

Customizing authentications in Documentum REST Services

Documentum REST Services in release 7.3 brings significant enhancements for authentications. **SAML 2.0 Web SSO** is now supported by the REST API. It is also a new authentication plugin type for Documentum Content Server. Documentum REST Services SAML 2.0 Web SSO can work with any SAML2 certificated Identity Providers. **Pre-authenticated** is another authentication scheme new to REST 7.3 users. With pre-authenticated authentication scheme, Documentum REST Services can be deployed within a sub trusted system, like **IBM Tivoli WebSEAL**. Besides, release 7.3 allows users to customize the authentication for the REST API including intercepting generic servlet filters and creating new authentication schemes.

In this post, we will talk about the authentication customization for Documentum REST Services. The contents of this document include:

- Customizing generic servlet filters
- Implementing anonymous access resources
- Creating custom authentications
- Exploring an OAuth2 authentication example

Customizing generic servlet filters

We noted that some REST customers have deep knowledge on web development and Spring frameworks. With the Documentum REST extensibility, they want to inject additional HTTP servlet filters to handle the requests with additional business logic, such as counting requests, logging specific requests, and more.

In Documentum REST Services release 7.3, we expose an interface for users to register custom servlet filters prior to or after the REST authentication. The extension point is in file **dctm-rest.war\WEB-INF\classes\META-INF\spring\extension\rest-api-custom-filters.xml**.

```
<beans xmlns="http://www.springframework.org/schema/beans"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
```

One very simple example is to count coming requests and authenticated requests by adding two filters before and after the authentication. Of course, you can do much more request and response interceptions with this extension point. Documentum REST SDK provides a sample project for the filter customization.



dctm-rest-sdk-7.3

+ ant-kit

+ java-api-docs

+ lib

+ maven-kit

+ rest-api-docs

- samples

Implementing anonymous access resources

+ advance-extension-sample

Documentum REST Services in release 7.3 has strengthened the web security in a lot of areas, seeing [Protecting transfer data in Documentum REST Services](#). One notable change is for any un-addressable request URIs, the server will return status **403 Forbidden**. But we also note there could be valid requirements on bypassing a resource URI. For example,

- Design a read-only & anonymous access resource for internal usage
- Add static resources such as CSS and image files in a custom REST project

- custom-servlet-filters-sample

- src

+ main

Therefore, release 7.3 brings new Java library and configurations for users to implement anonymous access resources. There are two approaches to implement anonymous access resources, **by configuration** or **by coding**.

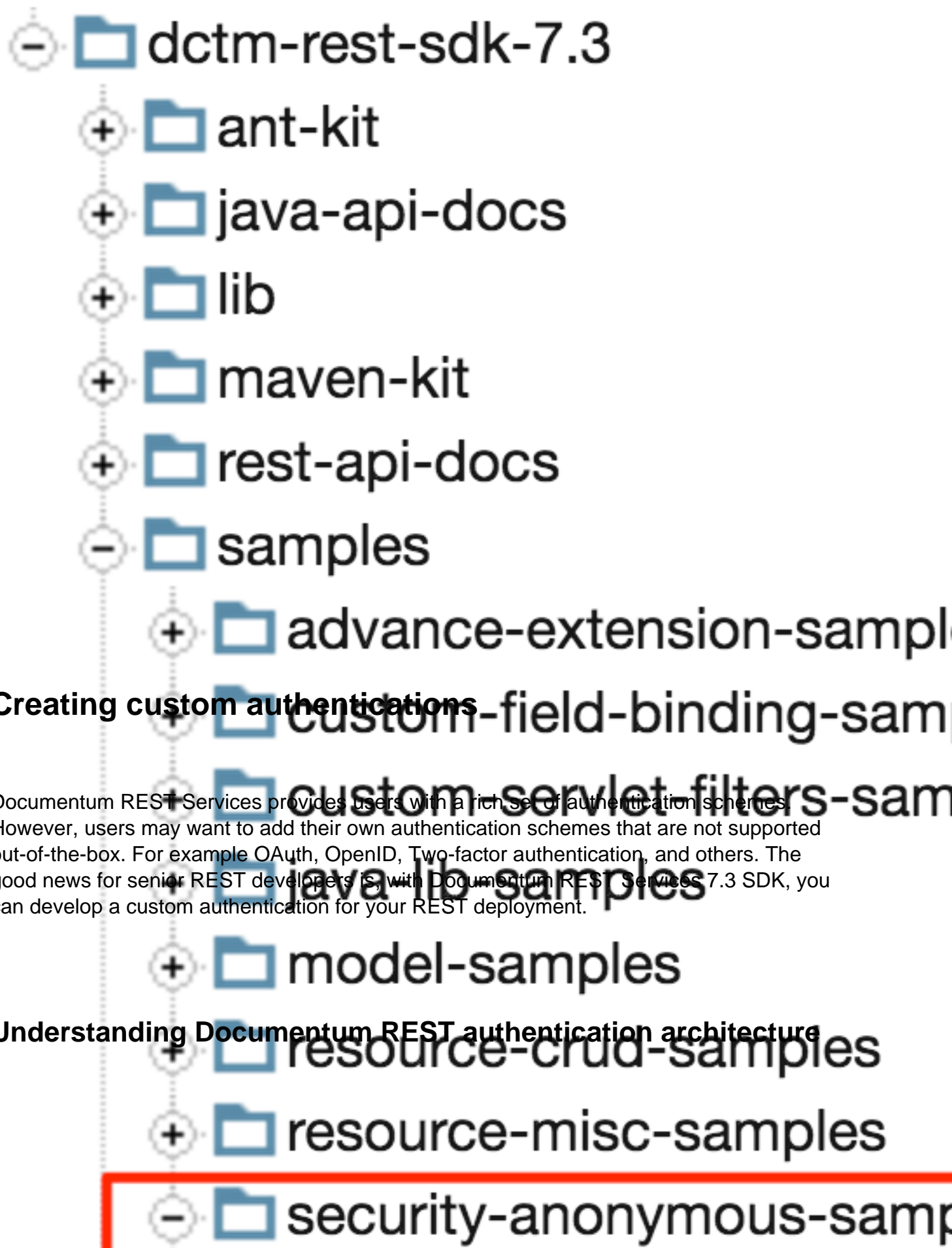
For static resource files that are added to the REST application, it is recommended to configure the runtime properties file `dctm-rest.war\WEB-INF\classes\rest-api-runtime.properties`, by adding a new entry.

```
rest.security.anonymous.url.patterns = /static/images/favicon.ico,/static/css/acme-main.css
```

For new REST services that are designed for anonymous access, it is recommended to annotate the resource controller Java class with annotation **@com.emc.documentum.rest.security.AnonymousAccess**.

```
@AnonymousAccess( contextInitializerClass = MyRepositoryContextInitializer.class, contextCleanerClass = DefaultRepositoryCo
```

Documentum REST SDK provides a sample project for anonymous access resources.

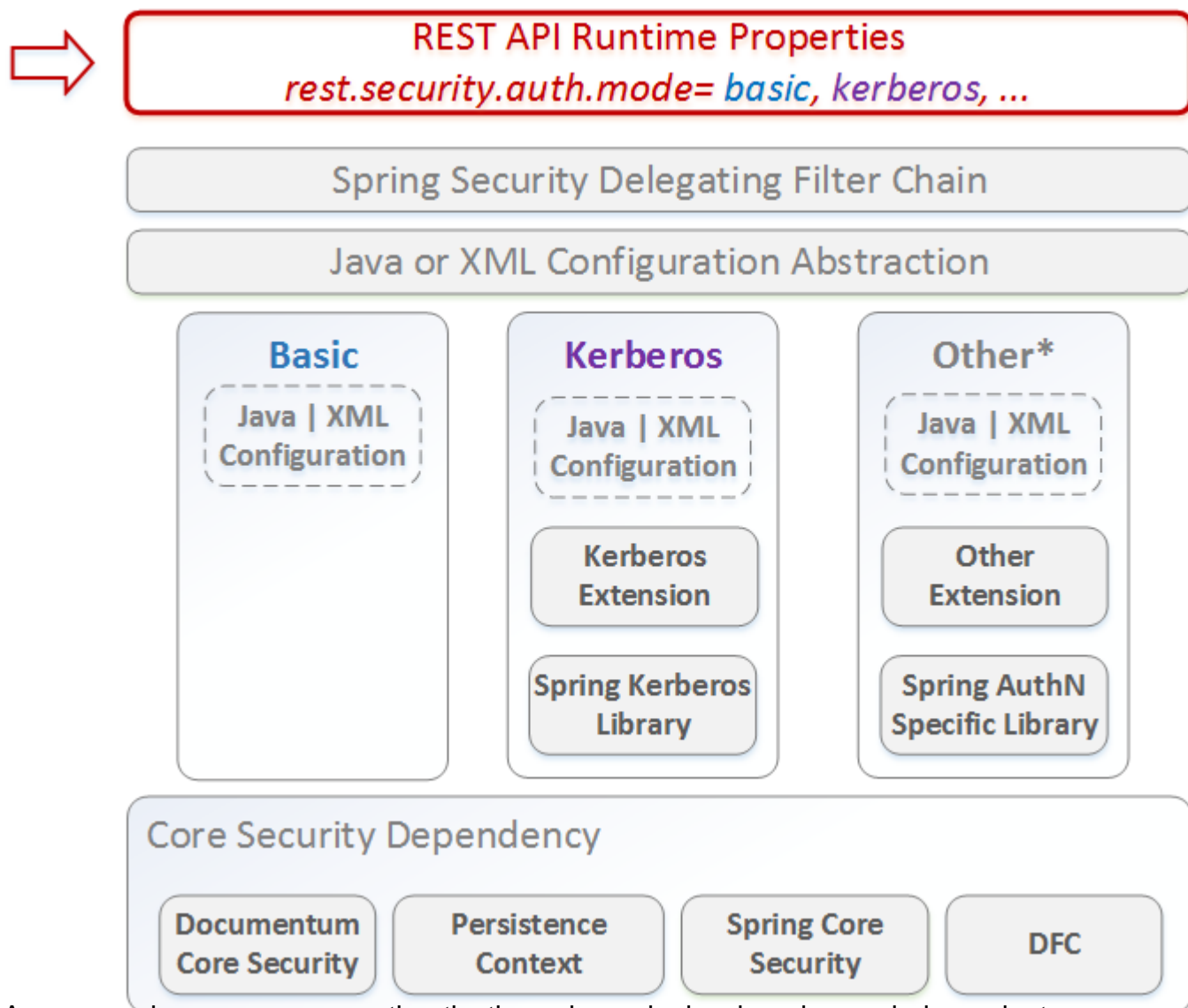


Creating custom authentications

Documentum REST Services provides users with a rich set of authentication schemes. However, users may want to add their own authentication schemes that are not supported out-of-the-box. For example OAuth, OpenID, Two-factor authentication, and others. The good news for senior REST developers is, with Documentum REST Services 7.3 SDK, you can develop a custom authentication for your REST deployment.

Understanding Documentum REST authentication architecture

Documentum REST Services has a layered and extensible authentication architecture, as illustrated in the diagram shown.



As you can image, any new authentication scheme is developed as an independent module (library) in Documentum REST Services. At the runtime, we use runtime properties configuration parameters to control which authentication scheme to be effective for the running REST application. Underlying, Documentum REST SDK provides Java libraries for developers to implement a new authentication module and login to Documentum Content Server.

The custom authentication development can be divided into two major steps:

- Developing a custom authentication module (jar)

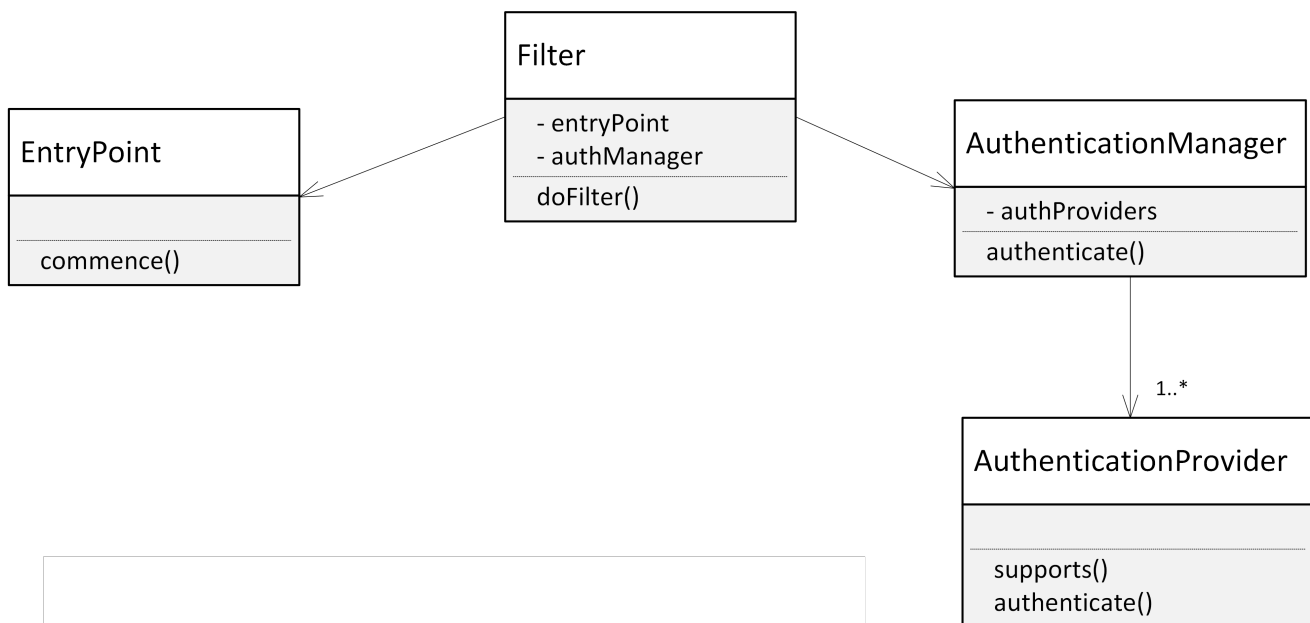
- Applying it to the REST deployment (war)

Developing a custom authentication module

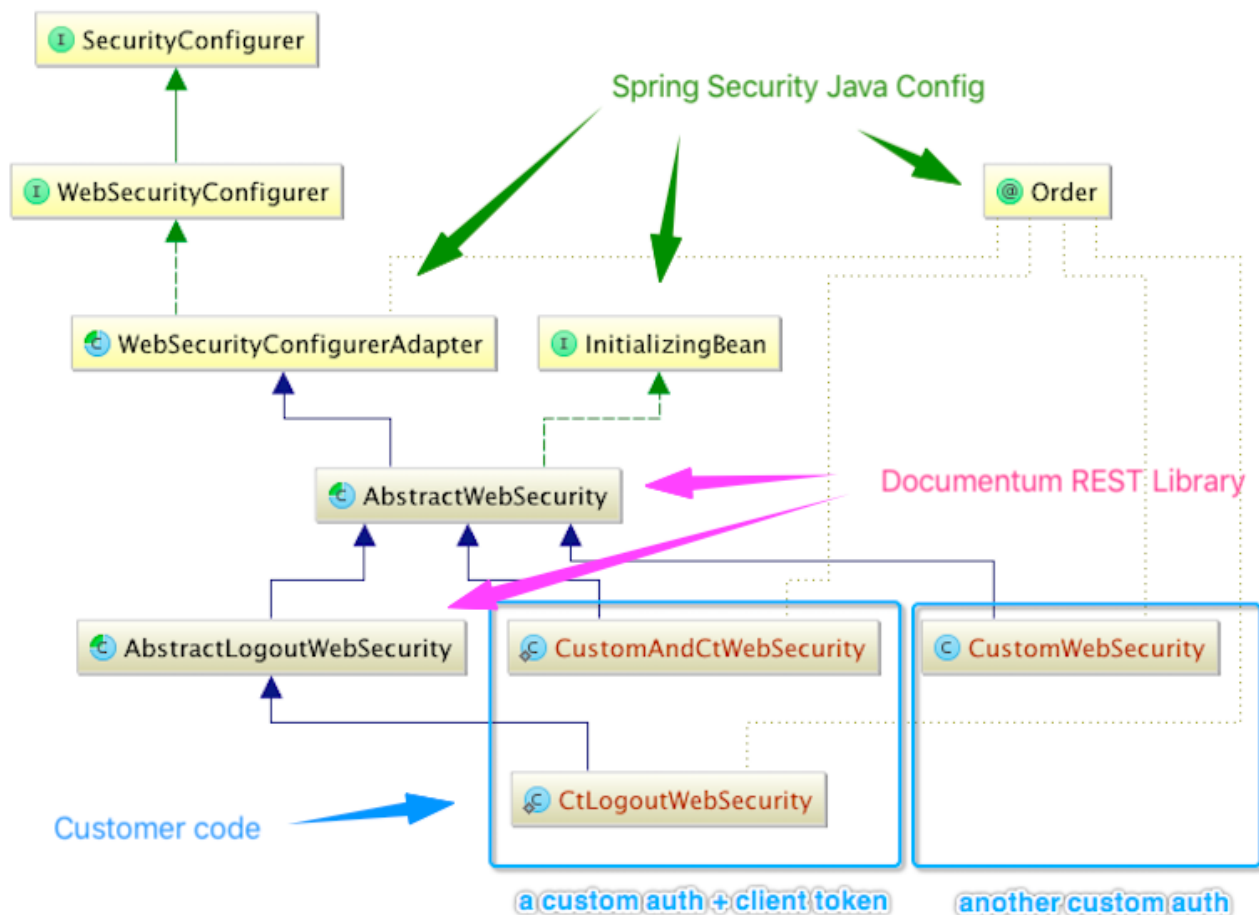
The goal of developing a custom authentication module is to build a **jar library** for the specific authentication type, and later package it to the REST war file for its usage in the REST application.

The simplest way to start with is to create a new Maven project from Documentum REST SDK maven-kit. Please refer to [Documentum REST Extensibility Tutorial \(2\): Create A Sample Project](#).

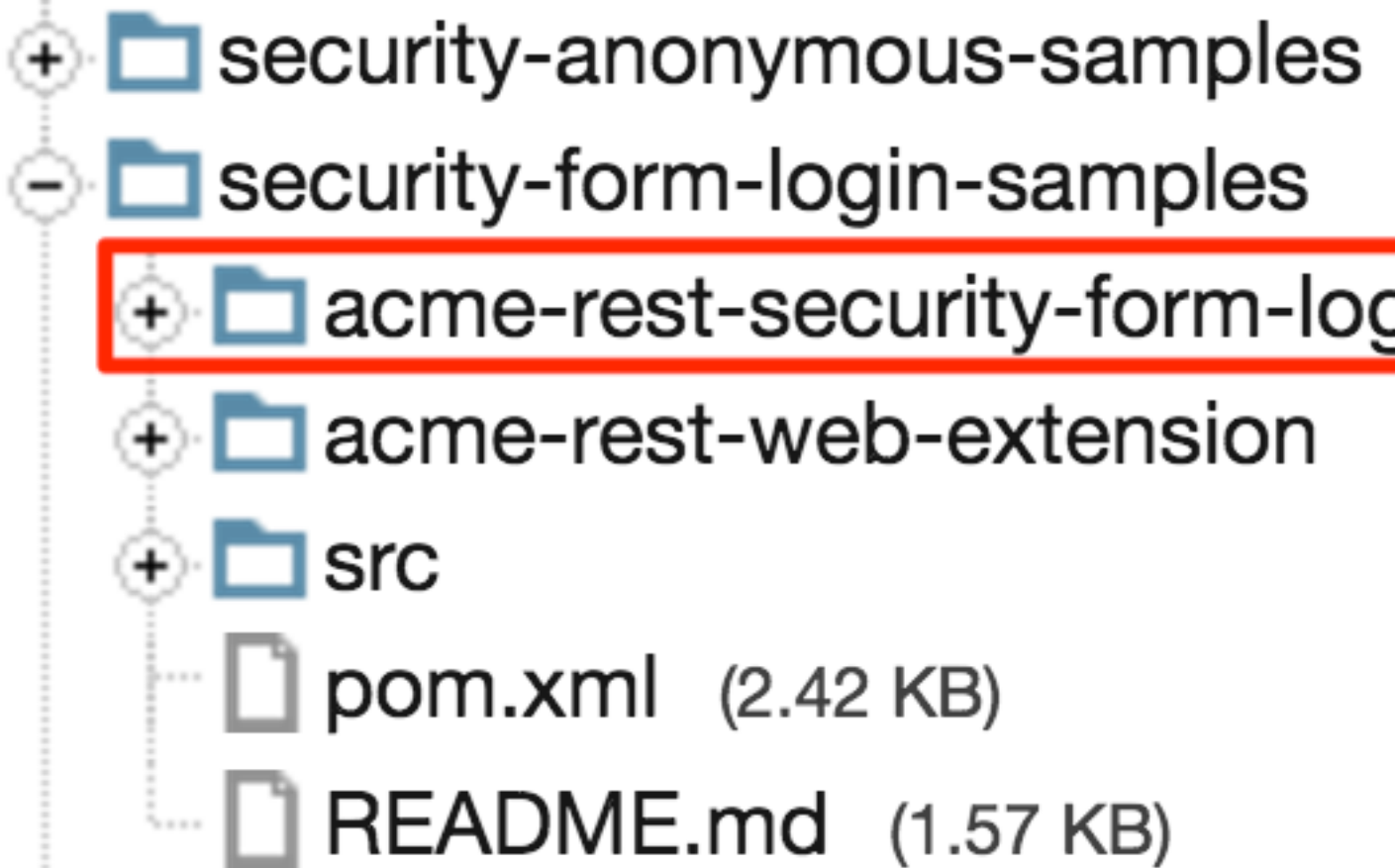
Assumed you have installed Documentum REST SDK 7.3 Maven artifacts, and created a new project. The next step is to create a new **Maven sub module** for your authentication type. As Documentum REST authentication is leveraging Spring Security to provide authentication functions, the basic requirement for a new authentication scheme is to implement below interfaces.



Documentum REST SDK provides further libraries to configure the implementations as Spring Java beans, by extending from class **com.emc.documentum.rest.context.jc.AbstractWebSecurity**.



To make it easy to start, Documentum REST SDK provides two sample projects for custom authentication development: **security-form-login-samples** and **security-oauth2-samples**. From them, you can find the sample modules for custom authentications.

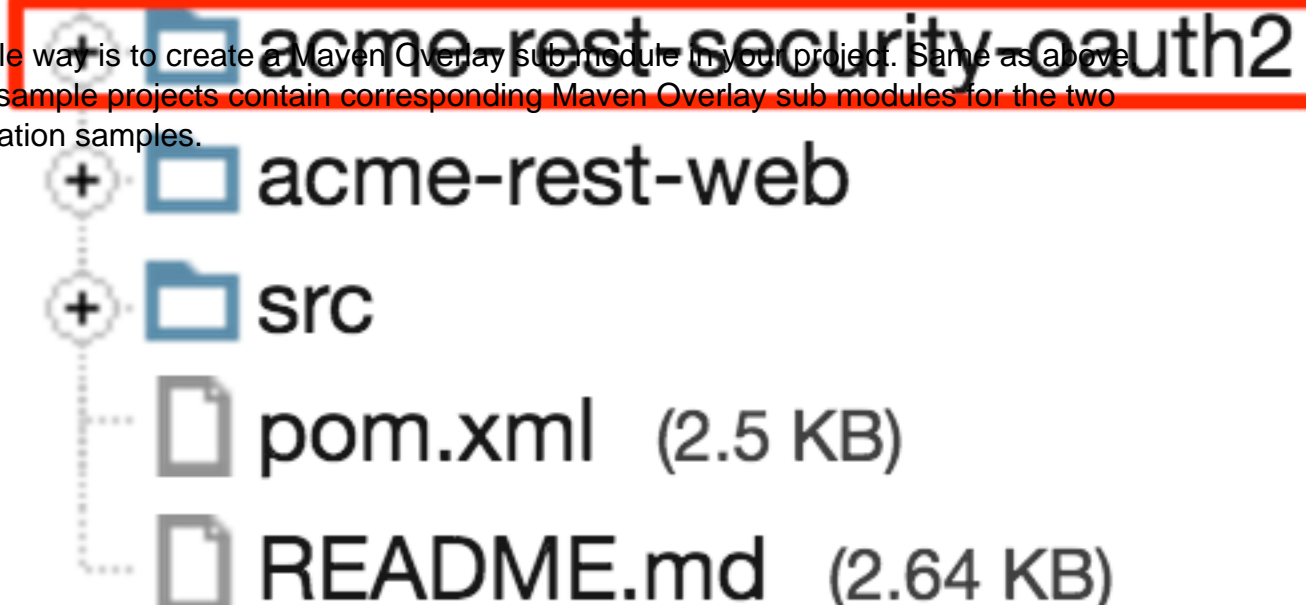


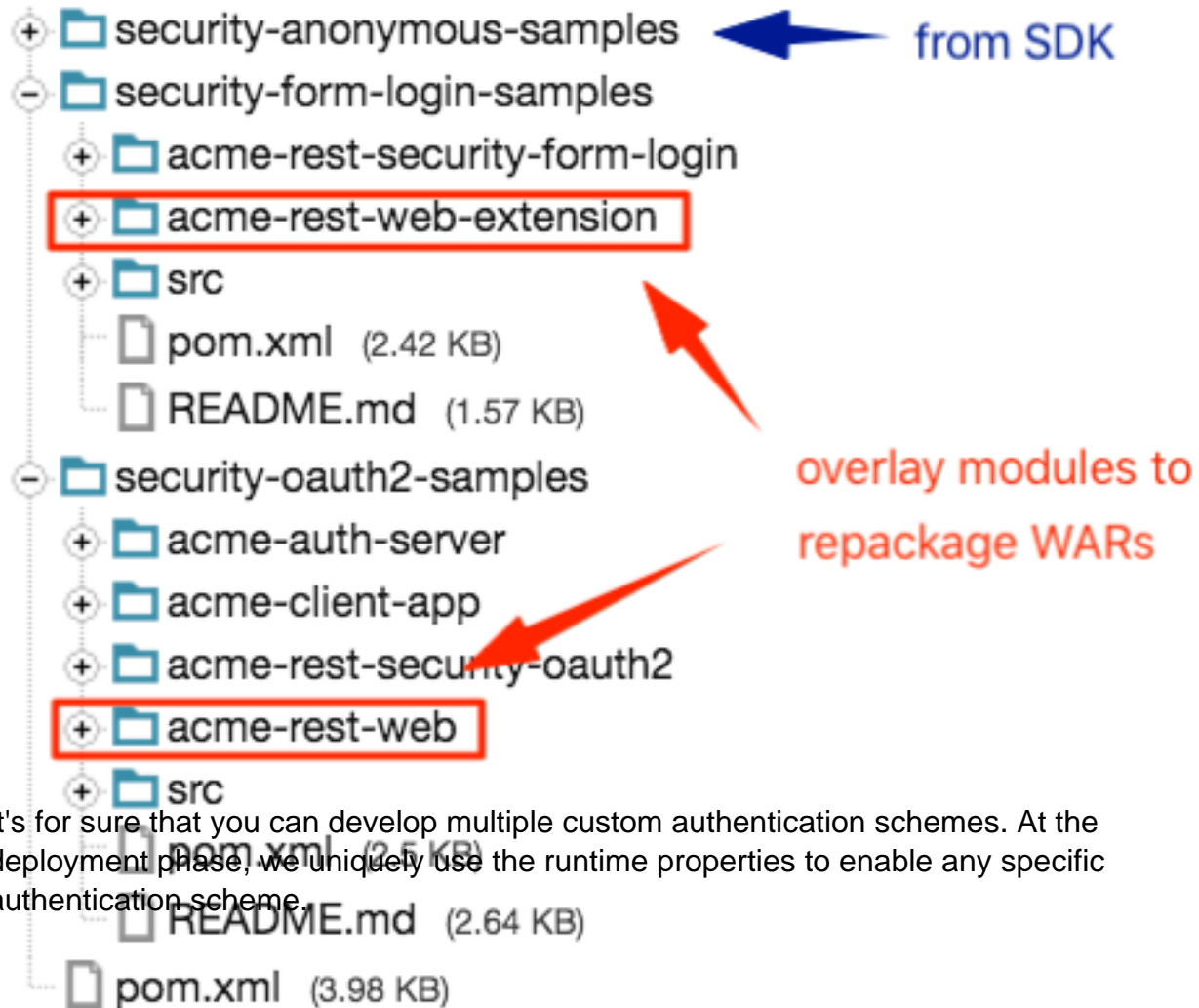
Applying to REST deployment

Once the module is developed, we need to complete the following 2 tasks:

- Add the custom authentication jar file into REST WAR file
- Configure runtime properties to enable the custom authentication

The simple way is to create a Maven Overlay sub module in your project. Same as above, the SDK sample projects contain corresponding Maven Overlay sub modules for the two authentication samples.





It's for sure that you can develop multiple custom authentication schemes. At the deployment phase, we uniquely use the runtime properties to enable any specific authentication scheme.

```
# auth mode rest.security.auth.mode=your_auth_type # other auth related properties ...
```

Documentum REST Services Developer Guide 7.3 has much more detail explanation on the Java classes to implement the custom authentication.

Exploring OAuth2 authentication sample

Now let's take some time to explore the OAuth2 sample project in Documentum REST SDK, located at **dctm-rest-sdk-7.3\samples\security-oauth2-samples**. It is a multi-module Maven project.

Components

- **acme-auth-server** builds out a WAR file for a very simple OAuth2 authentication service. It support OAuth2 authorization types [Authorization Code Grant](#) and [Implicit Grant](#). It supports the [Bearer authentication token](#) in [JWT](#). The server has an in-memory defined user *dadmin*.
- **acme-client-app** builds out a sample HTML5 application in WAR file which consumes Documentum REST Services. The project was modified based on the [Documentum REST HTML5 Client Samples](#) on GitHub. The login page redirects the unauthorized user agent to the OAuth2 authentication server.
- **acme-rest-security-oauth2** builds out a jar file which supports OAuth2 authentication on Documentum REST Services.
- **acme-rest-web** builds out a custom REST WAR file by adding **acme-rest-security-oauth2** as its library. It enables OAuth2 as the authentication scheme by configuring REST runtime properties.

Please follow **README.md** in each module to get more information on the project code. Our focus is to explore the modules **acme-rest-security-oauth2** and **acme-rest-web** to understand how to build the custom authentication.

OAuth Simplification

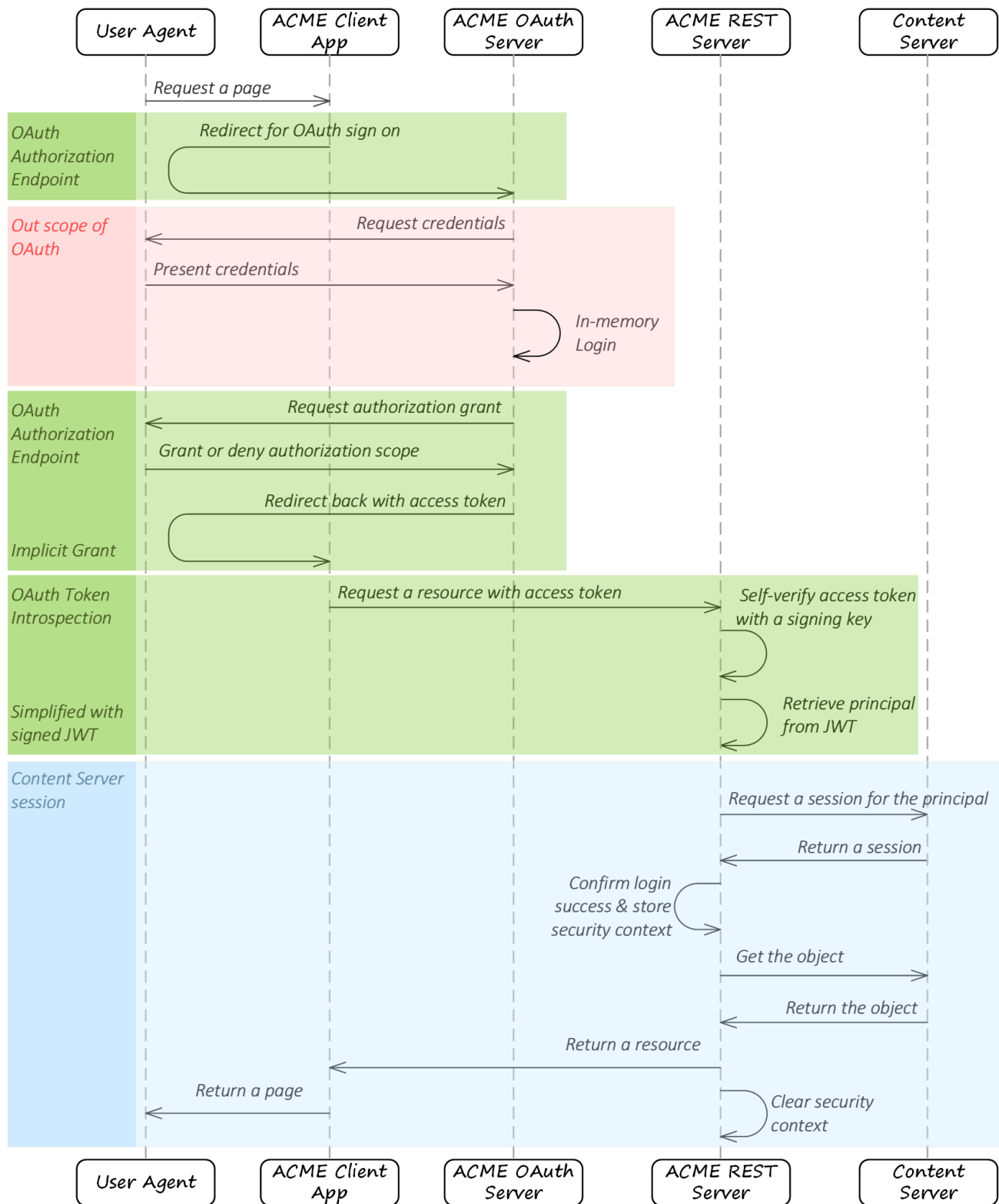
Our purpose for this reference implementation is merely to demonstrate the security extensibility feature of Documentum REST Services. Thus, in the reference implementation, we implement a simplified authorization flow of OAuth 2.0. The OAuth flow is simplified in these areas:

- The authorization grant between the client and the authorization server is implemented as ***Implicit Grant***. It simplifies the demo client application implementation which uses JavaScripts to perform the OAuth 2.0 authorization.
- The authentication part on the authorization server is implemented as ***In-memory authentication***. It avoids additional login configuration on the demo authorization server.
- The access token scope for the authorization grant defines three values read, write, delete. The demo application requires to ***grant all permissions*** to access the protected resources.

Customizing authentications in Documentum REST Services

- The access token validation between the resource server and the authorization server is implemented by the self-signed **JWT token**. We notice that there is a new standard [OAuth 2.0 Token Introspection](#) available since October 2015. However, at the moment we develop this reference implementation, it is not supported by [Spring Security OAuth Extension](#) yet.

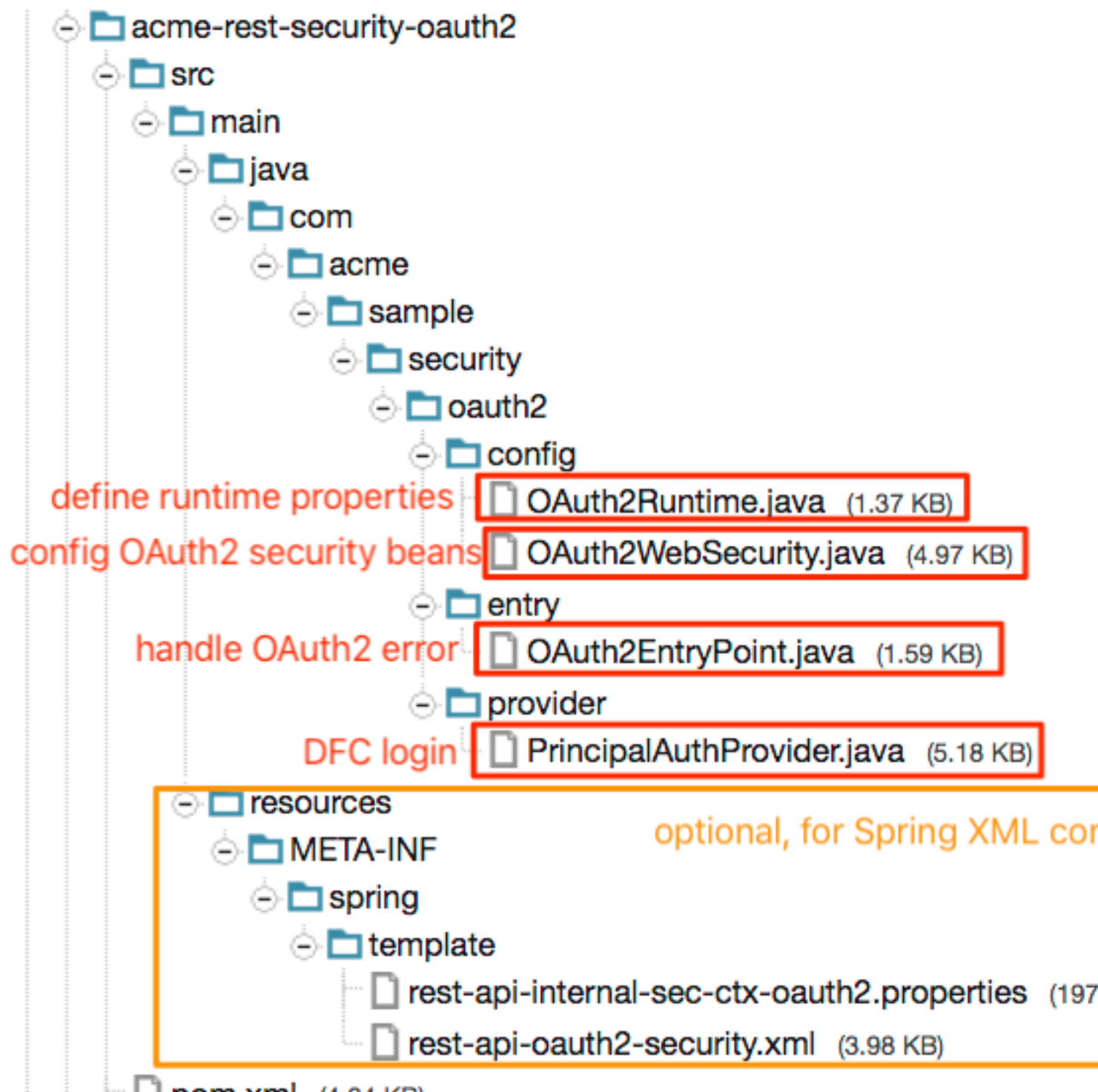
Below is the diagram of the authentication workflow in this OAuth2 sample application.



Documentum REST Services for OAuth 2.0 Reference Implementation

Exploring the security module

The security module mainly contains Java source code to implement and configure OAuth2 authentication. We highlight key source files below.



In the source code files, the most important one is **OAuth2WebSecurity**. This file configures the implementation beans (e.g. filter, provider, endpoint, etc.) with a condition of **rest.security.auth.mode=oauth2**. Please note **@AuthSchemeProfile** and **AbstractWebSecurity** are Documentum REST library classes for developers to configure the security.

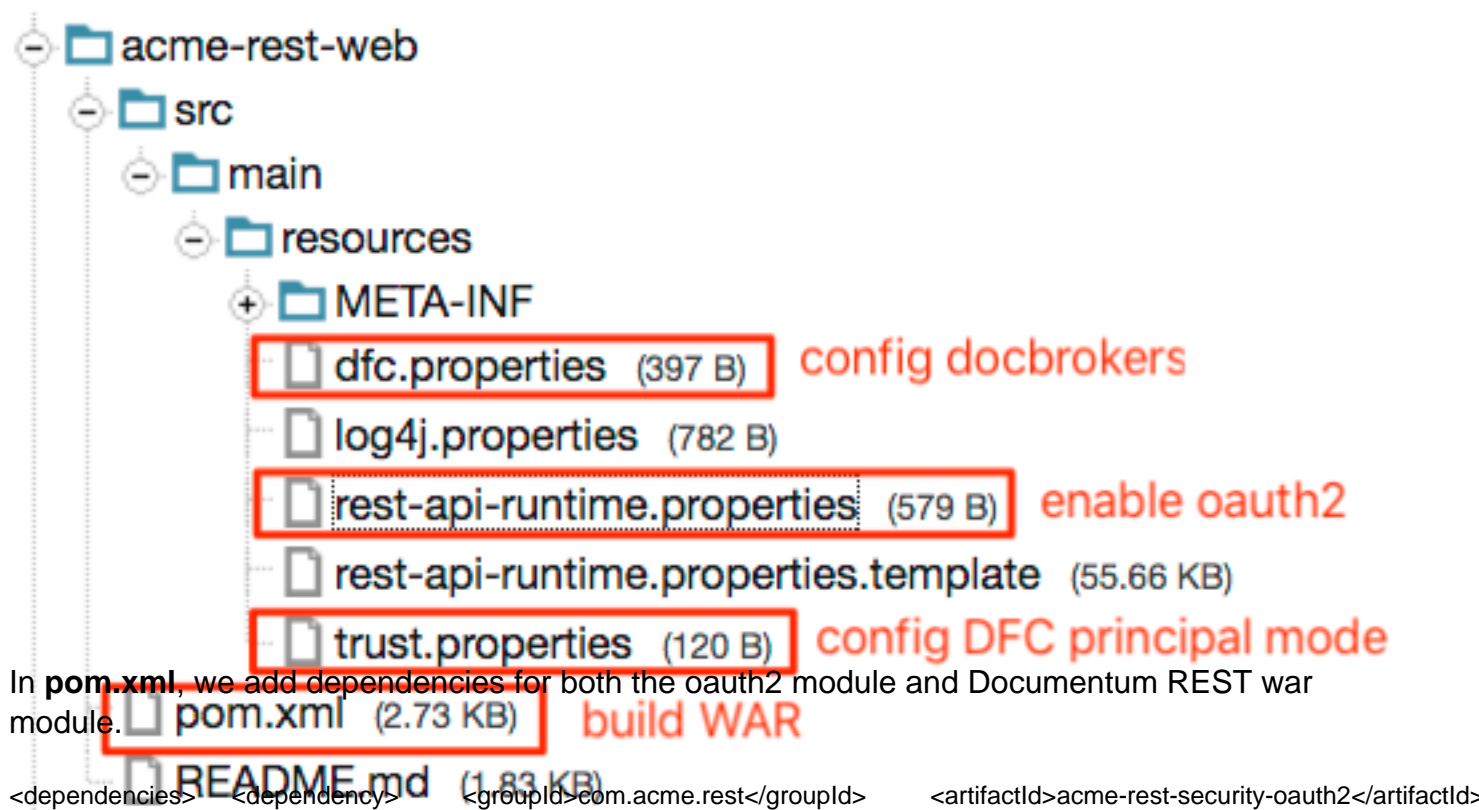
```
@AuthSchemeProfile(schemes = OAuth2WebSecurity.MODE) @Order(1) public class OAuth2WebSecurity extends AbstractWebS
```

The other source file to highlight is **PrincipalAuthProvider**. In this sample project, we did not develop a new Content Server login plugin, but instead just reuse DFC principal mode to enable the Content Server login for OAuth2 users. From the code you will find out that we get user name from OAuth2 JWT token and logins to Content Server in **AuthType.PRINCIPAL** with **MemoryRepositorySessionManager** provided by Documentum REST library. The **MemoryRepositorySessionManager** supports a rich set of Content Server login types, and is the class you mainly rely on for Documentum login in any custom authentication type.

```
public class PrincipalAuthProvider extends AbstractAuthProvider implements AuthenticationProvider { ... @Override public A
```

Exploring the web module

The web module contains build scripts and configuration files. We highlight key configurations below.

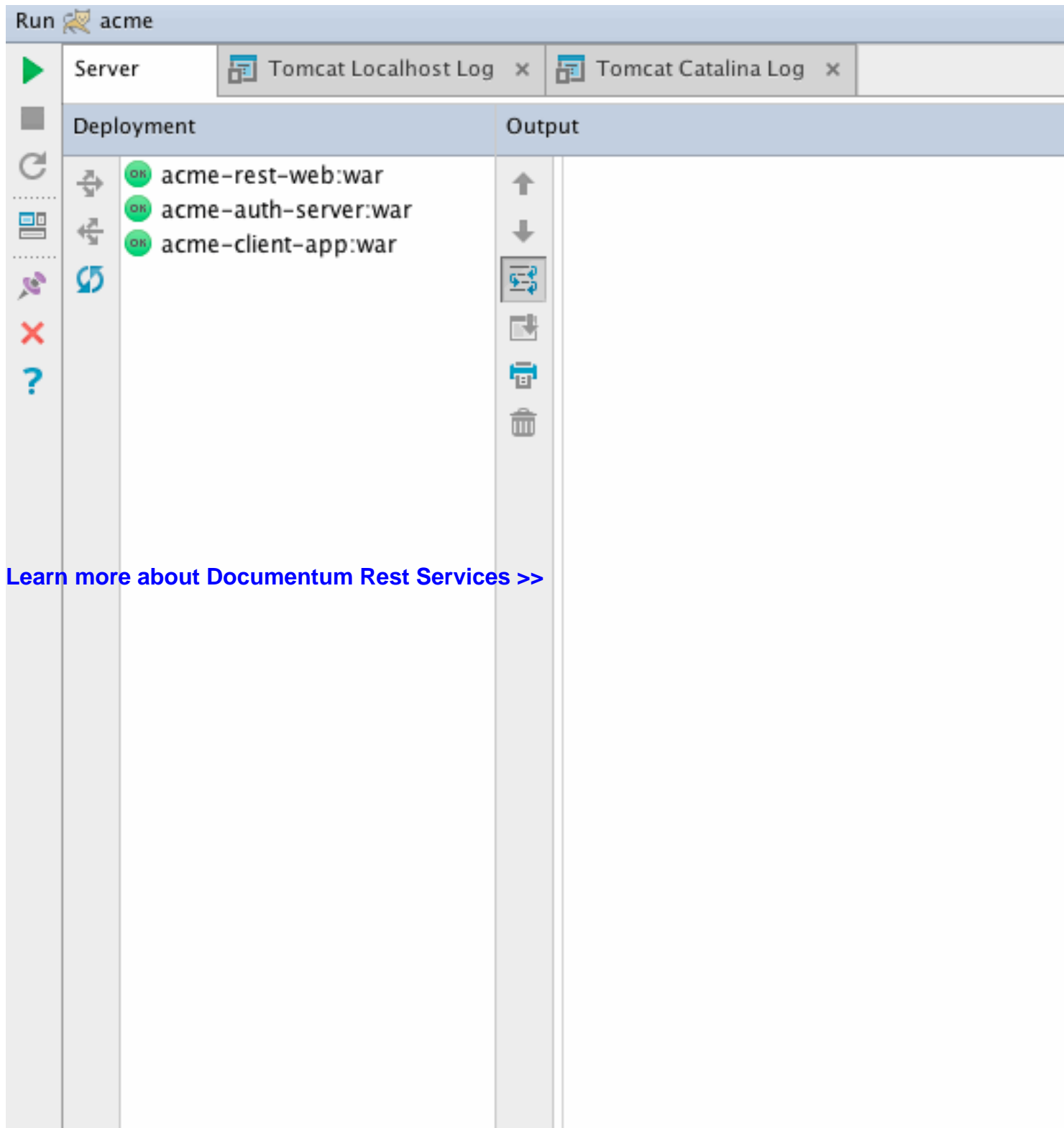


In **rest-api-runtime.properties**, we enable oauth2 authentication scheme.

```
# auth mode rest.security.auth.mode=oauth2 rest.security.realm.name=acme.com rest.context.config.location=com.acme.rest.secur
```

Demo

Below is a demo animation to show the OAuth2 authentication from the SDK sample project. By building the project correctly, you can run the demo in your own environment. **Click on the image to view the animation!**



[Learn more about Documentum Rest Services >>](#)