# Getting Started with Cayenne ROP (Remote Object Persistence)

Version 4.2 (4.2.M1)

# Table of Contents

1. Prerequisites . . . 2
   1.1. Prerequisites . . . 2
2. Remote Object Persistence Quick Start . . . 3
   2.1. Starting Client Project . . . 3
   2.2. Setting up Hessian Web Service . . . 6
   2.3. Porting Existing Code to Connect to a Web Service Instead of a Database . . . 8
   2.4. Adding BASIC Authentication . . . 12

Copyright © 2011-2020 Apache Software Foundation and individual authors

*License*

*Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at https://www.apache.org/licenses/LICENSE-2.0*

*Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.*

# Chapter 1. Prerequisites

## 1.1. Prerequisites

This tutorial starts where "Getting Started with Cayenne" left off. If you have gone through the previous tutorial, and have the "tutorial" project open in Eclipse, you can go directly to the next step. If not, here are the compressed instructions to prepare you for work with ROP:

- Step 1 - Eclipse Setup
- Step 2 - Create a project
- Step 3 - Create Cayenne OR Mapping
- Step 4 - Create Java Classes
- Step 5 - Convert the project to webapp.

Note that at "Step 5" you can skip the JSP creation part. Just setup `web.xml` and `maven-jetty-plugin` in the POM.

# Chapter 2. Remote Object Persistence Quick Start

## 2.1. Starting Client Project

### 2.1.1. Create an ROP Client Project in Eclipse

Creation of a new Eclipse project has been discussed in some details in "Getting Started with Cayenne" guide, so we will omit the screenshots for the common parts.

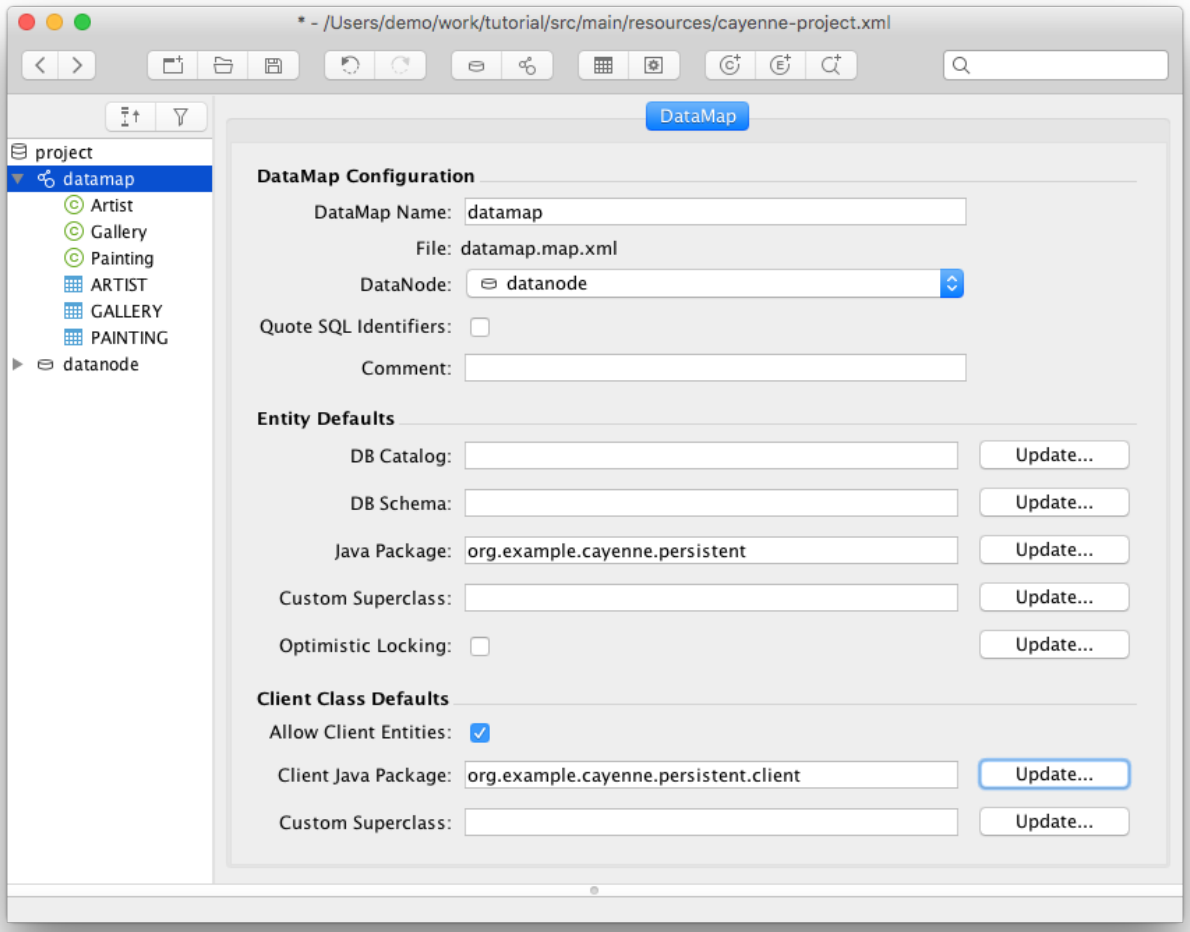
In Eclipse select "File > New > Other..." and then "Maven > Maven Project". Click "Next". On the following screen check "Create a simple project" checkbox and click "Next" again. In the dialog shown on the screenshot below, enter "org.example.cayenne" for the "Group Id" and "tutorial-rop-client" for the "Artifact Id" (both without the quotes) and click "Finish".

Now you should have a new empty project in the Eclipse workspace. Check that the project Java compiler settings are correct. Rightclick on the "tutorial-rop-client" project, select "Properties > Java Compiler" and ensure that "Compiler compliance level" is at least 1.5 (some versions of Maven plugin seem to be setting it to 1.4 by default).

### 2.1.2. Create Client Java Classes

The client doesn't need the XML ORM mapping, as it is loaded from the server. However it needs the client-side Java classes. Let's generate them from the existing mapping:

- Start CayenneModeler and open `cayenne.xml` from the "tutorial" project (located under `tutorial/src/main/resources`, unless it is already open.
- Select the "datamap" DataMap and check "Allow Client Entities" checkbox.
- Enter `org.example.cayenne.persistent.client` for the "Client Java Package" and click "Update.." button next to the field to refresh the client package of all entities.




- Select "Tools > Generate Classes" menu.
- For "Type" select "Client Persistent Objects".
- For the "Output Directory" select `tutorial-rop-client/src/main/java` folder (as client classes should go in the client project).
- Click on "Classes" tab and check the "Check All Classes" checkbox (unless it is already checked and reads "Uncheck all Classes").
- Click "Generate".

Now go back to Eclipse, right click on "tutorial-rop-client" project and select "Refresh" - you should see pairs of classes generated for each mapped entity, same as on the server. And again, we see a bunch of errors in those classes. Let's fix it now by adding two dependencies, "cayenne-client" and "hessian", in the bottom of the pom.xml file. We also need to add Caucho M2 repository to pull Hessian jar files. The resulting POM should look like this:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.example.cayenne</groupId>
    <artifactId>tutorial-rop-client</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.apache.cayenne</groupId>
            <artifactId>cayenne-client-jetty</artifactId>
            <!-- Here specify the version of Cayenne you are actually using -->
            <version>4.2.M1</version>
        </dependency>
        <dependency>
        <groupId>com.caucho</groupId>
            <artifactId>hessian</artifactId>
            <version>4.0.38</version>
        </dependency>
    </dependencies>

    <repositories>
        <repository>
            <id>caucho</id>
            <name>Caucho Repository</name>
            <url>http://caucho.com/m2</url>
            <layout>default</layout>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
            <releases>
                <enabled>true</enabled>
            </releases>
        </repository>
    </repositories>
</project>
```

Your computer must be connected to the internet. Once you save the pom.xml, Eclipse will download the needed jar files and add them to the project build path. After that all the errors should disappear.

Now let's check the entity class pairs. They look almost identical to their server counterparts, although the superclass and the property access code are different. At this point these differences are somewhat academic, so let's go on with the tutorial.

## 2.2. Setting up Hessian Web Service

### 2.2.1. Setting up Dependencies

Now lets get back to the "tutorial" project that contains a web application and set up dependencies. Let's add `resin-hessian.jar` (and the caucho repo declaration) and `cayenne-rop-server` to the `pom.xml`

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    ...
    <dependencies>
        ...
        <dependency>
            <groupId>org.apache.cayenne</groupId>
            <artifactId>cayenne-rop-server</artifactId>
            <!-- Here specify the version of Cayenne you are actually using -->
            <version>{version}</version>
        </dependency>
        <dependency>
            <groupId>com.caucho</groupId>
            <artifactId>hessian</artifactId>
            <version>4.0.38</version>
        </dependency>
    </dependencies>

    <build>
    ...
    </build>

    <repositories>
        <repository>
            <id>caucho</id>
            <name>Caucho Repository</name>
            <url>http://caucho.com/m2</url>
            <layout>default</layout>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
            <releases>
                <enabled>true</enabled>
            </releases>
        </repository>
    </repositories>
    </project>
```



*Maven Optimization Hint*

On a real project both server and client modules will likely share a common parent `pom.xml` where common repository delcaration can be placed, with child pom's "inheriting" it from parent. This would reduce build code duplication.

### 2.2.2. Client Classes on the Server

Since ROP web service requires both server and client persistent classes, we need to generate a second copy of the client classes inside the server project. This is a minor inconvenience that will hopefully go away in the future versions of Cayenne. Don't forget to refresh the project in Eclipse after class generation is done.

### 2.2.3. Configuring web.xml

Cayenne web service is declared in the web.xml. It is implemented as a servlet `org.apache.cayenne.rop.ROPServlet`. Open `tutorial/src/main/webapp/WEB-INF/web.xml` in Eclipse and add a service declaration:

```
<?xml version="1.0" encoding="utf-8"?>
 <!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
   "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>Cayenne Tutorial</display-name>
    <servlet>
        <servlet-name>cayenne-project</servlet-name>
        <servlet-class>org.apache.cayenne.rop.ROPServlet</servlet-class>
        <load-on-startup>0</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>cayenne-project</servlet-name>
        <url-pattern>/cayenne-service</url-pattern>
    </servlet-mapping>
    </web-app>
```



*Extending Server Behavior via Callbacks*

While no custom Java code is required on the server, just a service declaration, it is possible to customizing server-side behavior via callbacks and listeners (not shown in the tutorial).

### 2.2.4. Running ROP Server

Use previosly created Eclipse Jetty run configuration available via "Run > Run Configurations..." (or create a new one if none exists yet). You should see output in the Eclipse console similar to the following:

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building tutorial 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
...
[INFO] Starting jetty 6.1.22 ...
INFO::jetty-6.1.22
INFO::No Transaction manager found - if your webapp requires one, please configure
one.
INFO::Started SelectChannelConnector@0.0.0.0:8080
[INFO] Started Jetty Server
INFO: Loading XML configuration resource from file:cayenne-project.xml
INFO: loading user name and password.
INFO: Created connection pool: jdbc:derby:memory:testdb;create=true
    Driver class: org.apache.derby.jdbc.EmbeddedDriver
    Min. connections in the pool: 1
    Max. connections in the pool: 1
```

Cayenne ROP service URL is http://localhost:8080/tutorial/cayenne-service. If you click on it, you will see "Hessian Requires POST" message, that means that the service is alive, but you need a client other than the web browser to access it.

## 2.3. Porting Existing Code to Connect to a Web Service Instead of a Database

### 2.3.1. Starting Command Line Client

One of the benefits of ROP is that the client code is no different from the server code - it uses the same ObjectContext interface for access, same query and commit API. So the code below will be similar to the code presented in the first Cayenne Getting Started Guide, although with a few ROP-specific parts required to bootstrap the ObjectContext.

Let's start by creating an empty Main class with the standard main() method in the client project:

```
package org.example.cayenne.persistent.client;

public class Main {

    public static void main(String[] args) {

    }
}
```

Now the part that is actually different from regular Cayenne - establishing the server connection and obtaining the ObjectContext:

```
Map<String, String> properties = new HashMap<>();
properties.put(ClientConstants.ROP_SERVICE_URL_PROPERTY,
"http://localhost:8080/cayenne-service");
properties.put(ClientConstants.ROP_SERVICE_USERNAME_PROPERTY, "cayenne-user");
properties.put(ClientConstants.ROP_SERVICE_PASSWORD_PROPERTY, "secret");
properties.put(ClientConstants.ROP_SERVICE_REALM_PROPERTY, "Cayenne Realm");


ClientRuntime runtime = ClientRuntime.builder()
                        .properties(properties)
                        .addModule(new ClientJettyHttpModule())
                        .build();
ObjectContext context = runtime.newContext();
```

Note that the "runtime" can be used to create as many peer ObjectContexts as needed over the same connection, while ObjectContext is a kind of isolated "persistence session", similar to the server-side context. A few more notes. Since we are using HTTP(S) to communicate with ROP server, there's no need to explicitly close the connection (or channel, or context).

So now let's do the same persistent operaions that we did in the first tutorial "Main" class. Let's start by creating and saving some objects:

```
// creating new Artist
Artist picasso = context.newObject(Artist.class);
picasso.setName("Pablo Picasso");

// Creating other objects
Gallery metropolitan = context.newObject(Gallery.class);
metropolitan.setName("Metropolitan Museum of Art");

Painting girl = context.newObject(Painting.class);
girl.setName("Girl Reading at a Table");

Painting stein = context.newObject(Painting.class);
stein.setName("Gertrude Stein");

// connecting objects together via relationships
picasso.addToPaintings(girl);
picasso.addToPaintings(stein);

girl.setGallery(metropolitan);
stein.setGallery(metropolitan);

// saving all the changes above
context.commitChanges();
```

Now let's select them back:

```
// ObjectSelect examples
List<Painting> paintings1 = ObjectSelect.query(Painting.class).select(context);

List<Painting> paintings2 = ObjectSelect.query(Painting.class)
        .where(Painting.NAME.likeIgnoreCase("gi%")).select(context);
```

Now, delete:

```
// Delete object example
Artist picasso = ObjectSelect.query(Artist.class).where(Artist.NAME.eq("Pablo Picasso
")).selectOne(context);

if (picasso != null) {
    context.deleteObject(picasso);
    context.commitChanges();
}
```

This code is exactly the same as in the first tutorial. So now let's try running the client and see what happens. In Eclipse open main class and select "Run > Run As > Java Application" from the menu (assuming the ROP server started in the previous step is still running). You will some output in both server and client process consoles. Client:

```
INFO: Connecting to [http://localhost:8080/tutorial/cayenne-service] - dedicated
session.
INFO: === Connected, session:
org.apache.cayenne.remote.RemoteSession@26544ec1[sessionId=17uub1h34r9x1] - took 111
ms.
INFO: --- Message 0: Bootstrap
INFO: === Message 0: Bootstrap done - took 58 ms.
INFO: --- Message 1: flush-cascade-sync
INFO: === Message 1: flush-cascade-sync done - took 1119 ms.
INFO: --- Message 2: Query
INFO: === Message 2: Query done - took 48 ms.
INFO: --- Message 3: Query
INFO: === Message 3: Query done - took 63 ms.
INFO: --- Message 4: Query
INFO: === Message 4: Query done - took 19 ms.
INFO: --- Message 5: Query
INFO: === Message 5: Query done - took 7 ms.
INFO: --- Message 6: Query
INFO: === Message 6: Query done - took 5 ms.
INFO: --- Message 7: Query
INFO: === Message 7: Query done - took 2 ms.
INFO: --- Message 8: Query
INFO: === Message 8: Query done - took 4 ms.
INFO: --- Message 9: flush-cascade-sync
INFO: === Message 9: flush-cascade-sync done - took 34 ms.
```

As you see client prints no SQL statmenets, just a bunch of query and flush messages sent to the server. The server side is more verbose, showing the actual client queries executed against the database:

```
...
INFO: SELECT NEXT_ID FROM AUTO_PK_SUPPORT WHERE TABLE_NAME = ? FOR UPDATE [bind:
1:'ARTIST']
INFO: SELECT NEXT_ID FROM AUTO_PK_SUPPORT WHERE TABLE_NAME = ? FOR UPDATE [bind:
1:'GALLERY']
INFO: SELECT NEXT_ID FROM AUTO_PK_SUPPORT WHERE TABLE_NAME = ? FOR UPDATE [bind:
1:'PAINTING']
INFO: INSERT INTO ARTIST (DATE_OF_BIRTH, ID, NAME) VALUES (?, ?, ?)
INFO: [batch bind: 1->DATE_OF_BIRTH:NULL, 2->ID:200, 3->NAME:'Pablo Picasso']
INFO: === updated 1 row.
INFO: INSERT INTO GALLERY (ID, NAME) VALUES (?, ?)
INFO: [batch bind: 1->ID:200, 2->NAME:'Metropolitan Museum of Art']
INFO: === updated 1 row.
INFO: INSERT INTO PAINTING (ARTIST_ID, GALLERY_ID, ID, NAME) VALUES (?, ?, ?, ?)
INFO: [batch bind: 1->ARTIST_ID:200, 2->GALLERY_ID:200, 3->ID:200, 4->NAME:'Girl
Reading at a Table']
INFO: [batch bind: 1->ARTIST_ID:200, 2->GALLERY_ID:200, 3->ID:201, 4->NAME:'Gertrude
Stein']
INFO: === updated 2 rows.
INFO: +++ transaction committed.
INFO: --- transaction started.
INFO: SELECT t0.GALLERY_ID, t0.NAME, t0.ARTIST_ID, t0.ID FROM PAINTING t0
INFO: === returned 2 rows. - took 14 ms.
INFO: +++ transaction committed.
INFO: --- transaction started.
INFO: SELECT t0.GALLERY_ID, t0.NAME, t0.ARTIST_ID, t0.ID FROM PAINTING t0
      WHERE UPPER(t0.NAME) LIKE UPPER(?) [bind: 1->NAME:'gi%']
INFO: === returned 1 row. - took 10 ms.
INFO: +++ transaction committed.
INFO: --- transaction started.
INFO: SELECT t0.DATE_OF_BIRTH, t0.NAME, t0.ID FROM ARTIST t0 WHERE t0.NAME = ? [bind:
1->NAME:'Pablo Picasso']
INFO: === returned 1 row. - took 8 ms.
INFO: +++ transaction committed.
INFO: --- transaction started.
INFO: DELETE FROM PAINTING WHERE ID = ?
INFO: [batch bind: 1->ID:200]
INFO: [batch bind: 1->ID:201]
INFO: === updated 2 rows.
INFO: DELETE FROM ARTIST WHERE ID = ?
INFO: [batch bind: 1->ID:200]
INFO: === updated 1 row.
INFO: +++ transaction committed.
```

You are done with the basic ROP client!

## 2.4. Adding BASIC Authentication

You probably don't want everybody in the world to connect to your service and access (and update!) arbitrary data in the database. The first step in securing Cayenne service is implementing client authentication. The easiest way to do it is to delegate the authentication task to the web container that is running the service. HessianConnection used in the previous chapter supports BASIC authentication on the client side, so we'll demonstrate how to set it up here.

### 2.4.1. Securing ROP Server Application

Open web.xml file in the server project and setup security constraints with BASIC authentication for the ROP service:

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>CayenneService</web-resource-name>
        <url-pattern>/cayenne-service</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>cayenne-service-user</role-name>
    </auth-constraint>
</security-constraint>

<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Cayenne Realm</realm-name>
</login-config>

<security-role>
    <role-name>cayenne-service-user</role-name>
</security-role>
```

### 2.4.2. Configuring Jetty for BASIC Authentication



These instructions are specific to Jetty 6. Other containers (and versions of Jetty) will have different mechanisms to achieve the same thing.

Open pom.xml in the server project and configure a "userRealm" for the Jetty plugin:

```
<plugin>
    <groupId>org.eclipse.jetty</groupId>
        <artifactId>maven-jetty-plugin</artifactId>
        <version>9.4.8.v20171121</version>
        <!-- adding configuration below: -->
        <configuration>
            <userRealms>
                <userRealm implementation="
org.eclipse.jetty.security.HashLoginService">
                    <!-- this name must match the realm-name in web.xml -->
                    <name>Cayenne Realm</name>
                    <config>realm.properties</config>
                </userRealm>
            </userRealms>
        </configuration>
    </plugin>
</plugins>
```

Now create a new file called `realm.properties` at the root of the server project and put user login/password in there:

```
cayenne-user: secret,cayenne-service-user
```

Now let's stop the server and start it again. Everything should start as before, but if you go to http://localhost:8080/tutorial/cayenne-service, your browser should pop up authentication dialog. Enter "cayenne-user/secret" for user name / password, and you should see "Hessian Requires POST" message. So the server is now secured.

### 2.4.3. Running Client with Basic Authentication

If you run the client without any changes, you'll get the following error:

```
Mar 01, 2016 7:25:50 PM org.apache.cayenne.rop.http.HttpROPConnector logConnect
INFO: Connecting to [cayenne-user@http://localhost:8080/tutorial-rop-server/cayenne-
service] - dedicated session.
Mar 01, 2016 7:25:50 PM org.apache.cayenne.rop.HttpClientConnection connect
INFO: Server returned HTTP response code: 401 for URL: http://localhost:8080/tutorial-
rop-server/cayenne-service
java.rmi.RemoteException: Server returned HTTP response code: 401 for URL:
http://localhost:8080/tutorial-rop-server/cayenne-service
    at
org.apache.cayenne.rop.ProxyRemoteService.establishSession(ProxyRemoteService.java:45)
    at
org.apache.cayenne.rop.HttpClientConnection.connect(HttpClientConnection.java:85)
    at
org.apache.cayenne.rop.HttpClientConnection.getServerEventBridge(HttpClientConnection.
java:68)
    at
org.apache.cayenne.remote.ClientChannel.setupRemoteChannelListener(ClientChannel.java:
279)
    at org.apache.cayenne.remote.ClientChannel.<init>(ClientChannel.java:71)
    at
org.apache.cayenne.configuration.rop.client.ClientChannelProvider.get(ClientChannelPro
vider.java:48)
    at
org.apache.cayenne.configuration.rop.client.ClientChannelProvider.get(ClientChannelPro
vider.java:31)
    at
org.apache.cayenne.di.spi.CustomProvidersProvider.get(CustomProvidersProvider.java:39)
    at
org.apache.cayenne.di.spi.FieldInjectingProvider.get(FieldInjectingProvider.java:43)
    at
org.apache.cayenne.di.spi.DefaultScopeProvider.get(DefaultScopeProvider.java:50)
    at org.apache.cayenne.di.spi.DefaultInjector.getInstance(DefaultInjector.java:139)
    at
org.apache.cayenne.di.spi.FieldInjectingProvider.value(FieldInjectingProvider.java:105
)
    at
org.apache.cayenne.di.spi.FieldInjectingProvider.injectMember(FieldInjectingProvider.j
ava:68)
    at
org.apache.cayenne.di.spi.FieldInjectingProvider.injectMembers(FieldInjectingProvider.
java:59)
    at
org.apache.cayenne.di.spi.FieldInjectingProvider.get(FieldInjectingProvider.java:44)
    at
org.apache.cayenne.di.spi.DefaultScopeProvider.get(DefaultScopeProvider.java:50)
    at org.apache.cayenne.di.spi.DefaultInjector.getInstance(DefaultInjector.java:134)
    at
org.apache.cayenne.configuration.CayenneRuntime.newContext(CayenneRuntime.java:134)
    at org.apache.cayenne.tutorial.persistent.client.Main.main(Main.java:44)
```

Which is exactly what you'd expect, as the client is not authenticating itself. So change the line in Main.java where we obtained an ROP connection to this:

```
Map<String,String> properties = new HashMap<>();
properties.put(ClientConstants.ROP_SERVICE_URL_PROPERTY,
"http://localhost:8080/cayenne-service");
properties.put(ClientConstants.ROP_SERVICE_USERNAME_PROPERTY, "cayenne-user");
properties.put(ClientConstants.ROP_SERVICE_PASSWORD_PROPERTY, "secret");
properties.put(ClientConstants.ROP_SERVICE_REALM_PROPERTY, "Cayenne Realm");

ClientRuntime runtime = ClientRuntime.builder()
                        .properties(properties)
                        .addModule(new ClientJettyHttpModule())
                        .build();
```

Try running again, and everything should work as before. Obviously in production environment, in addition to authentication you'll need to use HTTPS to access the server to prevent third-party eavesdropping on your password and data.

Congratulations, you are done with the ROP tutorial!