WebAC Authorizations

In WebAC, an Authorization is a rule that describes who can access what, and how they can interact with the resource they can access. An Authorization is written as a series of RDF triples, with the Authorization resource as the subject for each of these triples. The Authorization resource URI is a hash URI, because of the requirement for potentially multiple, distinct Authorizations to be included in a single ACL resource. In this document, the subject `<#authz>` is used in the examples.

Prefixes used in this document:

```
@prefix acl:   <http://www.w3.org/ns/auth/acl#> .
@prefix ex:    <http://example.org/ns#> .
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix ldp:   <http://www.w3.org/ns/ldp#> .
@prefix pcdm:  <http://pcdm.org/models#> .
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .
@prefix webac: <http://fedora.info/definitions/v4/webac#> .
```

"Who?" - Users and Groups

The first part of the authorization describes who can access a resource. There are two ways to do this. The first is simply by naming particular users using the `acl:agent` property, as either strings or URIs. When using URIs, it is necessary to translate the string-based username that is used for container-based authentication (i.e. Tomcat/Jetty authentication) into a URI. This is done by prefixing the string with a URI that is set in the container configuration: `-Dfcrepo.auth.webac.userAgent.baseUri=http://example.org/user/` and/or `-Dfcrepo.auth.webac.groupAgent.baseUri=http://example.org/group/`

```
# as strings
<#authz> acl:agent "obiwan", "yoda" .

# as URI references
<#authz> acl:agent ex:obiwan, ex:yoda .
```

Fedora Implementation Note: This is a slight departure from the SOLID WebAC spec, where the object of the `acl:agent` property must be a URI. We chose to implement it in such a way that the WebAC authorization module supports both String Literals and URIs in order to ease the integration with existing authentication or single-sign-on systems that identify users with string usernames. See ACL Agents - Strings vs. URIs for further details.

However, listing individual users this way can get unwieldy, so you can also use the `acl:agentGroup` property to specify a group of users:

```
<#authz> acl:agentGroup </groups/jedi> .
```

The object of the `acl:agentGroup` property must be a URI to a group resource (of type `vcard:Group`) containing a list of its members:

```
# contents of </groups/jedi>, using strings to identify members:
<> a vcard:Group;
    vcard:hasMember "obiwan";
    vcard:hasMember "yoda";
    vcard:hasMember "luke" .

# contents of </groups/jedi>, using URIs to identify members:
<> a vcard:Group;
    vcard:hasMember ex:obiwan;
    vcard:hasMember ex:yoda;
    vcard:hasMember ex:luke .
```

Fedora Implementation Note: As currently implemented, the group resource must also be stored in Fedora; there is no support for referencing external URIs with the `acl:agentGroup` property.

"What?" - Resources and Resource Types

The next part of the authorization describes what resource can be accessed. As with the previous section on agents, there are also two ways to describe the resource. The first is to provide the URI to the resource using the `acl:accessTo` property:

```
<#authz> acl:accessTo </collections/rebels/plans> .
```

If you use `acl:accessTo` to protect a container, and add an `acl:default` predicate, that authorization rule by default will also apply to any of that container's children, unless that child has its own ACL.

Inheritable authorization using acl:default

```
<#authz> acl:accessTo </collections/rebels> ;
    acl:default </collections/rebels> .
```

The second is to use the `acl:accessToClass` property to state that the authorization rule applies to any resource with the named RDF type. For example, this authorization will apply to any pcdm:Container resources contained by /collections/rebels that do not have their own ACL:

```
<#authz> acl:accessToClass pcdm:Container ;
    acl:default </collections/rebels>
```

While Fedora will not prevent you from using `acl:accessToClass` without an `acl:default` statement on the same authorization, this is almost certainly not the behavior you want. Without the `acl:default` predicate to indicate that the authorization should be inheritable (see SOLID WebAC), the authorization will only apply to the protected resource if the protected resource has an RDF type that matches `acl:accessToClass`.

If your intent is to just protect the single resource, that intent is more clearly stated through using `acl:accessTo`. If your intent is to protect multiple resources based on their RDF type, then you will need the authorization to be inherited by using `acl:default` in conjunction with `acl:accessToClass`.

"How?" - Modes of Interaction

Finally, an authorization states how the users or groups can interact with the resource or type of resource. This is known as the mode of access, and is specified using the `acl:mode` property.

Mode	Meaning	Allowed HTTP Requests
`acl:Read`	read a resource	• GET • HEAD • OPTIONS
`acl:Write`	write to a resource	• PUT • POST • PATCH • DELETE
`acl:Append`	add to a resource	• POST to a LDP-RS • PATCH that only inserts triples to an LDP-RS
`acl:Control`	read and write a resource's ACL	• all methods, if the request URI is an ACL