

# Authentication and Authorization

- [Overview](#)
- [Servlet Container Configuration](#)
  - [Configure your repo.xml file](#)
  - [Configure your repository.json file](#)
  - [Configure your web.xml](#)
  - [Configure your web application container](#)
    - [Jetty](#)
    - [Tomcat](#)
- [Authorization Delegates](#)
  - [Overview](#)
  - [Fedora Administrators \(fedoraAdmin user role\)](#)
  - [FedoraAuthorizationDelegate Implementations](#)
  - [Step-by-step:](#)
  - [Example repo.xml \(repository and security beans\)](#)
  - [Example repository.json \(security section\)](#)
- [Access Roles Module](#)
- [Overview](#)
- [REST API](#)
- [Example Data](#)
- [Inheritance of Effective Roles](#)
- [Cascading Delete Permission](#)
- [Authorization Operations and Example](#)
  - [Order of operation:](#)
  - [Examples:](#)
- [Basic Role-based Authorization Delegate](#)
  - [Roles](#)
  - [Policy](#)
    - [Role/Permission Matrix](#)
  - [Configuring the Basic Role-Based Authorization Delegate](#)
- [XACML Authorization Delegate](#)
- [Bypassing Authorization](#)
  - [Step-by-Step:](#)
  - [Example repository.json \(security section\)](#)

## Overview

The Fedora 4 Authentication (AuthN) and Authorization (AuthZ) framework is designed to be flexible and extensible, to allow any organization to configure access to suit its needs.

The following sections explain the Fedora 4 AuthN/Z framework, and provide instructions for configuring some out-of-the-box access controls.

For clarity's sake, a distinction is made between Authentication and Authorization:

- **Authentication** answers the question "who is the person, and how do I verify that they are who they say they are?" Fedora 4 relies on the web servlet container to answer this question.
- **Authorization** answers the question, "does this person have permission to do what they want to do?". Fedora 4 provides three different ways to answer this question:
  - Simple servlet container authentication. Anyone who has authenticated through the web application container (Tomcat, Jetty, WebSphere, etc.) has permission to do everything – in effect all, authenticated users are superusers.
  - Basic Access Roles authorizations. Authenticated users are mapped onto one or more preconfigured roles; a user's role determines what they have permission to do.
  - XACML authorizations. Policies created using the XACML framework are used to determine what operations are permissible to whom, using user and resource properties exposed to the XACML engine.

## Servlet Container Configuration

Fedora 4 uses servlet container authentication (Realms) to provide minimal protection for your repository, including the set up of "superuser" accounts. User credentials are configured in your web application container, usually in a properties file or XML file. By configuring superuser accounts you can require authentication for all management (write) operations. This document describes how to set up Fedora and either Tomcat or Jetty to enable HTTP Basic Authentication, using simple user files. Consult your web application server documentation for other ways to configure and manage users; Fedora can handle any user principal passed to it by the servlet container, as provisioned by any of the container's supported authentication mechanisms.

The superuser role is **fedoraAdmin**. This is comparable to the **fedoraAdmin** superuser role in Fedora 3, used for Fedora 3 API-M operations.

- [Configure your repo.xml file](#)
- [Configure your repository.json file](#)
- [Configure your web.xml](#)

- [Configure your web application container](#)
  - [Jetty](#)
  - [Tomcat](#)

The Fedora authorization modules reside in separate source code modules from the core Fedora web-application.

- WebAC : <https://github.com/fcrepo4/fcrepo-module-auth-webac>
- RBACL : <https://github.com/fcrepo4/fcrepo-module-auth-rbac>
- XACML : <https://github.com/fcrepo4/fcrepo-module-auth-xacml>

As a result, each release includes pre-built Fedora "webapp-plus" war files that have the authorization modules included. You are recommended to use one of these "webapp-plus" war files as a starting point for having an authorization-enabled deployment.

- Webapp-plus releases: <https://github.com/fcrepo4-exts/fcrepo-webapp-plus/releases>

You can then follow the guidelines in the [Best Practices - Fedora Configuration](#) document to specify site-specific "repo.xml" and "repository.json" configurations, as further described below.

## 1. Configure your repo.xml file

Add the beans *authenticationProvider* and *fad* to your *repo.xml* file, and make the *modeshapeRepofactory* bean dependent on *authenticationProvider*. Use the class **org.fcrepo.auth.ServletContainerAuthenticationProvider** as your authentication provider. Here is an example *repo.xml* that configures authentication and authorization using the Basic Roles authorization delegate.

To specify a local *repo.xml* configuration, provide the system property as follows:

```
JAVA_OPTS="... -Dfcrepo.spring.repo.configuration=file:/local/repo.xml"
```

### repo.xml with authentication configured

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:p="http://www.springframework.org/schema/p"
  xmlns:util="http://www.springframework.org/schema/util"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context-3.0.xsd
    http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.
xsd">

  <!-- Context that supports the actual ModeShape JCR itself -->

  <context:annotation-config/>

  <bean name="modeshapeRepofactory"
    class="org.fcrepo.kernel.modeshape.spring.ModeShapeRepositoryFactoryBean"
    p:repositoryConfiguration="{fcrepo.modeshape.configuration:classpath:/config/servlet-auth
/repository.json}"
    depends-on="authenticationProvider"/>

  <bean class="org.modeshape.jcr.ModeShapeEngine" init-method="start"/>

  <bean id="connectionManager" class="org.apache.http.impl.conn.PoolingHttpClientConnectionManager" />

  <!-- Optional PrincipalProvider that will inspect the request header, "some-header", for user role
values -->
  <bean name="headerProvider" class="org.fcrepo.auth.common.HttpHeaderPrincipalProvider">
    <property name="headerName" value="some-header"/>
    <property name="separator" value=","/>
  </bean>

  <util:set id="principalProviderSet">
    <ref bean="headerProvider"/>
  </util:set>

  <bean name="fad" class="org.fcrepo.auth.webac.WebACAuthorizationDelegate"/>

  <bean name="authenticationProvider" class="org.fcrepo.auth.common.
ServletContainerAuthenticationProvider">
    <property name="fad" ref="fad"/>
    <property name="principalProviders" ref="principalProviderSet"/>
  </bean>

  <!-- For the time being, load annotation config here too -->
  <bean class="org.fcrepo.metrics.MetricsConfig"/>
</beans>
```

## 2. Configure your repository.json file

Modify the security section to enable both authenticated (via authentication provider) and internal sessions between Fedora and ModeShape.

To specify a local repository.json configuration, provide the system property as follows:

```
JAVA_OPTS="... -Dfcrepo.modeshape.configuration=file:/local/repository.json"
```

It should contain a "security" element that matches this block:

#### repository.json security

```
"security" : {
  "anonymous" : {
    "roles" : ["readonly", "readwrite", "admin"],
    "useOnFailedLogin" : false
  },
  "providers" : [
    { "classname" : "org.fcrepo.auth.common.ServletContainerAuthenticationProvider" }
  ]
},
```

### 3. Configure your web.xml

Configure your **web.xml**.

Modify `fcrepo-webapp/src/main/webapp/WEB-INF/web.xml` by uncommenting the security configuration

```
<!--Uncomment section below to enable Basic-Authentication-->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Fedora4</web-resource-name>
    <url-pattern>/*</url-pattern>
    <http-method>DELETE</http-method>
    <http-method>PUT</http-method>
    <http-method>HEAD</http-method>
    <http-method>OPTIONS</http-method>
    <http-method>PATCH</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>fedoraUser</role-name>
    <role-name>fedoraAdmin</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>fcrepo</realm-name>
</login-config>
```

The "auth-constraint" element must contain the roles defined as your users (see below for jetty and tomcat).

### 4. Configure your web application container

#### • Jetty

- Create your **jetty-users.properties** file. This file contains entries in the format *username: password [ , role, ...]*, where
  - *username* is the user's login id (the principal)
  - *password* is the user's password
  - *role* is the servlet role they are assigned upon login; jetty allows you to specify any number of roles (or no role at all). Fedora currently supports two roles: **fedoraAdmin**, which is the superuser role, and has rights to do everything; and **fedoraUser**, which is a user role, and must be granted permissions by the Policy Enforcement Point to perform actions.

Sample **jetty-users.properties** file that contains three users, two of whom are regular users, and the third of whom (fedoraAdmin) is a Fedora superuser:

#### jetty-users.properties

```
testuser: password1,fedoraUser
adminuser: password2,fedoraUser
fedoraAdmin: secret3,fedoraAdmin
```

- Configure your Jetty login realm.
  - Standalone
    - Modify your **jetty.xml** file to configure the login realm and include the jetty-users.properties file:

#### jetty.xml login service

```
<Configure class="org.eclipse.jetty.webapp.WebAppContext">

  <!-- Set this to the webapp root of your Fedora 4 repository -->
  <Set name="contextPath">/</Set>

  <!-- Set this to the path of of fcrepo4 WAR file -->
  <Set name="war"><SystemProperty name="jetty.home" default="."/>/webapps/fcrepo4</Set>

  <Get name="securityHandler">
    <Set name="loginService">
      <New class="org.eclipse.jetty.security.HashLoginService">
        <Set name="name">fcrepo4</Set>
        <!-- Set this to the path to your jetty-users.properties file -->
        <Set name="config"><SystemProperty name="jetty.home" default="."/>/path/to
        /jetty-users.properties</Set>
      </New>
    </Set>
  </Get>

</Configure>
```

- Embedded in Maven
  - The fcrepo-webapp Maven project includes jetty-maven-plugin. The property *jetty.users.file* sets the location of the **jetty-users.properties** file. Run the fcrepo-webapp server with the following system property:
 

```
-Djetty.users.file=/path/to/jetty-users.properties
```

See the [Jetty Authentication](#) documentation for more details.

## • Tomcat

- Create or edit your `$CATALINA_HOME/conf/tomcat-users.xml` file. It has entries of the form

```
<user name="principal" password="password" roles="role1, role2, ..." />
```

where:

- *name* is the user's login id (the principal)
- *password* is the user's password
- *roles* are the servlet roles they are assigned upon login; tomcat allows you to specify any number of roles (or no role at all). Fedora currently supports two roles: **fedoraAdmin**, which is the superuser role, and has rights to do everything; and **fedoraUser**, which is a user role, and must be granted permissions by the Policy Enforcement Point to perform actions.

Sample **tomcat-users.xml** file that contains three users, two of whom are regular users, and the third of whom (fedoraAdmin) is a Fedora superuser:

#### tomcat-users.xml

```
<tomcat-users>
  <role rolename="fedoraUser" />
  <role rolename="fedoraAdmin" />
  <user name="testuser" password="password1" roles="fedoraUser" />
  <user name="adminuser" password="password2" roles="fedoraUser" />
  <user name="fedoraAdmin" password="secret3" roles="fedoraAdmin" />
</tomcat-users>
```

- Configure your Tomcat login realm.
  - Modify your file `$CATALINA_HOME/conf/server.xml` file to configure the login realm with the Fedora 4 webapp context:

#### Tomcat server.xml Realm

```
<Context>
...
  <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
        resourceName="UserDatabase" />
</Context>
```

See the [Tomcat Realms](#) documentation for more details.

## Authorization Delegates

### Overview

Fedora Authorization Delegates allow you to implement one interface to enforce access control over your Fedora repository. This interface, `FedoraAuthorizationDelegate`, has callbacks that allow you to restrict ModeShape operations and filter search results. After following these configuration steps, Fedora's REST endpoints will respond with 403 response codes when the requested action is unauthorized by the authorization delegate.

Use of an authorization delegate and Fedora-specific authorization is optional. You can also configure Fedora to run without API security. You may want to only enforce container authentication or leave the service running completely unsecured, behind a firewall for instance. For details, see [How To Bypass Authorization](#).

### Fedora Administrators (fedoraAdmin user role)

The authorization delegate is not consulted when servlet credentials identify a client with the **fedoraAdmin** role. When the container has authenticated the connected client as a **fedoraAdmin**, all actions are permitted and PEP is bypassed.

### FedoraAuthorizationDelegate Implementations

There are two reference implementations available:

- [Basic Role-based Authorization Delegate](#) - An authorization delegate that operates on three fixed roles that may be assigned throughout the repository tree. (reader, writer, admin)
- [XACML Authorization Delegate](#)

You can also create an authorization delegate implementation and perform security checks differently, possibly including calls to remote services.

Two files contain the configuration options for authorization delegates:

- **repo.xml**: the global repository configuration file. Three beans enable the PEP extension:
  - *modeshapeRepoFactory*: should contain a dependency on the *authenticationProvider* bean
  - *authenticationProvider*: should specify the *ServletContainerAuthenticationProvider* class, so that the servlet container handles authentication
    - This bean should have a property "fad" that points to the *fad* bean, to enable the servlet container authentication provider to use the authorization delegate
  - *fad*: should point to your class with the authorization delegate implementation
- **repository.json**: the ModeShape configuration file. It contains a security section, where the **internal session** authentication between Fedora and the ModeShape storage layer is configured. Note that the roles configured here do not apply to end user authentication and authorization..

### Step-by-step:

1. Open the `repo.xml` file in your Fedora web application.
2. Add your authorization delegate implementation as a bean in this file and give it the ID of "fad". Your authorization delegate bean may include more specific configuration details than the example.
3. Now add the Fedora ModeShape Authentication Provider bean. (see `repo.xml` example)
4. Make sure that your `modeshapeRepoFactory` bean has the `depends-on` attribute pointing at the `authenticationProvider` (see `repo.xml` example).
5. Open your `repository.json` file.
6. Add `org.fcrepo.auth.ServletContainerAuthenticationProvider` as a provider in the security section. (see `repository.json` example)

### Example repo.xml (repository and security beans)

```
<bean name="modeshapeRepofactory" class="org.fcrepo.kernel.spring.ModeShapeRepositoryFactoryBean"
  depends-on="authenticationProvider">
  <property name="repositoryConfiguration" value="{fcrepo.modeshape.configuration:repository.json}" />
</bean>

<bean name="fad" class="your.own.implementation"/>

<bean name="authenticationProvider" class="org.fcrepo.auth.ServletContainerAuthenticationProvider">
  <property name="fad" ref="fad"/>
</bean>
```

### Example repository.json (security section)

```
"security" : {
  "anonymous" : {
    "roles" : ["readonly","readwrite","admin"],
    "useOnFailedLogin" : false
  },
  "providers" : [
    { "classname" : "org.fcrepo.auth.ServletContainerAuthenticationProvider" }
  ]
},
```

## Access Roles Module

- [Overview](#)
- [REST API](#)
- [Example Data](#)
- [Inheritance of Effective Roles](#)
- [Cascading Delete Permission](#)
- [Authorization Operations and Example](#)

## Overview

This module creates a REST API to assign new roles to identities and to query the roles already assigned on Fedora [resources](#).

In roles-based access control, users or groups are not granted specific actions on resources; rather, users and groups have roles assigned to them on resources, and these roles are mapped onto permitted actions elsewhere. This makes it much easier to manage permissions globally: rarely will masses of resources need to be updated if their permissions change. Only the role-to-permission mapping will be updated. Role-based access control is a common pattern in security, providing extensible role-specific behavior while retaining straightforward management.

This module does not define any specific roles or enforce permissions granted to roles. For roles to be effective, this module must be configured alongside an authorization delegate that is aware of roles. One roles-aware authorization delegate is provided as a reference implementation, the [Basic Roles-Based Authorization Delegate](#).

## REST API

The module adds another REST endpoint to every Fedora [resource](#) path. The URL pattern is as follows:

<path to Container>/fcr:accessroles

REST methods:

method	description
GET	Retrieves the roles assigned on a resource.
GET w/effective parameter	Retrieves the effective roles assigned on a resource, which may cascade from an ancestor role assignment.
POST	Sets all the roles assigned on a resource.
DELETE	Removes any roles assigned on a resource, such that effective roles are inherited again.

The POST and GET methods currently support a JSON structure (as Content-type *application/json*) where principals are mapped to lists of roles:

```
{
  "johndoe" : [ "reader" ],
  "janedoe" : [ "writer" ],
  "freddoe" : [ "patron", "editor" ]
}
```

This module assigns one or more roles to a string, which is the name of a security principal. ([java.security.Principal](#)) The principals used in your repository environment must have unique names. You may use whatever principals you wish, but we recommend applying the appropriate standard for your environment. This module does not validate principal names.

Fedora uses a principal named "EVERYONE" to represent the general public. This principal is added to every incoming web request. You may assign any role to the EVERYONE principal.

By default, role names are not validated, since the module does not define the set of role names that may be assigned in Fedora. However, you may configure a set of specific roles and then the API will validate roles.

## Example Data

```
root/ (default content roles, i.e. no roles for anyone)
  Container A (EVERYONE => reader; johndoe => admin)
    Binary 1 (johndoe => admin)
    Container Q (EVERYONE => reader; johndoe => admin)
      Container R (janedee => admin)
  Container B (EVERYONE => reader; johndoe => admin)
    Container T
      Container V
  Container C
```

## Inheritance of Effective Roles

Descendant resources inherit the roles assigned on ancestor resources *only if they have no roles assigned themselves*. If a resource has any roles assigned, *these assignments override ALL ancestor assignments*.

The following cases, based on the example data above, demonstrate how inheritance plays out.

1. Binary 1 of Container A only allows one principal to access the resource: *johndoe*. He will have *admin* privileges. None of the ACLs on Container A will be applied; the binary will not inherit the *EVERYONE => reader* ACL on Container A.
2. Container R, a child of Container Q, has its own content ACL: *janedee* has *admin* privileges on Container R. No one else has any access to the resource, not even the parent resource (Container Q) principals (*EVERYONE* and *johndoe*).
3. Container T, a child of Container B, has no content ACLs. So it inherits the ACLs of its most immediate ancestor with content ACLs: Container B. *EVERYONE* has reader privileges on Container T, and *johndoe* has *admin* privileges on the Container.
4. Container V also inherits the ACLs of Container B (its most immediate ancestor with content ACLs).
5. Container C has no content ACL; it inherits the ACLs of the root resource, which is to say, nothing. No one other than *fedoraAdmin* has any access to this Container.

## Cascading Delete Permission

When deleting a resource, the user must have an effective role that will allow them to delete *ALL* the descendant Containers under the resource. (binaries, child Containers, etc..) If any descendant resource cannot be deleted, then the entire delete transaction will be denied.

For example, in the graph shown above, the principal *johndoe* cannot delete container A, although he has an admin role on it and its binary; that is because he does not have an effective role on Container R, the resource's grandchild, that will permit him to delete it. If he wants to delete Container A, he will first have to ask *janedee* to delete Container R.

## Authorization Operations and Example

(Editor's note: this section would make more sense within the Basic Roles auth delegate documentation/page)

### Order of operation:

- **Container Authentication:** A user comes into the system. They are assigned a **user principal**:
  - If they authenticate through some authentication gateway, then their principal may be generated from some of the person's attributes;
  - Whether they authenticate or not, the request will always acquire an "EVERYONE" principal.
- **Fedora Principal Provider extensions:** Principal provider extensions may bring in more principals after authentication, such as groups, from sources like LDAP.
- **Fedora Roles Authorization Delegate Queries for Assigned Roles on Content: What roles have been assigned?**
  - The authorization layer queries the requested repository resource(s) for any content-assigned roles.



- If none are found locally, then it will query each ancestor in turn until role assignments are found.
- If no role assignments are found in the tree of resources, then a default set of role assignments is used. (see Container C above)
- **Fedora Roles Authorization Delegate - Role Resolution: What roles does this request have?**
  - The set of principals in the request are compared to the principals in the ACLs on the resource. The roles for each matching principal in the Container ACL are the **effective roles** for the user.
  - At this point we have the effective access roles for this operation
- **Fedora Roles Authorization Delegate - Policy Enforcement: Does this role have permission to perform the requested action?**
  - Note: The Fedora Authorization Delegate is an extension point, so enforcement will vary by the chosen implementation. We assume that installations will combine the access roles module with a roles-based authorization delegate.
  - The effective roles, assigned to the user on the content, are used to determine if the user has permission to perform the action on a given resource.
  - **Basic Roles authorization delegate** implementation does permission checks in java code:
    - Permission is determined by evaluating at a minimum the effective roles for the user on the resource in question, and the action requested.
  - In other roles-based authorization delegate implementations, more factors may also enter into the equation to determine permission.
- The authorization delegate will return a response to ModeShape, which will throw an exception to Fedora if access has been denied.
- Fedora will respond with a 403 if the given REST operation is denied.
- The one exception to this process is the **fedoraAdmin container role**. If the request has a fedoraAdmin user role (in the container), then no resource checks are made. The authorization delegate is not consulted as admins have permission to do everything. Resources will never have the fedoraAdmin role explicitly assigned to them, since it is a container role and not a content role. (e.g. a tomcat user role)

## Examples:

1. Unauthenticated user requests to see Container A.
  - a. The user is assigned the user principal "EVERYONE".
  - b. The authorization delegate intercepts the request, gets the ACLs for Container A: "EVERYONE" => "reader" and "johndoe" => "admin".
  - c. The authorization delegate compares the user principal "EVERYONE" to the principals in Container A's ACLs, and sees that "EVERYONE" matches. The effective role for this request is "reader", the role paired with the principal "EVERYONE" on the Container.
  - d. The authorization delegate sees if the role "reader" can view the Container; it can.
  - e. The authorization delegate returns "yes", and the request proceeds.
2. Unauthenticated user requests to see binary 1 on Container A.
  - a. The user is assigned the user principal "EVERYONE".
  - b. The authorization delegate intercepts the request, gets the ACLs for binary 1: "johndoe" => "admin".
  - c. The authorization delegate compares the user principal "EVERYONE" to the principals in binary 1's ACLs, but does not find a match.
  - d. The authorization delegate returns "no", and the request is denied.
3. Unauthenticated user requests to delete Container B.
  - a. The user is assigned the user principal "EVERYONE".
  - b. The authorization delegate intercepts the request, gets the ACLs for Container B: "EVERYONE" => "reader" and "johndoe" => "admin".
  - c. The authorization delegate compares the principal "EVERYONE" to the principals in Container B's ACLs, and sees that "EVERYONE" matches. The effective role for this request is "reader", the role paired with the principal "EVERYONE" on the Container.
  - d. The authorization delegate sees if the role "reader" can delete the Container; it cannot.
  - e. The authorization delegate returns "no", and the request is denied.
4. John Doe requests to update binary 1 on Container A.
  - a. The user is assigned the user principals "johndoe" and "EVERYONE".
  - b. The authorization delegate intercepts the request, gets the ACLs for binary 1: "johndoe" => "admin".
  - c. The authorization delegate compares the user principals "johndoe" and "EVERYONE" to the principals in binary's ACLs, and sees that "johndoe" matches. The effective role for this request is "admin", the role paired with the principal "johndoe" on the Container.
  - d. The authorization delegate sees if the role "admin" can update the Container; it can.
  - e. The authorization delegate returns "yes", and the request proceeds.

## Basic Role-based Authorization Delegate

This authorization delegate makes decisions based on the four basic roles of "metadata reader", "reader", "writer", and "admin". These roles are assigned to principals on Fedora [Glossary](#). Assigned roles are inherited through the repository tree until blocked by another assignment.

The role **metadata reader** has not yet been implemented.

This authorization delegate makes use of the [Access Roles Module](#) to assign and query roles in the repository.

## Roles

- **metadata reader** - can retrieve information about Fedora Containers, but cannot retrieve content
- **reader** - can retrieve information about Fedora Containers, including content
- **writer** - all permissions of reader; can create, modify and delete Fedora Containers
- **admin** - all permissions of writer; can modify the roles assigned to Fedora Containers

## Policy

The permissions granted to these roles are fixed. Rather than consulting any declarative policy, this authorization delegate has hard-coded role-permission assignments in the source code.

## Role/Permission Matrix

	metadata reader	reader	writer	admin
read properties	X	X	X	X
read content		X	X	X
write			X	X
write roles				X

## Configuring the Basic Role-Based Authorization Delegate

See [Authorization Delegates](#) for more information on how an authorization delegate is configured.

Edit your **repo.xml** file to configure the authentication provider. The file should contain these three beans, as shown:

```
<bean name="modeshapeRepofactory" class="org.fcrepo.kernel.spring.ModeShapeRepositoryFactoryBean"
  depends-on="authenticationProvider">
  <property name="repositoryConfiguration" value="{fcrepo.modeshape.configuration:repository.json}" />
</bean>

<bean name="fad" class="org.fcrepo.auth.roles.basic.BasicRolesAuthorizationDelegate"/>

<bean name="authenticationProvider" class="org.fcrepo.auth.common.ServletContainerAuthenticationProvider">
  <property name="fad" ref="fad"/>
</bean>
```

Edit your **repository.json** file to enable an authenticated internal session between Fedora and ModeShape, so that the security section matches the example shown:

```
"security" : {
  "anonymous" : {
    "roles" : ["readonly", "readwrite", "admin"],
    "useOnFailedLogin" : false
  },
  "providers" : [
    { "classname" : "org.fcrepo.auth.common.ServletContainerAuthenticationProvider" }
  ]
},
```

## XACML Authorization Delegate

Unable to render {include} The included page could not be found.

## Bypassing Authorization

Running Fedora without authorization means that the REST API is available to any request coming from the container and lacks any finer-grained security. This is useful when Fedora is running behind another application that connects to Fedora and implements its own security checks. In addition, this configuration is useful for temporary demonstrations and for running software tests that do not require security.

This configuration does not preclude the use of container authentication to secure Fedora. However, container roles are not used for any further authorization within Fedora. All requests are treated as superusers.

The security bypass for REST endpoint is accomplished by supplying an alternate ModeShape authentication provider for servlet credentials. This servlet authentication provider permits all actions at the modeshape level and does not use a PEP (Policy Enforcement Point).

### Step-by-Step:

1. If you previously configured a PEP, open your repo.xml file and remove any beans that are instances of "org.fcrepo.auth.common.ServletContainerAuthenticationProvider".
2. Also remove the PEP bean, if one was configured.
3. Remove the depends-on attribute on the modeshapeRepofactory bean, if there is one.
4. Open your repository.json file
5. Under security, configure the "BypassSecurityServletAuthenticationProvider", as shown in the example below.

### Example repository.json (security section)

```
"security" : {  
  "anonymous" : {  
    "roles" : ["readonly","readwrite","admin"],  
    "useOnFailedLogin" : false  
  },  
  "providers" : [  
    { "classname" : "org.fcrepo.auth.common.BypassSecurityServletAuthenticationProvider" }  
  ]  
},
```