



Adobe

Developing Service Providers

Adobe® Flash® Media Rights Management Server

May 2008

Version 1.0

© 2008 Adobe Systems Incorporated. All rights reserved.

Adobe® Flash® Media Rights Management Server 1.0 Developing Service Providers for Microsoft® Windows®, Linux®, and UNIX®
Edition 1.1, May 2008

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names, company logos and user names in sample material or sample forms included in this documentation and/or software are for demonstration purposes only and are not intended to refer to any actual organization or persons.

Adobe, the Adobe logo, Flash, and LiveCycle are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Java is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd.

All other trademarks are the property of their respective owners.

This product contains either BSAFE and/or TPEM software by RSA Security Inc.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)

This product includes software developed by the IronSmith Project (<http://www.ironsmith.org/>).

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

Contents

About This Document	4
Who should read this document?	4
Additional information.....	4
1 Creating Custom Authentication Providers	5
Setting up your development environment	6
Defining the external authentication provider implementation.....	6
Authenticating users and groups	7
Packaging the authentication provider	9
Deploying the authentication provider	9
Authenticating users using the authentication provider	9
2 Creating External Authorization Handlers	11
Setting up your development environment	12
Defining the external authorization handler implementation.....	12
Defining the component XML file for the authorization handler.....	15
Packaging the authorization handler	16
Deploying the authorization handler	17

About This Document

The Adobe® Flash® Media Rights Management Server SPI (Service Provider Interface) lets developers create authorization and authentication custom service providers. This document provides task-based information on how to use the Flash Media Rights Management Server SPI to create custom service providers for Flash Media Rights Management Server.

Who should read this document?

This document provides information for Java™ developers who use the SPIs to create custom authorization and authentication handlers for Flash Media Rights Management Server.

Additional information

The resources in this table provide additional information about Rights Management Server.

For information about	See
The Rights Management Server solution, development environment, run-time environment, and each Rights Management Server component	Overview
Installing, configuring, and deploying Rights Management Server	Installing and Deploying Rights Management Server for JBoss Using Turnkey Installing and Deploying Rights Management Server for JBoss Installing and Deploying Rights Management Server for WebLogic
Managing administrative users and user roles	User Management Help
Installing Flash Media Server	Adobe Flash Media Server Installation Guide
Customizing and configuring Flash Media Server	Adobe Flash Media Server Administration and Configuration Guide
The Java™ interfaces and classes used to create custom service providers	Adobe Flash Media Rights Management Server API Reference (Javadoc)
Securing video content and playlists by using the Rights Management Server command line tools	Securing Video Content and Playlists
Delivering content in Adobe Media Player	Adobe Media Player Content Developer Kit
Using Adobe Media Player to find and view content	Adobe Media Player Help

1

Creating Custom Authentication Providers

You can create custom authentication providers for Flash Media Rights Management Server to use when authenticating users. This chapter explains how to use the Flash Media Rights Management Server SPI to develop custom authentication providers that you can integrate with Flash Media Rights Management Server.

Typically, authentication occurs as described in the following sequence:

1. A user enters their user name and password into Adobe Media Player to access specific content.
2. Flash Media Rights Management Server sends the user name and password to the authentication provider.
3. The authentication provider connects to the user store and authenticates the user.
4. The authentication provider returns the results to Flash Media Rights Management Server.
5. Flash Media Rights Management Server either lets the user log in or denies the user access to the service.
6. After the user is authenticated, Flash Media Rights Management Server uses an External Authorization Handler to determine whether the user is permitted to access specific content. (See [Creating External Authorization Handlers](#).)

By using the Flash Media Rights Management Server SPI, you can create a custom authentication provider and then configure Flash Media Rights Management Server to use the custom authentication provider to supplement its default authentication provider.

Summary of steps

To develop a custom authentication provider, perform the following steps:

1. Set up your development environment.
2. Define the authentication provider implementation.
3. Define the component XML file.
4. Package the authentication provider into a JAR file.
5. Deploy the authentication provider.
6. Test the authentication provider.

Sample files

This section creates a Java class that corresponds to the *ReversingAuthenticator.java* file located at *[install directory]\Adobe\FMRMS1.0\sdk\samples*, where *[install directory]* is the Flash Media Rights Management Server installation location. As you read through this section, it is recommended that you also refer to this JAVA file.

Setting up your development environment

The first step to create an external authentication provider is to set up your development environment by creating a Java project, such as an Eclipse project. The Flash Media Rights Management Server SPI requires a JAR file named *um-spi.jar*. That is, you must set this file in your project's class path. If you do not reference this JAR file, you cannot use the Flash Media Rights Management Server SPI in your Java project.

The Flash Media Rights Management Server SPI also requires a JAR file named *fmrms-spi.jar*. Again, you must set this file in your project's class path. The following table lists the installation location of JAR files that are installed.

File	Description	Location
um-spi.jar	A required file.	<i>[install directory]\Adobe\LiveCycle8\LiveCycle_ES_SDK spi</i>
fmrms-spi.jar	A required file.	<i>[install directory]\Adobe\FMRMS1.0\sdk spi</i>
adobe-livecycle-client.jar	Required to use the Java code in <i>ReversingAuthenticator.java</i>	<i>[install directory]\Adobe\LiveCycle8\LiveCycle_ES_SDK\client-libs\common</i>

Defining the external authentication provider implementation

To develop a custom authentication provider, create a Java class that implements the `com.adobe.fmrms.spi.authentication.ExternalAuthenticator` interface.

Note: The `com.adobe.drm.authentication` package is deprecated and should not be used.

You must implement these methods, which are invoked by Flash Media Rights Management Server when it performs authentication:

- `authenticate()`
- `getDomainName()`

Authenticating users and groups

To create an implementation that allows for the authentication of users, implement the `authenticate` method in `com.adobe.fmrms.spi.authentication.ExternalAuthenticator`, which expects the following parameter values:

`username`: A `java.lang.String` that contains the user's name

`password`: A `java.lang.String` that contains the user's password

`Map`: A `java.util.Map` object that contains information such as the IP address of the client application.

The `authenticate` method returns a boolean value that specifies whether the user was authenticated.

The following file shows the class implementing the `ExternalAuthenticator` interface. This sample code requires that the password be the reverse of the user name.

Example: Defining the authentication provider implementation

```
package com.adobe.fmrms.spi.samples.authentication.reverse;

import com.adobe.fmrms.spi.authentication.ExternalAuthenticator;
import com.adobe.logging.AdobeLogger;
import java.util.Map;

public class ReversingAuthenticator extends ExternalAuthenticator
{
    private static AdobeLogger logger =
AdobeLogger.getAdobeLogger(ReversingAuthenticator.class);

    protected boolean authenticate (String userName, String password, Map
additionalCredentials)
    {
        if (userName == null || "".equals(userName) || password == null ||
"".equals(password)) {
            return false;
        }

        logger.info("Checking credentials for " + userName + " (IP=" +
additionalCredentials.get(CREDENTIAL_CLIENT_IP) + ")");
        if (userName.length() == password.length()) {
            boolean valid = true;
            for (int i = 0; valid && i < userName.length(); i++) {
                valid &= (userName.charAt(i) == password.charAt(password.length() -
i - 1));
            }
            return valid;
        }
        return false;
    }

    protected String getDomainName ()
    {
        return "Ext_Auth_Reverse";
    }
}
```

Defining the component XML file for the authentication provider

You must create a component XML file to deploy an authentication provider. A component XML file exists for each component (an authentication provider is a component) and provides metadata about the component. For information about the component XML file, see "Component XML Elements" in [LiveCycle SDK Help](#).

Note: The signature for the `authenticate` method that is exposed in the service is not the same as the `authenticate` method in the `ReversingAuthenticator` class. The `component.xml` file refers to the `authenticate` method in the following base class:

```
com.adobe.fmrms.spi.authentication.ExternalAuthenticator.
```

The following `component.xml` file is used for the authentication provider. Notice that the service name is `ReversingAuthenticator`, and the operations that this service exposes are named `authenticate`, `getConfigName`, and `getDomainForUsers`.

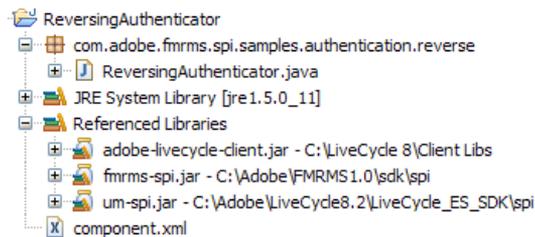
Example: Defining the component XML file for the authentication provider

```
<component xmlns="http://adobe.com/idp/dsc/component/document">
<component-id>com.adobe.fmrms.spi.samples.authentication.reverse.ReversingAuthenticator</component-id>
  <version>1.0</version>
  <class-path>fmrms-spi.jar</class-path>
  <services>
    <service name="ReversingAuthenticator">
<implementation-class>com.adobe.fmrms.spi.samples.authentication.reverse.ReversingAuthenticator</implementation-class>
      <specifications>
        <specification
spec-id="com.adobe.idp.um.spi.authentication.AuthProvider"/>
        <specification
spec-id="com.adobe.fmrms.spi.authentication.ExternalAuthenticator"/>
      </specifications>
      <auto-deploy category-id="External Authenticator"
service-id="ReversingAuthenticator" major-version="1" minor-version="0" />
      <operations>
        <operation name="authenticate" method="authenticate">
          <input-parameter name="credential" type="java.util.Map" />
          <input-parameter name="authConfigs" type="java.util.List" />
          <output-parameter name="echoed-value"
type="com.adobe.idp.um.spi.authentication.AuthResponse" />
        </operation>
        <operation name="getConfigName" method="getConfigName">
          <output-parameter name="echoed-value" type="java.lang.String" />
        </operation>
        <operation name="getDomainForUsers" method="getDomainForUsers" >
          <output-parameter name="echoed-value" type="java.lang.String"/>
        </operation>
      </operations>
    </service>
  </services>
</component>
```

Packaging the authentication provider

To deploy the authentication provider to LiveCycle ES, first package your Eclipse project into a JAR file. Ensure that external JAR files on which the authentication provider's business logic depends are included. Also, the component XML file must be present. The component.xml file and external JAR files must be located at the root of the JAR file.

The following illustration shows the Eclipse project's content, which is packaged into the external authentication provider's JAR file.



Package the authentication provider into a JAR file named `ReversingAuthenticator.jar`. In the previous diagram, notice that JAVA files are listed. After JAVA files are packaged into a JAR file, they are replaced with the corresponding CLASS files. Without the CLASS files, the authentication provider will not work.

Deploying the authentication provider

You must deploy the authentication provider by using the External Rights Services Installer tool. For information about using this tool, see [Deploying the authorization handler](#).

Authenticating users using the authentication provider

After you deploy the authentication provider, you can use it to authenticate users.

► **To use the authentication provider:**

1. Restart the application server.
2. Start LiveCycle Administration Console.
3. On the Home page, click **Settings > User Management > Domain Management > New Domain** to create a domain. Users are added to this domain when they successfully authenticate. The domain identifier must match the domain name that is specified in the authenticator's `getDomainName` method (for the `ReversingAuthenticator` provider, the name must be `Ext_Auth_Reverse`).
4. Click **Home > Settings > User Management > Domain Management > New Enterprise Domain**.
5. Provide the name and identifier of the domain.
6. Click **Add Authentication**. You are prompted to select an authentication provider.

7. Select **Custom**. After deploying the example by using the External Rights Services Installer tool, `ReversingAuthenticator` is displayed as an available provider.
8. Click **OK**. You can now save the domain.
9. To test the provider, package a FLV file and use the Adobe Flash Player to authenticate.
10. Create a policy that uses this authenticator.
11. Package a FLV file that references this policy.
12. Create a playlist that references this FLV file.
13. Play the FLV file using Adobe Media Player. You should be prompted to authenticate. Specify a user name and a password as the reverse of the user name.

2

Creating External Authorization Handlers

You can create external authorization handlers for Flash Media Rights Management Server. External authorization handlers provide centralized access control for FLV files in your organization. FLV file access can be controlled by the same control mechanism that your order management system uses. Flash Media Rights Management Server controls access to policy-protected FLV files by evaluating the policy when a user attempts to access a policy-protected file. For example, you can create an external authorization handler that grants a user a voucher for the FLV file only if your order management system indicates that the user paid for access to the content.

As part of the policy evaluation process, Flash Media Rights Management Server requires both the identifier value of the principal who is requesting access to the policy-protected FLV file and the identifier of the policy-protected FLV file. After Flash Media Rights Management Server receives these values, it generates a set of permissions. When an external authorization handler is registered with Flash Media Rights Management Server, you can enhance the policy evaluation process by restricting access to certain users or limiting the amount of time for which a user is granted access.

After an external authorization handler determines whether access to the content is permitted, it returns an optional expiration date to Flash Media Rights Management Server. If the policy has not expired, Flash Media Rights Management Server creates a voucher that specifies permissions that control access to a policy-protected FLV file. The voucher is returned to the client where it is stored if voucher-caching is enabled. If the voucher that is stored on the client expires, the client requests a new one from the server the next time the user tries to play the FLV file. If the policy on the server expires, a voucher is not issued to the client.

Summary of steps

To develop an external authorization handler, perform the following steps:

1. Set up your development environment.
2. Define the external authorization handler implementation.
3. Define the component XML file.
4. Package the authorization handler into a JAR file.
5. Deploy the external authorization handler.
6. Test the external authorization handler.

Sample files

This section creates a Java class that corresponds to the `FileAuthorizer.java` file located in `[install directory]\Adobe\FMRMS1.0\sdk\samples`, where `[install directory]` is the Flash Media Rights Management Server installation location. As you are reading this section, it is recommended that you also refer to the `FileAuthorizer.java` file.

Setting up your development environment

The first step to create an external authorization handler is to set up your development environment by creating a Java project, such as an Eclipse project. The Flash Media Rights Management Server SPI requires a JAR file named `edc-server-spi.jar`. That is, you must set this file in your project's class path. If you do not reference this JAR file, you cannot use the Flash Media Rights Management Server SPI in your Java project.

The Flash Media Rights Management Server SPI also requires a JAR file named `fmrms-spi.jar`. Again, you must set this file in your project's class path. This JAR file is installed along with the Flash Media Rights Management Server SDK. This table lists the installation location of JAR files that are installed.

File	Description	Location
<code>edc-server-spi.jar</code>	A required JAR file	<code>[install directory]\Adobe\LiveCycle8\LiveCycle_ES_SDK\spi</code>
<code>fmrms-spi.jar</code>	A required JAR file	<code>[install directory]\Adobe\FMRMS1.0\sdk\spi</code>
<code>adobe-livecycle-client.jar</code>	Required to use the Java code in <code>FileAuthorizer.java</code>	<code>[install directory]\Adobe\LiveCycle8\LiveCycle_ES_SDK\client-libs\common</code>

Defining the external authorization handler implementation

To develop an external authorization handler, create a Java class that extends the `com.adobe.fmrms.spi.authorization.ExternalAuthorization` base class. This class contains a method named `isUserAuthorized`, which Flash Media Rights Management Server invokes when a client application, such as Adobe Media Player, attempts to open a policy-protected FLV file.

Note: The `com.adobe.drm.authorization` package is deprecated and should not be used.

When creating an external authorization handler, you must implement the `isUserAuthorized` and `isAnonymousAuthorized` methods. You can create application logic within these methods to meet your business requirements. The `isUserAuthorized` method accepts the following three parameters:

`userId`: Identifier of the user requesting access.

`contentId`: Identifier of the content to which the user is requesting access. The `content id` value is set at packaging time using the `-i` option.

`externalAuthProps`: A `java.util.Map` that contains names and values of properties that are specified in the `getAuthorizerProperties` method (the values of these properties can be specified at the time the policy is created by using the `-zname` value option). This parameter may also contain additional information, such as the client's IP address.

Note: You can also use the `isAnonymousAuthorized` method. This method is similar to the `isUserAuthorized` method; however, there is no user identifier value. This method can be used to change the expiration time for a FLV file that has anonymous access, without modifying the policy (and therefore affecting all content that is protected by that policy).

The `isUserAuthorized` method returns an `AuthorizationResult` object, which indicates whether the user is authorized to view the content and, optionally, the number of days for which access is granted. If an expiration is specified and it is sooner than the expiration defined by the policy, the expiration set in the external authorization handler is used. However, the expiration cannot be extended past the date that the policy allows.

Example: Defining the external authorization handler implementation

The following external authorization handler implementation uses a text file to determine whether a user can access a FLV file. A text file lists the users who can access each piece of content. For example, if the file contains these lines, users 1, 2, and 3 can access the FLV file with the content identifier value of `contentId1`:

```
contentId1=user1, user2, user3
contentId2=user1
```

Only user 1 can access the FLV file with the content identifier value of `contentId2`. In this example, an expiration time is not set; therefore, the voucher expires based on the information specified in the policy.

The `contentId` value can be assigned to the FLV file at the time the FLV file is encrypted using Media Packager. The location of the text file is specified by the `Order_File` property. Because this property name is in the list that is returned by the `getAuthorizerProperties` method, a value for the property must be specified when the policy is created using the `-zname` value option.

Note: These authorizer properties appear in the policy that is included in the FLV file content and in the voucher. Therefore, it is not recommended that you use authorizer properties for sensitive information, such as internal server locations or passwords.

```
package com.adobe.fmrms.spi.samples.authorization.file;

import com.adobe.fmrms.spi.authorization.AuthorizationResult;
import com.adobe.fmrms.spi.authorization.ExternalAuthorization;
import com.adobe.edc.server.spi.authorization.ExternalAuthPropertyDTO;
import com.adobe.logging.AdobeLogger;
import java.io.FileInputStream;
import java.util.*;
import java.util.logging.Level;

public class FileAuthorizer extends ExternalAuthorization
{
    private static AdobeLogger logger =
        AdobeLogger.getAdobeLogger(FileAuthorizer.class);

    private static final String PROP_ORDER_LOCATION = "Order_File";

    protected ExternalAuthPropertyDTO[] getAuthorizerProperties ()
    {
        ExternalAuthPropertyDTO prop = new
        ExternalAuthPropertyDTO (PROP_ORDER_LOCATION, null);
        ExternalAuthPropertyDTO[] props = {prop};
        return props;
    }

    protected AuthorizationResult isUserAuthorized (String userId, String
    contentId, Map externalAuthProps)
    {
        logger.info("Checking authorization for " + userId + " to " + contentId +
            " (IP=" + externalAuthProps.get (PROPS_CLIENT_IP) + ")");
        Properties orders = getAccessInfo(externalAuthProps);
        if (orders != null) {
```

```
        List allowedUsers = parseList(orders.getProperty(contentId));
        if (allowedUsers.contains(userId)) {
            return new AuthorizationResult(true);
        }
    }
    return new AuthorizationResult(false);
}

protected AuthorizationResult isAnonymousAuthorized (String contentId, Map
externalAuthProps)
{
    logger.info("Checking authorization for anonymous to " + contentId +
        " (IP=" + externalAuthProps.get(PROPS_CLIENT_IP) + ")");
    Properties orders = getAccessInfo(externalAuthProps);
    if (orders != null) {
        if (orders.containsKey(contentId)) {
            return new AuthorizationResult(true);
        }
    }
    return new AuthorizationResult(false);
}

private Properties getAccessInfo(Map externalAuthProps) {
    List prop = (List) externalAuthProps.get(PROP_ORDER_LOCATION);
    if (prop != null && prop.size() > 0) {
        String orderfile = (String) prop.get(0);
        Properties orders = new Properties();
        try {
            orders.load(new FileInputStream(orderfile));
        } catch (Exception e) {
            logger.log(Level.SEVERE, "Error loading authorization file: ", e);
            return null;
        }
        return orders;
    }
    return null;
}

private List parseList (String strList)
{
    ArrayList contents = new ArrayList();
    if (strList == null)
        return contents;
    StringTokenizer stok = new StringTokenizer(strList, ",");
    while (stok.hasMoreTokens()) {
        String item = stok.nextToken();
        if (item != null) {
            contents.add(item);
        }
    }
    return contents;
}
}
```

Note: This Java class is saved as a JAVA file named *FileAuthorizer.java*.

Defining the component XML file for the authorization handler

You must create a component XML file to deploy an external authorization handler. A component XML file exists for each component (an external authorization handler is a component) and provides meta data about the component. The component XML file that is used for the external authorization handler component contains the following XML elements:

component-id: Specifies a unique identifier for the component.

version: Specifies the component version.

class-path: Specifies JAR files that are required by the component. For JAR files to be used by the component, they must be specified within this element.

services: Specifies the services that are part of this component. This element can contain one or many service elements. For a single service component, specify one service element.

service: Specifies the name of the service.

implementation-class: Specifies the name of the implementation class for the service.

operations: Specifies the operations that are part of this service. This element can contain one or many operation elements.

operation: Specifies the operation name.

input-parameter: Specifies the name and type of the input parameter for this specified operation. An input parameter element must exist for each parameter that corresponds to the operation. Also, each parameter's data type must be specified using the type attribute.

output-parameter: Represents a single output parameter from an operation, of which there can be many. The name of an output parameter must be unique to other output parameters that make up the same operation signature.

The following component.xml file is used for the external authorization handler. Notice that the service name is *FileAuthorizer* and the operation this service exposes is named *evaluate*. (The *evaluate* method is defined in the `com.adobe.fmrms.spi.authorization.ExternalAuthorization` base class and therefore is not in the code for the *FileAuthorizer* class.) The input parameter is `com.adobe.edc.server.spi.authorization.ExternalAuthDTO`, and the output value is `com.adobe.edc.server.spi.authorization.ExternalAuthResultDTO`.

Example: Defining the component XML file for the authorization handler

```
<component xmlns="http://adobe.com/idp/dsc/component/document">
  <component-id>com.adobe.drm.authorization.impl.file.FileAuthorizer</component-id>
  <version>1.0</version>
  <class-path>fmrms-spi.jar</class-path>
  <dynamic-import-packages>
    <package>com.adobe.livecycle.rightsmanagement.client.*</package>
  </dynamic-import-packages>
  <services>
    <service name="FileAuthorizer">
      <specifications>
        <specification
spec-id="com.adobe.edc.server.spi.authorization.ExternalAuthorizer"/>
      </specifications>
    </service>
  </services>
</component>
```

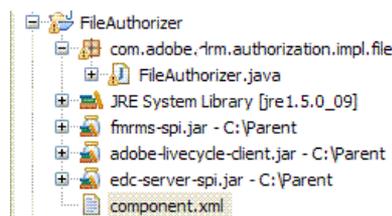
```
</specifications>
<specification-version>1.0</specification-version>

<implementation-class>com.adobe.drm.authorization.impl.file.FileAuthorizer</
implementation-class>
  <auto-deploy category-id="External Authorization"
service-id="FileAuthorizer" major-version="1" minor-version="0"/>
  <operations>
    <operation name="evaluate">
      <input-parameter name="input"
type="com.adobe.edc.server.spi.authorization.ExternalAuthDTO"
required="true"/>
      <output-parameter name="result"
type="com.adobe.edc.server.spi.authorization.ExternalAuthResultDTO"/>
    </operation>
    <operation name="getProviderProperties">
      <output-parameter name="result"
type="com.adobe.edc.server.spi.authorization.ExternalAuthPropertyDTO"/>
    </operation>
  </operations>
</service>
</services>
</component>
```

Packaging the authorization handler

To deploy the external authorization handler, package your Java project into a JAR file. Also, the component XML file must be present. The component.xml file and fmrms-spi.jar file must be located at the root of the JAR file.

The following illustration shows the Java project's content, which is packaged into the external authorization handler's JAR file.



In the above illustration, notice that the FileAuthorizer JAVA file is listed. After JAVA files are packaged into a JAR file, they are replaced with the corresponding CLASS files. Without the CLASS files, the authorization handler will not work.

Note: If the authorization handler relies on other external JAR files, those JAR files should be included at the root of the external authorization JAR file and listed in the class-path element in the component.xml file.

Deploying the authorization handler

When the external authorization handler is packaged into a JAR file, you can deploy it by using the External Rights Services Installer tool. This tool consists of a JAR file named `AdobeDeployer.jar` and is stored in `[install directory]\Adobe\FMRMS1.0\fmrms_tools\libs`

The following options are available when running the External Rights Services Installer tool:

list: A list of component identifier values and versions that implement the External Authorization SPI.

install path-to-jar: The name and location of the JAR file that represents the external authorization handler.

uninstall component-id component-version: Stop and uninstall the component with the specified identifier value and version.

-c configfile: The name and location of the configuration file. If this option is not specified, the External Rights Services Installer tool looks for the `fmrms_tools.properties` file in the working directory. The configuration file specifies the following properties:

`policyServer.server:` The location of the Flash Media Rights Management Server (for example, `http://localhost:8080`).

`policyServer.username:` The Flash Media Rights Management Server user name that is used to deploy the external authorization handler. This user must be assigned the Application Administrator role.

`policyServer.password:` The corresponding password value.

The following example shows the syntax that is required to deploy an external authorization handler named `FileAuthorizer.jar` that is located in `C:\Adobe`. Run this command from `[install directory]\Adobe\FMRMS1.0\fmrms_tools`:

```
java -jar libs\AdobeDeployer.jar install C:\Adobe\FileAuthorizer.jar
```

Note: After you deploy the external authorization handler, you can create a policy that uses it. (See "Creating and updating policies" in *Securing Video Content and Playlists*.)

Caution: You must uninstall a previous external authorization component before you deploy a new one. For example, if you are installing version 2.0 of a component, ensure that you uninstall 1.0 first.