

**State of Florida**

**Department of Environmental Protection**

**Portal Authentication Filter: Quick Start Guide**

**GDE-11051301.1.0**

Table of Contents

Overview 3

Implementation 3

DEP Maven Repository 3

Security Manager Dependency 3

Define the filter in the web.xml 4

Setting Public Resources 4

Greeting Users 4

Developer Properties 5

Security Checks 6

In Java Code 6

In JSP Code 6

Additional Resources 7

# Overview

The purpose of this document is to provide a quick-start tutorial on the use of the FDEP Portal Authentication Filter.

 The main steps to perform are:

1. Define the FDEP repository in your pom
2. Add the library as a dependency to the project
3. Define the filter in the web.xml
4. Add the optional developer.properties file in resources

**Important:** You must create your application in DepSec before using your application or the authentication system will fail.

Implementation

## DEP Maven Repository

The Maven Dependency for the Portal Authentication Filter requires the definition of the mtbld core maven repository and the inclusion of the correct release based upon your projects core framework. The FDEP Maven Repository is defined as:

<repository>

<id>mtbld</id>

<name>MTBLD Repository</name>

<url>[http://mtbld.dep.state.fl.us/maven/repository</url](http://mtbld.dep.state.fl.us/maven/repository%3C/url)>

<snapshots>

<enabled>false</enabled>

</snapshots>

<releases>

<enabled>true</enabled>

</releases>

</repository>

## Security Manager Dependency

This is an example of the dependency snippet for the projects POM.XML file. Please use Archiva on the FDEP Software Development Infrastructure suite to find the latest release:

<dependency>

<groupId>dep</groupId>

<artifactId>securitymanager</artifactId>

<version>2.0.10</version>

</dependency>

## Define the filter in the web.xml

Add both the filter and the requisite filter mapping to the web.xml:

<filter>

<filter-name>Authenticate</filter-name>

<filter-class>dep.otis.components.security.core.PortalAuthenticateFilter</filter-class>

<!-- sb: this is only needed if you’re using the DepSec dynamic menu feature. -->

<init-param>

<param-name>dataSource</param-name>

<param-value>jdbc/DepKickstart</param-value>

</init-param>

<!—sb: this is only needed if your application has public portions that users do not need to be logged in for. -->

<init-param>

<param-name>publicResources</param-name>

<param-value>/public</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>Authenticate</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

## Setting Public Resources

You can set up public resources in your application, which will allow visitors to access those resources without logging in. For example, to allow any request to /public, /css, or /images to go unfiltered, add the following init-param to the web.xml definition of the PortalAuthenticateFilter:

<init-param>

<param-name>publicResources</param-name>

<param-value>/public;/images;/css</param-value>

</init-param>

Resources must be semi-colon separated and can match partially against the URL. In the above example, any request that begins with /public will go unfiltered (e.g., /public/page1.jsp, /public/page2.jsp).

## Greeting Users

You can display a greeting to the currently logged in user by using the Security Manager’s greeting tag, such as:

<sm:greeting user="${sessionScope.UserInfoObject}"/>

This will produce the following output:

*Welcome, <Name of Logged In User>. You are logged on with a role of Admin.*

For user’s who are not logged on, the following will display:

*Welcome, Guest. You may register, log on, or just browse .*

This will indicate the user’s highest role (including administrator roles). If you only want to display the user’s highest application role (security value less than 200), then you can use the useHighestAppRole attribute, such as:

<sm:greeting user="${sessionScope.UserInfoObject}" useHighestAppRole="${true}"/>

## Developer Properties

The developer.properties file allows you to bypass the normal authentication process and simulate being a specific user with specific roles. Below is a sample developer.properties file. This file ***should not be checked into Subversion***, but instead should only be used locally for development on your PC.

Place the file in your classpath (typically WEB-INF/classes or src/main/resources, if you use the Maven 2 Standard Directory Layout).

# This is a sample developer.properties file. You can

# enable or disable developer mode by changing the value

# of isDeveloperMode between true and false.

isDeveloperMode=true

firstName=Steven

middleName=D.

lastName=Benitez

generation=

email=steven.benitez@dep.state.fl.us

userId=benitez\_s

samAccount=FLORIDADEP.NET$benitez\_s

secondaryUid=Benitez\_S

# Security roles should be semi-colon (;) separated. Refer

# to the Javadoc for SecurityRole.valueOf(String) for more

# information on the format of security role strings.

securityRoles=Admhjin:200;AGM Oversight:255

## Security Checks

When a user is authenticated by the PortalAuthenticateFilter, their user information is placed into session as UserInfoObject. The UserInfoObject contains details about the user and also contains a SecurityContext, which contains details about the user’s roles. Using this object, you can perform security checks in either your application code or in your UI layer.

### In Java Code

import dep.otis.components.security.AccessViolation;

import dep.otis.components.security.UserInfoObject;

import dep.otis.components.security.SecurityRole;

// getting the UserInfoObject from session

// your specific framework may have a different way of getting objects

// from session

UserInfoObject uio = (UserInfoObject)request.getSession().getAttribute(UserInfoObject.SESSION\_KEY);

if (uio.getSecurityContext().hasRole("Admin")) {

// the user is an Admin

}

if (uio.getSecurityContext().inRole("Admin", "Supervisor")) {

// the user is either an Admin or a Supervisor

// (has at least one of the specified roles)

}

// gets all of the user's roles for your application

Set<SecurityRole> allRoles = uio.getSecurityContext().getAllRoles();

if (userInfoObject.isExternalUser()) {

// only allow internal users

throw new AccessViolation("You don't have access.");

}

### In JSP Code

<%@ taglib prefix="sm" uri="http://dep.state.fl.us/otis/tags/security" %>

<c:if test="${sm:hasRole('Admin')}">

 The user has the Admin role.

</c:if>

<c:if test="${sessionScope.UserInfoObject.externalUser}">

The user is an external user.

</c:if>

# Additional Resources

For additional information about using the Security Manager API, please refer to the Javadoc located at: <http://sdi.dep.state.fl.us/dep/securitymanager/apidocs/>