Fedora where a single policy states all the rules for a particular digital object. In either case, it is essential to understand the policy combining algorithm that is configured for your repository's XACML-based Authorization module. By default, the "Deny Overrides" algorithm is configured in Fedora, which means that when multiple policies are applicable to an incoming request, deny will trump permit. As you add new policies to the mix, you must be aware of what kinds of policies are already active in the repository. Also, when writing a policy that contains more than one rule, you must understand the rule combining algorithm (which is specified in the root element of an individual XACML policy). The sample policies use the "first-applicable" rule combining algorithm, which means that the first applicable rule in the policy will prevail.

This document is not intended to be a comprehensive tutorial on writing XACML policies. Anyone intending to author custom XACML policies for Fedora is encouraged to read the following documentation provided by OASIS Technical Committee that defined the XACML standard, and Sun who is the provider of the open source Sun XACML engine that is used in the Fedora implementation. It is very important to understand the basics of XACML to ensure that a suite of policies works as intended. One of the most important concepts in using XACML is understanding how multiple policies can interact with each other (in good ways, or in ways you didn't intend). By following the examples in this guide, you should be able to set up many kinds of access control policies for your repository. With additional help from the following documents, you should be able to do more advanced policies, and change some of the XACML settings for how sets of policies are combined.

More information on writing XACML policies:

[OASIS XACML Specification](#): this is the official specification and a good reference document

[A Brief Introduction to XACML](#): this is nice introduction to the XACML concepts

[OASIS XACML Technical Committee](#): this home page of the technical committee provides access to other documents on XACML

More information on the Fedora security architecture:

[Securing Your Fedora Repository](#): documentation on security options and configuring configuring user authentication sources for Fedora repositories

[Fedora Authorization with XACML Policy Enforcement](#): documentation on configuration and implementation of the Fedora XACML-based policy enforcement module

[Binding to user attributes to policies](#): a discussion of how to use attributes from different sources (e.g., Tomcat, LDAP, Shibboleth) in policies

2 Writing Fedora XACML Policies

2.1 The Fedora Policy Vocabulary

A [Fedora-specific policy vocabulary](#) is defined to enable the creation of XACML policies for Fedora repositories and digital objects. This vocabulary define a set of URNs that can be used to identify specific Fedora API operations, Fedora object attributes, and the Fedora environment within an XACML policy. These URNs are used as attribute designators in XACML policies, specifically within a SubjectAttributeDesignator, ResourceAttributeDesignator, ActionAttributeDesignator, or EnvironmentAttributeDesignator.

The set of identifiers defined for the Fedora policy vocabulary can be found in the Fedora software distribution at:

```
$FEDORA_HOME\server\fedora-internal-use\vocabulary.txt
```

This vocabulary provides a set of identifiers (URNs) that can appear in XACML policies to refer to Fedora API operations (Actions in XACML), any aspects of a Fedora digital object (Resources in XACML), key attributes of the environment in which Fedora runs in (Environment in XACML), and common subject (i.e., user) attributes. (Other user attributes are named according to site-usage and so their names aren't included in the Fedora XACML vocabulary.)

2.2 Policy Basics: Identifier, Description, and Rule Combining Algorithm

Every policy has an identifier, a rule combining algorithm, and a description. In the root element of an XACML policy there is an attribute to provide the policy with a unique identifier. Also, the <Description> element provides a place to put a textual description of the purpose of the policy.

```
<Policy PolicyId="deny-apia"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable"
  xmlns="urn:oasis:names:tc:xacml:1.0:policy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <Description>This policy will DENY access to THESIS datastreams.</Description>

  <Target>
    ...
  </Target>

  <Rule>
    ...
  </Rule>

</Policy>
```

The main body of a policy consists of a **Policy Target** and one or more **Rules** which are described in the next sections. Note that in the root element of a policy, the rule combining algorithm (i.e., attribute "RuleCombiningAlgId"), specifies how the Fedora Policy Enforcement Module will deal with multiple Rules in a policy (how those rules are combined and evaluated together). This algorithm is valid for only the specific policy containing it, and is independent of similar algorithms in other policies. It governs how the various effects of the potentially several rules of a policy are combined into the single effect of the policy as a whole. It is also independent of the policy-combining algorithm operative for all policies collectively, which governs how the various results of all policies are combined into a single result.

2.3 Defining the Policy Target

A Policy Target is the part of a policy that specifies matching criteria for figuring out whether a particular policy is applicable to an incoming service request. A Target contains three basic "matching" components: **Subjects**, **Actions**, and **Resources**. All of these components must be matched to the context of an incoming request for the policy to be applicable. These matching specifications can be built upon [XACML Functions](#). (A fourth matching component, **Environments**, is defined in XACML and will be available in Fedora's XACML policies when it is available in the Sun XACML version as used in Fedora.)

```
<Policy PolicyId="deny-apia"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable"
  xmlns="urn:oasis:names:tc:xacml:1.0:policy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <Description>This policy will DENY access to THESIS datastreams.</Description>

  <Target>

    <Subjects>
      ...
    </Subjects>

    <Resources>
      ...
    </Resources>

    <Actions>
      ...
    </Actions>

  </Target>

  <Rule/>

</Policy>
```

A Policy Target can be specified for a Policy (or for a PolicySet, which is an advanced way of grouping policies together). A <Target> element is defined at the Policy level (as a child of the root <Policy> element). A Policy Target applies to any contained Rules that are expressed in that policy. However, a Rule may have its own Target, in which case the Rule-level Target overrides — for that Rule only — the Policy level Target. Typically, a Target defined at the Rule level is used to replace and so tighten a broader match specification found at the overall Policy level. (This is described below.)

Resources

All Fedora resources (objects and datastreams) have attribute identifiers defined in the Fedora policy vocabulary (see: ^vocabulary.txt).

The **<Resources>** element of a Policy Target is used to wrap one or more descriptions of the kinds of Fedora resources (objects, datastreams, disseminations, etc.) that the policy should apply to. At runtime, the Policy Enforcement Module will compare attributes of a requested resource against the criteria in the <Resources> specification within the policy Target to determine if the policy is applicable to the incoming request. For example, to define a policy that is applicable to any Fedora resource, the following is specified:

```
<Resources>
  <AnyResource/>
</Resources>
```

Within a single <Resource> specification, there may be one or more attributes that together determine whether a policy match should occur. Each <ResourceMatch> element is used to specify the name/value of an attribute of a Fedora resource.

If multiple <ResourceMatch> elements are specified, they will be logically **ANDed** together, meaning that for a policy to be applicable to an incoming service request, **all** <ResourceMatch> specifications must match the attributes of the requested Fedora resource. The AttributeID in the <ResourceAttributeDesignator> element is used to identify a particular resource attribute by a URN, as defined in the Fedora policy vocabulary. In the example below, there are two attributes to match on: "urn:...datastream:id" and "urn:...mimeType". The snippet says that a policy match will occur if the incoming request context indicates that the requested resource has the datastream id of THESIS and the MIME type of "application/pdf."

```
<Resources>
  <Resource>
    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <ResourceAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:resource:*datastream:id*"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        THESIS
      </AttributeValue>
    </ResourceMatch>
    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <ResourceAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:resource:datastream:mimeType"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        application/pdf
      </AttributeValue>
    </ResourceMatch>
  </Resource>
</Resources>
```

To create an **OR condition** for resource matching, multiple <Resource> elements must be specified. If there are multiple **<Resource>** elements within the <Resources> wrapper component, the <Resource> elements are **logically OR-ed together**. This means that a match on only one of the Resource specifications is necessary for the policy to apply to a service request. For example, the snippet below says that a resource match will occur if the incoming request is for a digital object that has the content model type of either "UVA_STD_IMAGE" or "MRSID."

```

<Resources>
  <Resource>
    <ResourceMatch
      MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          UVA_STD_IMAGE
        </AttributeValue>
        <ResourceAttributeDesignator
          AttributeId="urn:fedora:names:fedora:2.1:resource:object:contentModel"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ResourceMatch>
      </Resource>
    <Resource>
      <ResourceMatch
        MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            MRSID
          </AttributeValue>
          <ResourceAttributeDesignator
            AttributeId="urn:fedora:names:fedora:2.1:resource:object:contentModel"
            DataType="http://www.w3.org/2001/XMLSchema#string" />
          </ResourceMatch>
        </Resource>
      </Resources>

```

Actions

All Fedora service operations have an action identifier in defined by the Fedora policy vocabulary (see: ^vocabulary.txt).

The **<Actions>** element of a Policy Target is used to wrap one or more service operations that this policy should apply to. At runtime, the Policy Enforcement Module will compare the identity of an incoming request against the criteria specific in the <Actions> of a Target in a policy. For example, to define a policy that is applicable to **any** Fedora service operation, the following is specified:

```

<Actions>
  <AnyAction/>
</Actions>

```

From a practical standpoint in Fedora, there are only two attributes that pertain to identifying Fedora API operations:

1. an attribute that indicates what Fedora API is in context
2. an attribute that indicates the specific service operation within that API.

To create a policy that is intended for an *entire* service (e.g., ALL operations of API-A) do the following:

```

<Actions>
  <Action>
    <ActionMatch
      MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          urn:fedora:names:fedora:2.1:action:api-a
        </AttributeValue>
        <ActionAttributeDesignator
          AttributeId="urn:fedora:names:fedora:2.1:action:api"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ActionMatch>
      </Action>
    </Actions>

```

To create a policy that is about a specific operation in a Fedora API do the following:

```

<Actions>
  <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">
        urn:fedora:names:fedora:2.1:action:id-getDatastreamDissemination
      </AttributeValue>
      <ActionAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:action:id"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </ActionMatch>
    </Action>
  </Actions>

```

The above can be considered a shortcut for fully qualifying a service operation within its respective service API. An alternative way to specify an Action as a Fedora API operation is to refer to the Fedora service API **AND** the service operation. As with <SubjectMatch> and <ResourceMatch> specifications, multiple **<ActionMatch> elements are *logically AND-ed together**. For example the following snippet says that the policy will match if the incoming request pertains to the Fedora API-A service AND the service request is for the "getDatastreamDissemination" operation.

```

<Actions>
  <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        urn:fedora:names:fedora:2.1:action:api-a
      </AttributeValue>
      <ActionAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:action:api"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </ActionMatch>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        urn:fedora:names:fedora:2.1:action:id-getDatastreamDissemination
      </AttributeValue>
      <ActionAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:action:id"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </ActionMatch>
    </Action>
  </Actions>

```

To create an **OR condition** for action matching, multiple <Action> elements must be specified. If there are multiple **<Action> elements** within the <Actions> wrapper component, the <Action> elements are **logically OR-ed together**. This means that a match on only one of the Action specifications is necessary for the policy to apply to a service request. For example, the snippet below says that a resource match will occur if the incoming request is **either** the getDatastreamDissemination operation of API-A or the getDissemination operation of API-A.

Note: The "shortcut" method of referring to a Fedora API operation is used in the example (i.e., we have an ActionMatch for the specific Fedora API of "api-a") because Fedora actions identifiers are unique in themselves. Fedora automatically knows that the getDatastreamDissemination operation is part of API-A.

```

<Actions>
  <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        urn:fedora:names:fedora:2.1:action:id-getDatastreamDissemination
      </AttributeValue>
      <ActionAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:action:id"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
    </ActionMatch>
  </Action>
  <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        urn:fedora:names:fedora:2.1:action:id-getDissemination
      </AttributeValue>
      <ActionAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:action:id"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
    </ActionMatch>
  </Action>
</Actions>

```

Subjects

The Fedora policy vocabulary (see: ^\vocabulary.txt) defines general-purpose attributes for use in policies (e.g., login-id). However, attributes for subjects will vary depending on what a repository uses as the source of user information (e.g., tomcat-users.xml, LDAP, Shibboleth). Fedora XACML policies can make reference to the identifiers of any subject attribute that can be passed into Fedora from authenticating sources. See the section below on "Getting User Attributes into the Fedora Policy Enforcement Module" for more information on the sources of subject attributes.

The **<Subjects>** element of a Policy Target is used to wrap one or more descriptions of users or agents that this policy should apply to. At runtime, the Fedora Policy Enforcement Module will compare attributes of the user/agent making a service request against the criteria specific in the **<Subjects>** specification of the policy Target to determine if the policy is applicable to the incoming request. For example, to define a policy that is applicable to any kind of user or agent, the following is specified:

```

<Subjects>
  <AnySubject/>
</Subjects>

```

Within a single **<Subject>** specification, there may be one or more XACML attributes that together determine whether a policy match should occur. Each **<SubjectMatch>** element is used to specify an name/value of an attribute of a user/agent. Multiple **<SubjectMatch>** *elements are used to specify multiple attributes of a subject, and are ***logically AND-ed together**. This means that for a policy to be applicable to an incoming service request, **all** **<SubjectMatch>** specifications must match the attributes of the requesting user/agent. In the example below, there is only one attribute to match on (i.e., "fedoraRole"). The AttributeId in the **<SubjectAttributeDesignator>** element is used to identify a particular subject attribute by its local or global identifier. The snippet says that a policy match will occur if the incoming request context indicates that the user/agent has a role attribute with the value of "administrator."

```

<Subjects>
  <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        administrator
      </AttributeValue>
      <SubjectAttributeDesignator AttributeId="fedoraRole" MustBePresent="false"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
    </SubjectMatch>
  </Subject>
</Subjects>

```

To create an **OR condition** for subject matching, multiple **<Subject>** elements must be specified. If there are multiple **<Subject>** elements within the **<Subjects>** wrapper component, the **<Subject>** elements are **logically OR-ed together**. This means that a match on only one of the Subject specifications is necessary for the policy to apply to a service request. For example, the snippet below says that a subject match will occur if the requesting user has the role of either "administrator" or "superuser."

```

<Subjects>
  <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        administrator
      </AttributeValue>
      <SubjectAttributeDesignator AttributeId="fedoraRole" MustBePresent="false"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
    </SubjectMatch>
  </Subject>
  <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        superuser
      </AttributeValue>
      <SubjectAttributeDesignator AttributeId="fedoraRole" MustBePresent="false"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
    </SubjectMatch>
  </Subject>
</Subjects>

```

Environments

The Environments component of a Target is intended to specify aspects of the runtime environment that would make the policy match the incoming request. Such attributes include the current date, current time, the IP address of the client, and the protocol being used for the request. The **Environments** element is discussed in the OASIS XACML specification, but it is ***not yet implemented by the Sun XACML engine*** that underlies the Fedora Authorization module. This prevents the expression of environment matching criteria within Targets.

Therefore, do not create policies that specify Environment matching criteria in the policy Target.

Although the Environments element is not currently supported for use within a Target, this does not mean that you cannot encode matching criteria for environmental attributes within a policy. Within a policy Rule, you can specify a Condition that contains matching criteria for environmental attributes. Refer to the discussion of policy rules below for an example of Environment attribute matching in a Condition.

2.4 Defining Policy Rules

Each policy has at least one, and possibly more, rules. There must be at least one Rule in a policy that matches the incoming request for a policy to be deemed applicable to that request. The way the Sun XACML engine determines whether a rule is applicable to an incoming request is by evaluating the Target and optional Condition (if it exists). These are ANDed together, and the rule's effect achieved if the ANDed value is TRUE. (If there is no Condition, this result is simply the value of the Target.) The rule's Target is so used, and if it has no Target, the policy's Target is used instead.

```

<Policy PolicyId="deny-apia"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable"
  xmlns="urn:oasis:names:tc:xacml:1.0:policy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <Description>This policy will DENY access to Dublin Core datastreams.</Description>

  <Target>
    ...
  </Target>

  <Rule RuleId="1" Effect="Deny">

    <Target>
      ...
    </Target>

    <Condition>
      ...
    </Condition>

  </Rule>
</Policy>

```

Rule

A policy contains one or more Rules. Each rule has a **RuleId** and an **Effect**. An Effect is the intended consequence of a satisfied rule, which can be either "Deny" or "Permit." This means that if the rule is deemed applicable to an incoming service request, and the rule's conditions evaluate to TRUE, then the specified effect should be enforced.

Target

Each Rule in a policy can have its own Rule Target. While a Policy Target generally describes the kinds of requests to which an entire policy applies, a Rule Target describes the kinds of request to which a particular rule applies. If a Rule Target is not present, the Policy Target is used to determine whether the Rule is applicable to an incoming request. When a policy target exists, it is applicable to every rule in the policy which does not have its own Target. In practice, a rule target is often more constrained than the associated policy target, fine tuning to specific Subject/Resource/Action match criteria that are in the context of a the particular rule.

Refer to the documentation above for the [Defining a Policy Target](#) for the structure of a Target, since rule and policy Targets are defined using the same elements.

Condition

A Condition is a predicate that must be satisfied for a rule to be assigned its effect. These predicates can be built upon [XACML Functions](#).

While Targets are appealing, frame-like expressions, they have a constrained logic which isn't always expressive enough to narrow down whether a policy is applicable to a service request. Hence, the need for Condition elements. If either the policy Target or the rule Target is not able to adequately express a constraint, a Condition can be added to a Rule. **A Condition can appear only within a Rule.** It cannot appear within a Target, nor directly under Policy or PolicySet. If a Condition is intended to be applicable to the entire Policy, the Condition must be repeated in every Rule in that Policy. Unlike the relationship of rule targets to policy targets, conditions do in fact begin with the associate (rule or policy) target, and proceed to further constrain that target.

2.5 XACML Functions

The XACML specification defines numerous functions that can be used in defining attribute match criteria in Targets and in defining predicates for Conditions. Consult the [XACML Specification](#) for a complete list of functions with their descriptions. For convenience, here is a small sampling of convenient functions with their XACML identifiers:

- **Equality predicates**
 - **String Equality** – urn:oasis:names:tc:xacml:1.0:function:string-equal
 - **Boolean Equality** – urn:oasis:names:tc:xacml:1.0:function:boolean-equal
 - **Date/Time Equality** – urn:oasis:names:tc:xacml:1.0:function:date-time-equal
 - others
- **Logical functions**
 - **OR** – urn:oasis:names:tc:xacml:1.0:function:or
 - **AND** – urn:oasis:names:tc:xacml:1.0:function:and
 - **NOT** – urn:oasis:names:tc:xacml:1.0:function:not
 - others
- **Comparison functions**
 - **Greater Than** – urn:oasis:names:tc:xacml:1.0:function:integer-greater-than
 - **Less Than** – urn:oasis:names:tc:xacml:1.0:function:integer-less-than
 - **Greater Than or Equal** – urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal
 - **Less Than or Equal** – urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal
 - **Date/Time Greater Than** – urn:oasis:names:tc:xacml:1.0:function:date-time-greater-than
 - others
- **Bag and Set functions**
 - **Bag of Strings** – urn:oasis:names:tc:xacml:1.0:function:string-bag
 - **Member of Set** – urn:oasis:names:tc:xacml:1.0:function:type-at-least-one-member-of
 - others

Below is an example Condition that uses several of these functions. This Condition evaluates to TRUE if the client IP address (from the environment of the incoming request) is NOT a member of a set of privileged IP addresses. The Condition element itself contains an outer-most function which is a **negation function**. Within the condition, we see the application of the **set membership function**, which specifies that the environment attribute "clientIpAddress" (from the Fedora vocabulary) should be evaluated. Finally, the inner most **bag function** wraps a set of possible values for the clientIpAddress attribute. Again, if the clientIpAddress on the incoming request is not one of those in the bag of addresses, then the rule's Deny effect should take place.


```

<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
    <EnvironmentAttributeDesignator
      AttributeId="urn:fedora:names:fedora:2.1:environment:httpRequest:clientIpAddress"
      DataType="http://www.w3.org/2001/XMLSchema#string" />
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        127.0.0.1
      </AttributeValue>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        128.84.103.11
      </AttributeValue>
    </Apply>
  </Apply>
</Condition>

```

In summary, each policy must have at least one Rule. For a Rule to have an effect, (1) the Rule must match the incoming request by virtue of a Target match (either via a policy Target, or a constraining rule Target), and (2) if a Condition is specified, the condition predicate evaluates to TRUE. An applicable rule will result in a Permit or Deny for an incoming request, based on what is specified in the Rule Effect.

2.6 Required vs. Optional Attributes in a Policy

There are times when an attribute that is referred to by a policy target will not be available on an incoming service request. By default, when the policy matching activity occurs - and an attribute specified in a policy is not found in the incoming request context - an Indeterminate result is returned and an authorization exception is thrown. Policy authors can avoid unwanted Indeterminate results by indicating in the attribute designators of a Target or Condition that a particular attribute can be considered optional in terms of whether it must exist in the incoming request context. This is done by setting `MustBePresent="false"` on a SubjectAttributeDesignator, ResourceAttributeDesignator, ActionAttributeDesignator, or EnvironmentAttributeDesignator element. This will tell the Fedora Policy Enforcement module that it's ok if the incoming request does not have the specified attribute available within it. (The implicit/unstated default is `MustBePresent="true"`)

Let's take an example to make this clearer. Consider a policy where the SubjectMatch specification talks about an attribute "fedoraRole" and specifies that the value of this attribute must be "administrator" in order for this policy to be considered applicable by the PDP. Now consider an incoming service request that has a user login id (e.g., "wdn5e") in the request context, but this user does not have a "fedoraRole" attribute associated with it. So, when the PEP tries to determine whether this policy is applicable to the incoming service request, it returns INDETERMINATE because it can't figure out whether there is a subject match without the presence of a "fedoraRole" attribute. This will cause an authorization exception to be thrown for the request because the PDP expects the "fedoraRole" attribute to be present in the request context. However, we essentially want to somehow indicate that the fedoraRole attribute is considered "optional" on an incoming request (i.e., not every incoming request must have this particular attribute in context). To do this, you must indicate in the policy Target that the attribute does not have to be present (`MustBePresent="false"`) in the incoming request as follows:

```

<Subjects>
  <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        administrator
      </AttributeValue>
      <SubjectAttributeDesignator AttributeId="fedoraRole" MustBePresent="false"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
    </SubjectMatch>
  </Subject>
</Subjects>

```

In this example, it's easy to imagine that another policy could independently permit access. Hence the fit of `MustBePresent="false"`: if the policy above lacks an attribute, it may not be crucial to the ultimate authorization decision. Policies are not authored in isolation, but to work together.

2.7 Recommended Best Practices for Authoring Fedora XACML Policies

1. Set the PolicyId attribute in the XACML policy to match the filename of the policy.

```
<Policy PolicyId="deny-objects-to-students" . . . > – corresponds to filename of "/repository-policies/deny-objects-to-students.xml"
```

2. For object-specific policies, especially if kept in an XML file, set the PolicyID in the XACML and the policy filename to match the object PID, but with concession to demand of OS filenames (e.g., uses dash instead of colon).

```
<Policy PolicyId="demo-5" . . . > – corresponds to filename of "/object-policies/demo-5.xml"
```

3. Policies should use simplest rule-combining algorithm which gives desired outcome. Avoid a more complicated algorithm which happens to work, but which confuses because it implies more than what's there. A simple choice is the "first-applicable" rule combining algorithm which give precedence to the first rule in a policy to apply to a situation.

```
<Policy PolicyId="demo-5" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable" . . . >
```

4. An object-specific policy should be coded so that it applies only to that specific object. So-coded, if misplaced among general repository policies, that wouldn't be hurtful.
5. Try to stick with "single-effect" policies, that is a policy should either permit or deny. Things can get confusing if a single policy has some rules that permit and some that deny. If most policies are single-effect, try to have them all be single-effect. You may wind up writing more individual policies, some that deny and some that permit, but from a policy management standpoint, it is probably easier to have atomic, unambiguous policies.

2.8 XACML Gotchas

1. XACML provides for an AttributeValue in a <Target> evaluation as a single value, but provides for an AttributeValue in a <Condition> evaluation as "bags" (sets), doing so even for either singleton or empty bags. Code policies accordingly.
2. MatchId functions (which are used in Targets) are much restricted in allowed values, compared to the values allowed in the analogous FunctionIds (which are used in Conditions). There are no existing functions which are self-contained boolean combinations, such as not-equal. Since attributes are generally not boolean themselves (and so possibly negated), the not function can't be used as a MatchId, e.g., in a SubjectMatch element. Since SubjectMatch, e.g., expresses a single binary operation, there is no possibility of introducing negative logic into a Target. [An exception would be an explicit value returned by an attribute finder, which would signify the absence of the attribute.]
3. Despite some evidence that <Environments> was added to <Target> generally, it doesn't seem to work currently in sunxacml.
4. sunxacml has a relaxed parsing of policies; e.g., we have encountered schema violation (e.g., Action omitted between Actions and ActionMatch) which resulted only in the policy not being evaluated correctly, as opposed to failing parse. How widespread this is, we don't know. As a precaution, policies should be tested for effect. This is good practice, anyway, since testing is the only check of the policy-writer's understanding of xacml and against the inevitable typ0.
5. Though sunxacml parsing is relaxed, <Description> </Description> apparently requires at least one-character content: <Description/> doesn't do it.
6. In SubjectMatch, ResourceMatch, and ActionMatch blocks, place AttributeValue elements before AttributeDesignator. Also, avoid using two AttributeDesignator elements (without any AttributeValues). Though it may seem logical to use other ordering or attribute selection, it doesn't match the standard and won't work.

```
<ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#dateTime">
    2004-12-07T20:22:26.705Z
  </AttributeValue>
  <ResourceAttributeDesignator
    AttributeId="urn:fedora:names:fedora:2.1:resource:object:lastModifiedDate"
    DataType="http://www.w3.org/2001/XMLSchema#dateTime" />
</ResourceMatch>
```

3. Default Repository-Wide Policies

out-of-the-box, the Fedora repository is configured with a default set of access control policies that provide for a highly restricted management service (API-M), an open access service (API-A), and an open OAI provider service. These default access control policies establish the same level of out-of-the-box security on the repository that was previously configured for Fedora 2.0 release; however, as of Fedora 2.1 these basic access controls are now specified as XACML policies. Please consult the [Default Repository Policies](#) documentation for a description of each default policy.

4. Custom Policies - Sample Repository-Wide Policies

The sample policies are written with the assumption that Fedora's [Default Repository-Wide Policies](#) are unedited and activated. These default policies lock down access to the Fedora API-M service so that only the Fedora Administrator is permitted access. The default policies also result in open access to API-A (all users are permitted). Given this starting point, you can think writing custom policies as a way to "loosen up" the API-M defaults and "tighten up" the API-I defaults. In other words, it is likely that you will want to write policies that let more users have access to API-M operations. Also, you will likely want to add restrictions in who can access digital objects, datastreams, and disseminations (i.e., via API-A). The sample policies will demonstrate how to do these things, given various authentication scenarios. Notice that there are some policies which restrict access based on user identity/attributes based on Tomcat's default user directory (tomcat-users.xml). Other policies demonstrate how to restrict access by user attributes/groups that are defined in an LDAP directory.

Note that the sample policies have been written for demonstration purposes. They are not intended to work as a collaborative set of policies, since they often demonstrate different ways of doing the same basic thing (e.g., one policy demonstrating rules based on Tomcat user identity, another showing a similar thing with LDAP groups). If you want to try them out, you can put one or more of the sample policies into play by following the instructions for [activating and loading policies](#) into a Fedora repository. However, it is recommended that you test them individually to understand the effect each policy has. This approach of augmenting the default policies, which are left as-is, allows progressive learning, without endangering your repository. It may be that this approach goes farther in opening up API-M than in tightening up API-I, and that eventually the default policy for API-A will need to be replaced by one or more policies written to your site's needs. So it goes. Ultimately, you can proceed to write a meaningful suite of policies that are intended to work together for your repository.

4.1 Repository Policies to *tighten* the API-A defaults at the *service interface* level

P o l i c y	S e r v i c e	XACML Policy File	Policy Description
4 . 1 . 1	A P I - A	deny-apia-to-ldap-group.xml	Deny access to all API-A methods to users who are "Librarians" or "Info Technologists" (as indicated by their LDAP attributes).
4 . 1 . 2	A P I - A	deny-apia-if-not-tomcat-role.xml	This policy will DENY access to ALL API-A methods to users who are NOT in the "administrator" or "professor" ROLES.
4 . 1 . 3	A P I - A	deny-apia-to-tomcat-user.xml	This policy will deny access to all API-A methods to a particular user based on login id (as registered in the tomcat-users.xml file).
4 . 1 . 4	A P I - A	deny-apia-except-by-owner.xml	Deny access to all API-A methods to any user unless that user is the owner of the object being accessed. This sample policy primarily exists to show how to create a policy that compares the owner-id of an object to the login-id of the current user. It is important to note that due to how XACML policies are processed, you cannot do this comparison in the <Subject> section of the XACML policy. The comparison must appear in a <Condition> specification in the <Rule> section.

4.2 Repository Policies to *tighten* the API-A defaults based on *object attributes*

P o l i c y	S e r v i c e	XACML Policy File	Policy Description
4 . 2 . 1	A P I - A	deny-objects-by-pids-to-tomcat-role.xml	Overall, this policy will identify a set of objects by their PIDs and it will DENY ALL APIA access to users of particular ROLES. NOTE: As a repository-wide policy, this policy demonstrates how to control access to specific objects (identified by PID). As an alternative, it is possible to create "object-specific" policies that either resides in the digital object's POLICY datastream, or that is stored in the object-specific policy directory. (See the Fedora system documentation on XACML policies for more information.)
4 . 2 . 2	A P I - A	deny-objects-by-cmodel-to-ldap-group.xml	This policy will DENY all APIA access to digital objects that are EAD Finding AIDS. This is based on the object content model attribute having a value of "UVA_EAD_FINDING_AID." Specifically, the policy will DENY access to users that belong to a particular LDAP-defined GROUP.
4 . 2 . 3	A P I - A	deny-objects-hide-datastreams-if-not-tomcat-role.xml	The overall intent of this policy is datastream hiding, meaning that raw datastreams must not be accessible to anyone except very privileged users, but service-mediated disseminations are accessible by a broader audience. The key point is that students can access disseminations of the object, but not the raw datastreams. This might typically be done in cases where lesser privileged users are given a derivation of the main datastream, or a lesser quality view, or a less complete view of the raw datastream content. Given that an object is of a certain content model (in this case UVA_STD_IMAGE), this policy will DENY datastream access to users who do NOT have the ROLE of "administrator" or "professor". It will also DENY dissemination access to users who do NOT have the ROLE of "student," "administrator," or "professor."

4.3 Repository Policies to *tighten* the API-A defaults at the *datastream level*

P o l i c y	S e r v i c e	XACML Policy File	Policy Description
4 . 3 . 1	A P I - A	deny-apia-datastream-all-to-all-users.xml	This policy will DENY access to ALL datastreams. Specifically, it will DENY access to ALL USERS making requests to the getDatastreamDissemination method of API-A.
4 . 3 . 2	A P I - A	deny-apia-datastream-DC-to-all-users.xml	This policy will DENY access to Dublin Core datastreams. Specifically, it will DENY access to ALL users making getDatastreamDissemination requests on API-A to obtain datastreams with an identifier of 'DC.'

4.3.3	A-PI-A	deny-apia-datastream-DC-to-tomcat-group-ALT1.xml	This policy will DENY access to Dublin Core datastreams. Specifically, it will deny access to USER GROUPS making getDatastreamDissemination requests on API-A for datastreams with a datastream identifier of 'DC.' User groups are defined using custom roles in the tomcat-users.xml file.
4.3.4	A-PI-A	deny-apia-datastream-DC-to-tomcat-group-ALT2.xml	This policy will DENY access to Dublin Core datastreams. Specifically, it will deny access to USER GROUPS making getDatastreamDissemination requests on API-A for datastreams with a datastream identifier of 'DC.' User groups are defined using custom roles in the tomcat-users.xml file.
4.3.5	A-PI-A	deny-apia-datastream-MRSID-if-not-tomcat-role.xml	This policy will DENY access to MRSID image datastreams by controlling access to the getDatastreamDissemination method of the Fedora Access Service (API-A). Specifically, it will DENY access to users who are NOT of particular ROLES when the requested resource is a datastream with identifier of 'MRSID.'
4.3.6	A-PI-A	deny-apia-datastream-TEISOURCE-to-tomcat-user.xml	This policy will DENY access to TEI datastreams by controlling access to the getDatastreamDissemination method of the Fedora Access Service (API-A). The TEI datastream is identified as a Resource where the Fedora datastream id has the value of 'TEISOURCE.' This policy will DENY access to a SPECIFIC USER based on login id (as registered in the tomcat-users.xml file).

4.4 Repository Policies to *tighten* the API-A defaults at the *dissemination* level

Policy	Service	XACML Policy File	Policy Description
4.4.1	A-PI-A	deny-apia-dissem-demo1-getMedium-to-all-users.xml	This policy will DENY access to the 'demo:1/getMedium' dissemination (defined on a disseminator that subscribes to the demo:1 behavior definition. Specifically, it will DENY access to ALL users making getDissemination requests on API-A for the 'demo:1/getMedium' dissemination.
4.4.2	A-PI-A	deny-apia-dissem-demo1-getMedium-to-ldap-group.xml	This policy will DENY access to the 'demo:1/getMedium' dissemination (defined on a disseminator that subscribes to the demo:1 behavior definition. Specifically, it will DENY access to users of particular LDAP-defined GROUPS who are making getDissemination requests on API-A for the 'demo:1/getMedium' dissemination.
4.4.3	A-PI-A	deny-apia-dissem-demo1-getMedium-if-not-tomcat-role.xml	This policy will DENY access to the 'demo:1/getMedium' dissemination (defined on a disseminator that subscribes to the demo:1 behavior definition. Specifically, it will DENY access to users who are NOT of particular ROLES who are making getDissemination requests on API-A for the 'demo:1/getMedium' dissemination.
4.4.4	A-PI-A	deny-apia-dissem-demo1-getMedium-to-tomcat-user.xml	This policy will DENY access to disseminations that are available on objects via a disseminator subscribing to the 'demo:2' behavior definition. Specifically, it will DENY access to a particular user (as registered in the tomcat-users.xml file).
4.4.5	A-PI-A	deny-apia-dissem-DualResImage-to-all-users.xml	This policy will DENY access to ALL disseminations that are available on objects via a particular disseminator (one that subscribes to an image-based behavior definition whose PID is 'demo:1/DualResImage'). Specifically, it will DENY access to ALL users making getDissemination requests on this disseminator.

4.5 Repository Policies to *loosen* the API-M defaults at the *service interface* level

Policy	Service	XACML Policy File	Policy Description
4.5.1	API-M	permit-apim-by-ldap-group.xml	
4.5.2	API-M	permit-apim-by-tomcat-group.xml	
4.5.3	API-M	permit-apim-by-tomcat-user.xml	
4.5.4	API-A / API-M	permit-if-owner.xml	If the logged-in user is the owner of an object, permit all actions. Note: This policy also works if the object has multiple owners and the logged-in user is one of them.

5 Custom Policies - Sample Object-Specific Policies

5.1 Object-specific policies with multiple policy rules

Object-specific policies are policies that refer to one particular digital object. An object-specific policy is stored in the "POLICY" datastream of the digital object to which it pertains.

Policy	XACML Policy File	Policy Description
--------	-------------------	--------------------

5	de	By using multiple policy rules , this policy shows how to deny access to all raw datastreams in the object except to particular users (e.g., the object owners). It also shows how to deny access to a particular disseminations to selected user roles.
1	mo	
1	/A-5.	
1	xml	
5	de	By using multiple policy rules , this policy shows how to deny access to particular datastreams in the object. 1) The policy will DENY everyone except professors and researchers access to particular source datastreams of the demo:11 object by controlling access to the getDataStreamDissemination method of the Fedora Access Service (API-A). 2) The policy will DENY everyone except students, professors, and researchers, access to all disseminations of demo:11. 3) This policy will also DENY ALL access to the demo:11 object to a SPECIFIC USER based on login id (as registered in the tomcat-users.xml file). NOTE: The net effect of the policy permits open access to the descriptive metadata datastream of demo:11.
1	mo	
1	/A-1.	
2	xml	
5	de	By using multiple policy rules , this policy shows how to deny access to particular datastreams in the object. The policy will DENY visitors access to the TEI and FOP source datastreams of the demo:26 object by controlling access to the getDataStreamDissemination method of the Fedora Access Service (API-A). These datastreams are open to all other kinds of users, and Disseminations are open to all users. This is an object-specific policy. It could be stored inside the demo:26 digital object in the POLICY datastream OR in the directory for object-specific policies. (The directory location is set in the Authorization module configuration in the Fedora server configuration file (fedora.fcfig).
1	mo	
1	/A-2.	
3	6.	
3	xml	

Unable to render {include}

The included page could not be found.