Oracle Complex Event Processing: Tutorial: CQL Processor

An Oracle Tutorial
Updated December 2013

Supported Version: Oracle Event Processing 11g (11.1.1.7)

Objectives:

This tutorial illustrates how to create a CQL processor and perform a simple "pattern matching" query, convert the output of the query into a new event type and write events to the console.

Table of Contents

Set-up: 4
Part 1: Adding a custom adapter 4
Part 2: Define the New Event Type 8
Part 3: Enhance the Event Bean 12
Part 4: Deploying the Application 14
Part 5: Testing 15

Set-up:

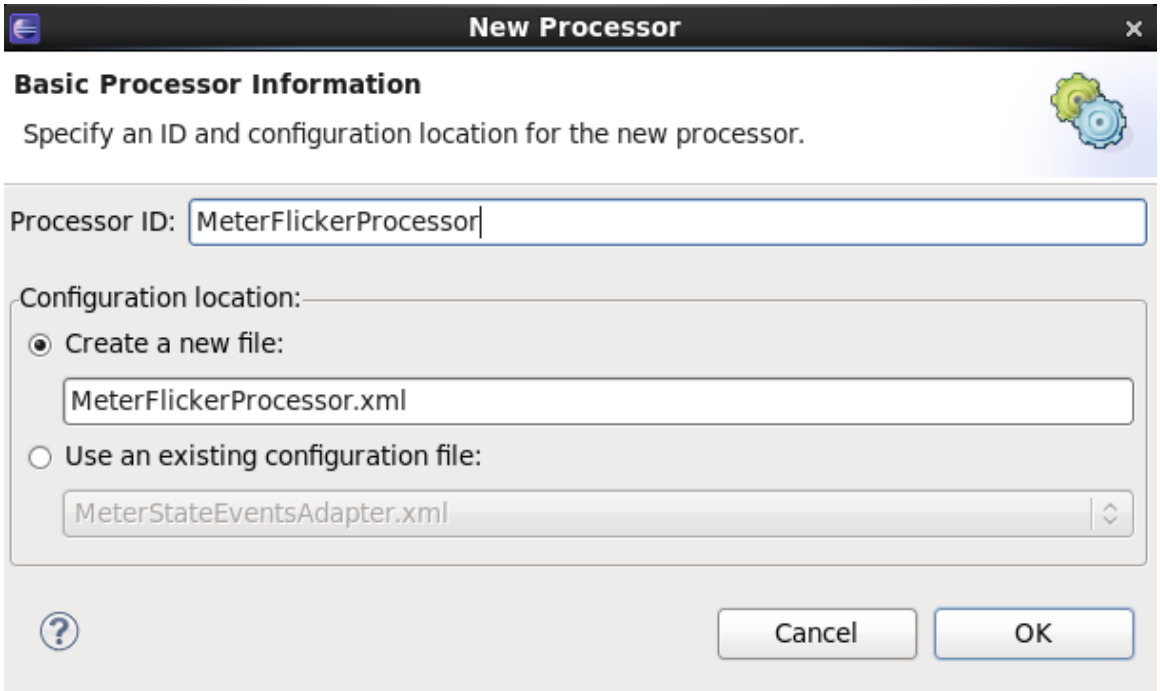
1. This tutorial is a continuation of tutorial 01_EPN_Editor. You must have an environment created to do that tutorial and complete the tutorial or import the solution.

Part 1: Adding a custom adapter

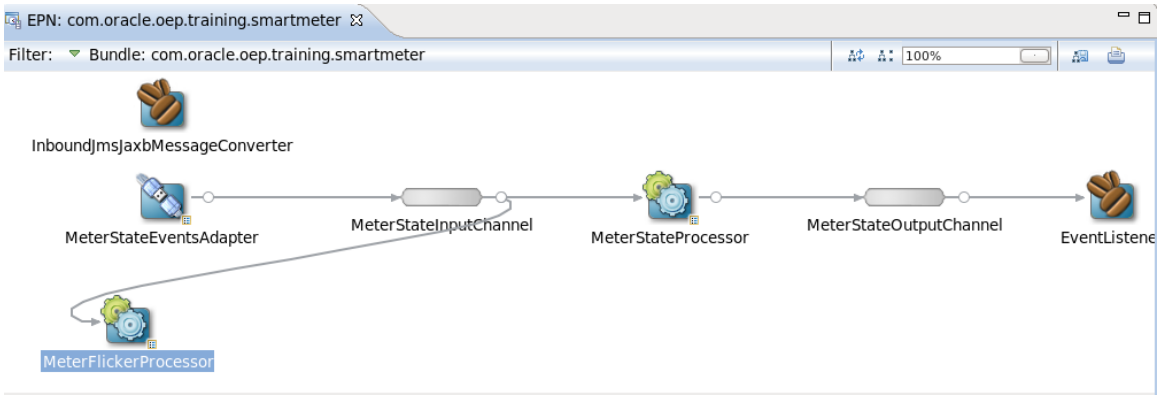
In this exercise, you will create a new custom adapter that reads information from a TCP/IP socket.

1. Open the EPN Editor for the existing "com.oracle.oep.training.smartmeter" project.

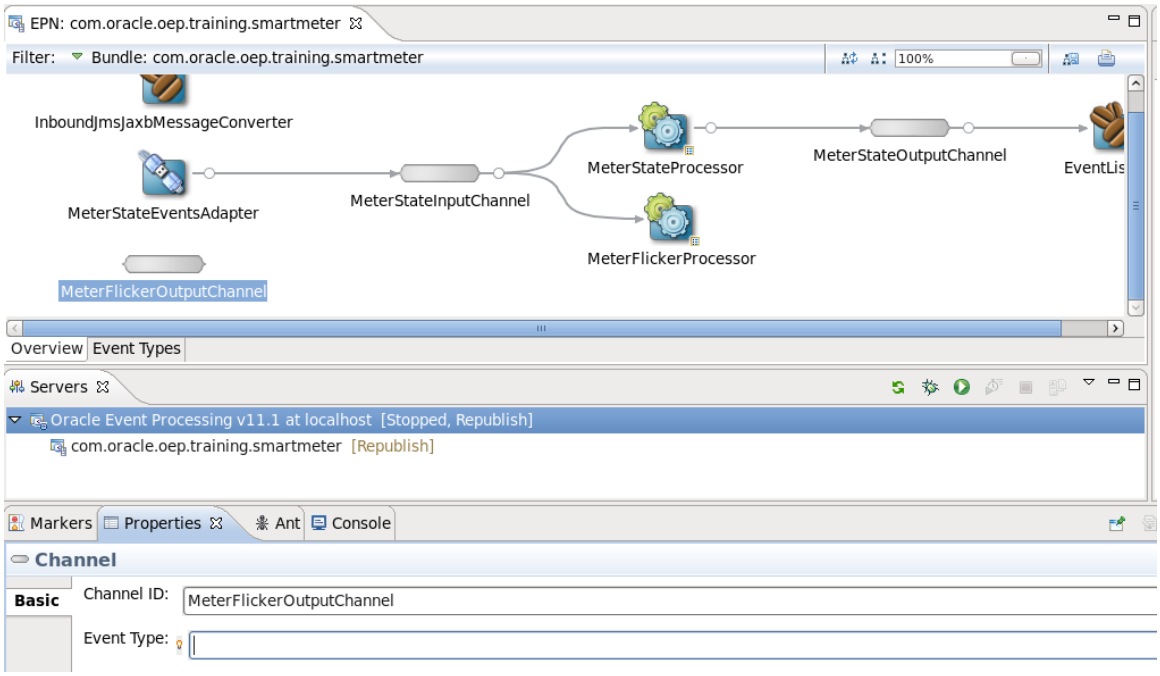
2. Right-click anywhere on the canvas and create a new processor called "MeterFlickerProcessor" and click OK.



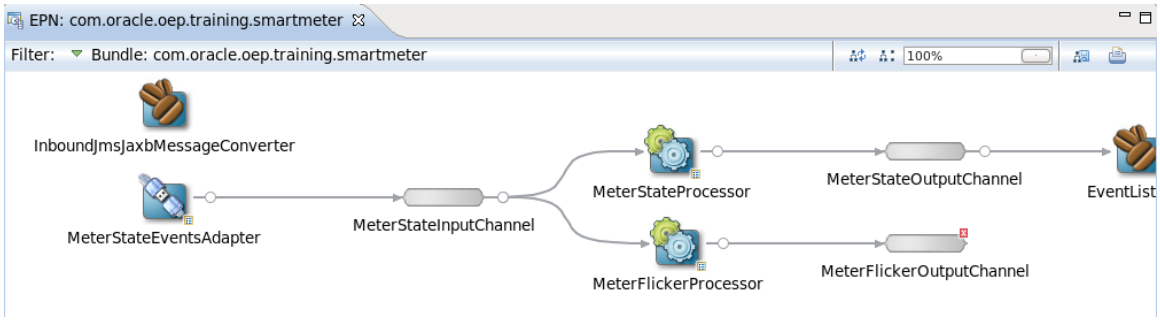
3. Join the "MeterStateInputChannel" to the new processor icon. Remember to click and hold down the mouse and drag to the end-point.



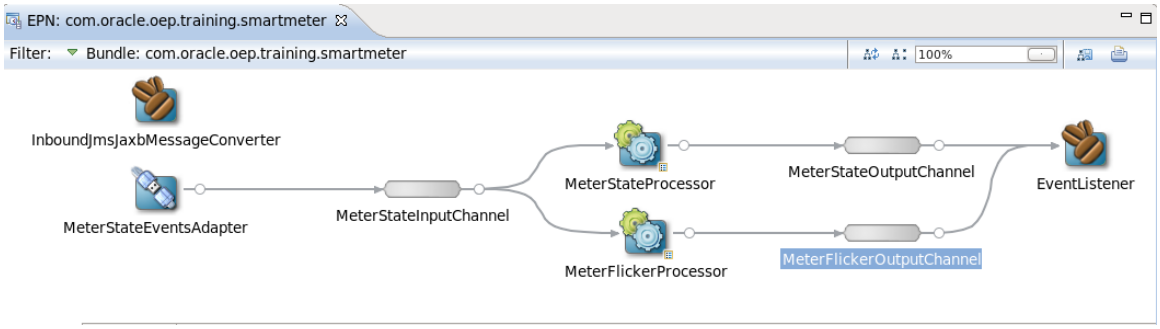
4. Create a new channel "MeterFlickerOutputChannel".



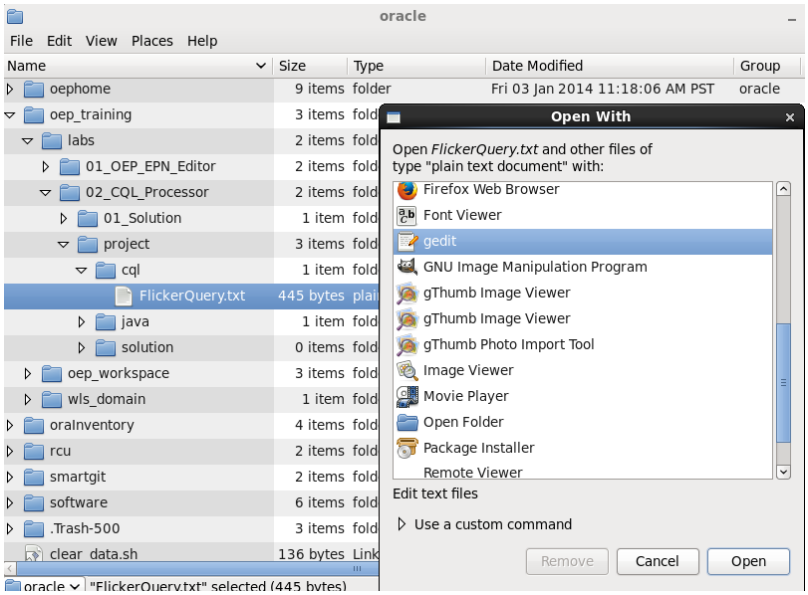
5. Join the "MeterFlickerProcessor" to the "MeterFlickerOutputChannel".



6. Join the "MeterFlickerOutputChannel" to the existing "EventListener" bean.



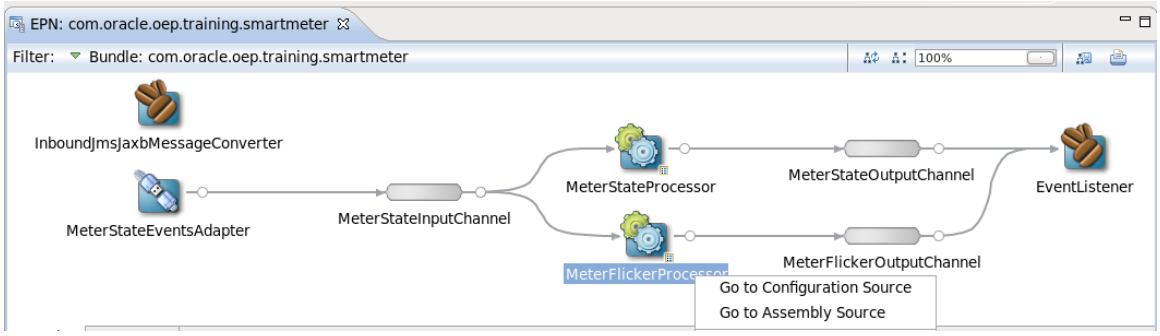
Browse to find the file "<OEP_TRAINING>\labs\02_CQL_Processor\project\cql\FlickerQuery.txt".



7. Copy the query text. (Highlight and use CTRL+C or right-click and choose "Copy")

```
<query id="FlickerQuery">
       <![CDATA[
  SELECT
       flicker.meterId as meterId,
       flicker.flickerTime as flickerTime
  FROM MeterStateInputChannel MATCH_RECOGNIZE
  (
  PARTITION BY meterId
  MEASURES
       PowerDown.meterId AS meterId,
       PowerDown.statusTime as flickerTime
  PATTERN (PowerDown PowerUp)
       DEFINE
       PowerDown AS state = 'POWER_DOWN',
       PowerUp AS state = 'POWER_UP'
  ) AS flicker
       ]]>
 </query>
```

8. Go back to the EPN Editor and right-click on the "MeterFlickerProcessor" and choose "Go to Configuration Source".



9. Replace the default query in the processor with the one that you copied from the "FlickerQuery.txt" file.

EPN: com.oracle.oep.training.smartmeter | MeterFlickerProcessor.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
              xmlns:jdbc="http://www.oracle.com/ns/ocep/config/jdbc">
    <processor>
        <name>MeterFlickerProcessor</name>
        <rules>
             <query id="FlickerQuery">
    <![CDATA[
  SELECT
    flicker.meterId as meterId,
    flicker.flickerTime as flickerTime
  FROM MeterStateInputChannel MATCH_RECOGNIZE
  (
  PARTITION BY meterId
  MEASURES
    PowerDown.meterId AS meterId,
    PowerDown.statusTime as flickerTime
  PATTERN (PowerDown PowerUp)
    DEFINE
    PowerDown AS state = 'POWER_DOWN',
    PowerUp AS state = 'POWER_UP'
  ) AS flicker
    ]]>
  </query>
        </rules>
    </processor>
</wlevs:config>
```

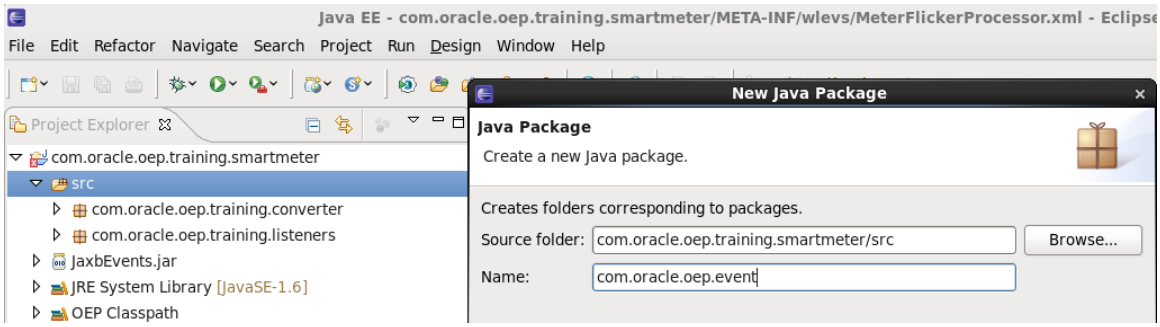
There is still an error because the downstream channel to this processor does not have an event type. You will define the event type in the next section and assign it to the "MeterFlickerOutputChannel".

```
EPN: com.oracle.oep.training.smartmeter    MeterFlickerProcessor.xml
<?xml version="1.0" encoding="UTF-8"?>
<wlevs:config xmlns:wlevs="http://www.bea.com/ns/wlevs/config/application"
              xmlns:jdbc="http://www.oracle.com/ns/ocep/config/jdbc">
    <processor>
      A channel related to this processor does not specify a required event type or specifies an event type with
      no valid properties. This problem will prevent queries for this processor configuration element from
      being validated.
  SELECT
    flicker.meterId as meterId,
    flicker.flickerTime as flickerTime
  FROM MeterStateInputChannel MATCH_RECOGNIZE
  (
  PARTITION BY meterId
  MEASURES
    PowerDown.meterId AS meterId,
    PowerDown.statusTime as flickerTime
  PATTERN (PowerDown PowerUp)
    DEFINE
    PowerDown AS state = 'POWER_DOWN',
    PowerUp AS state = 'POWER_UP'
  ) AS flicker
    ]]>
 </query>
        </rules>
    </processor>
</wlevs:config>
```

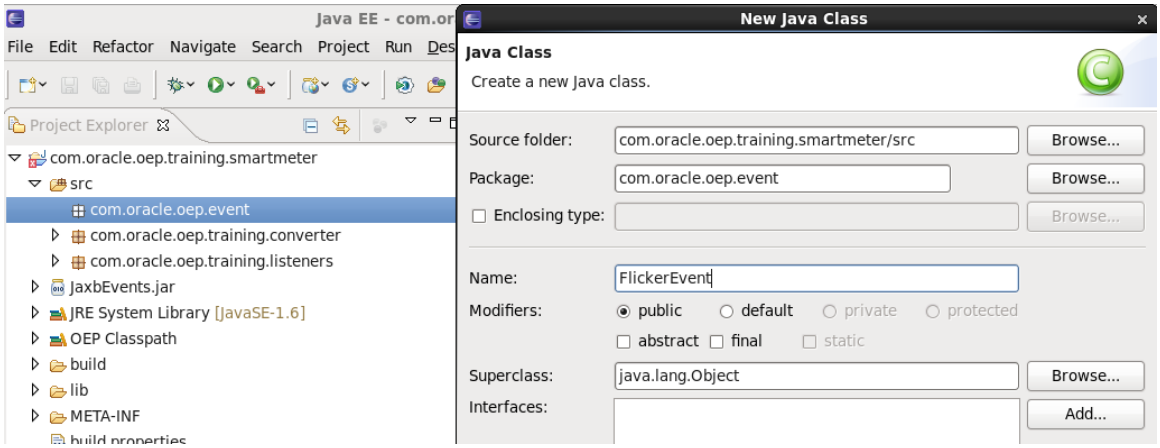
Part 2: Define the New Event Type

The query that you defined can be used to create a new event type.

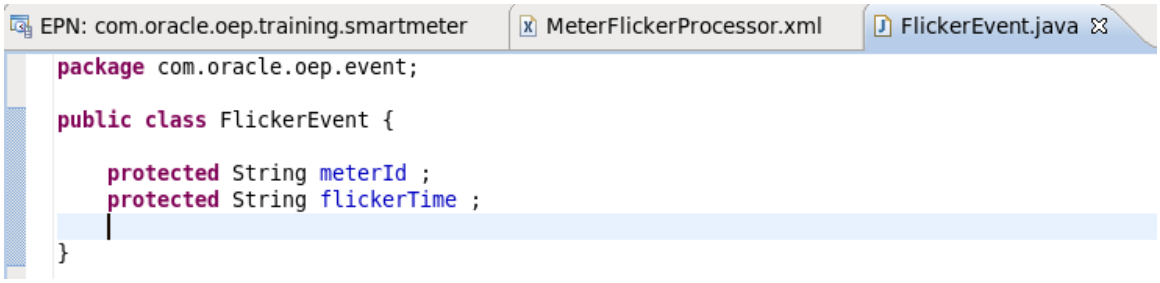
1. Create a new package by clicking on the "src" folder and using right-click to choose "New", "Package". Give it the name: "com.oracle.oep.event". Click "Finish".



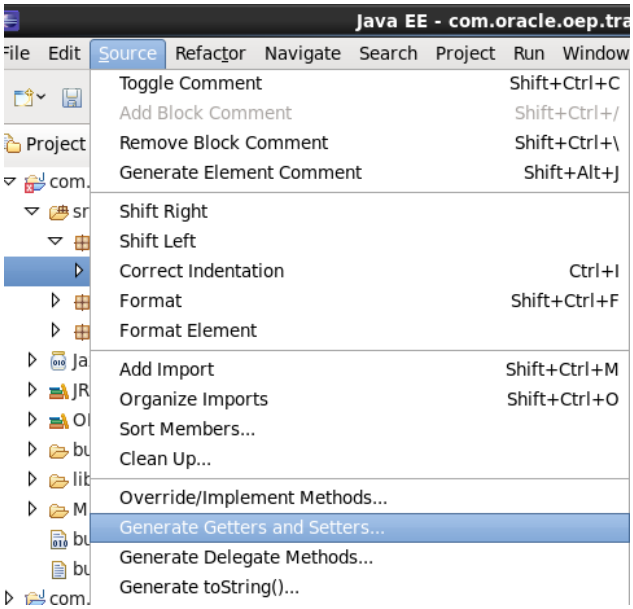
2. Create a new class in that package by right-clicking on the package name and selecting "New", "Class". Name the class "FlickerEvent". Click "Finish".



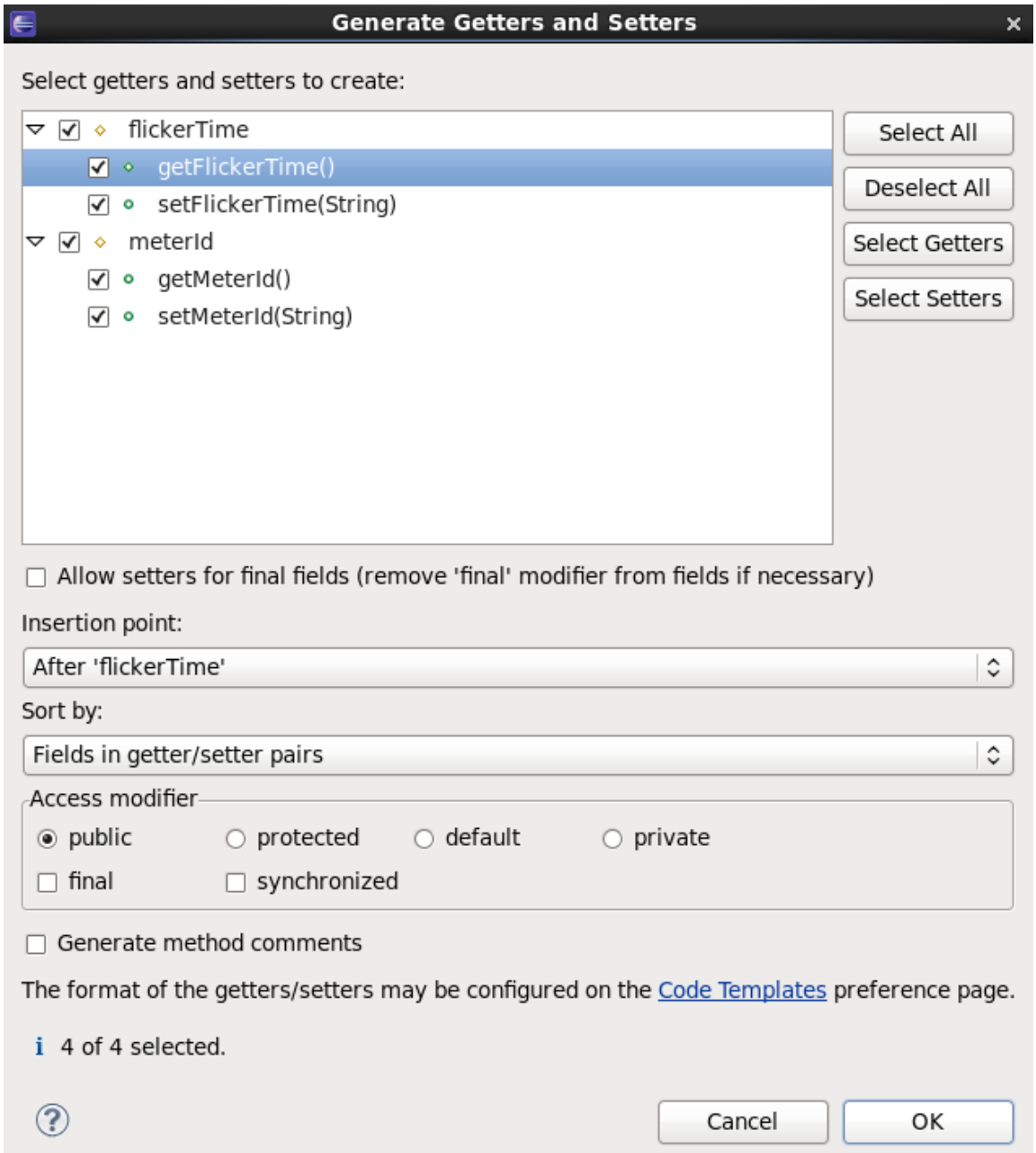
3. The empty class will open in the main screen. You can add properties to this class by typing the 2 variables as shown in the picture below:



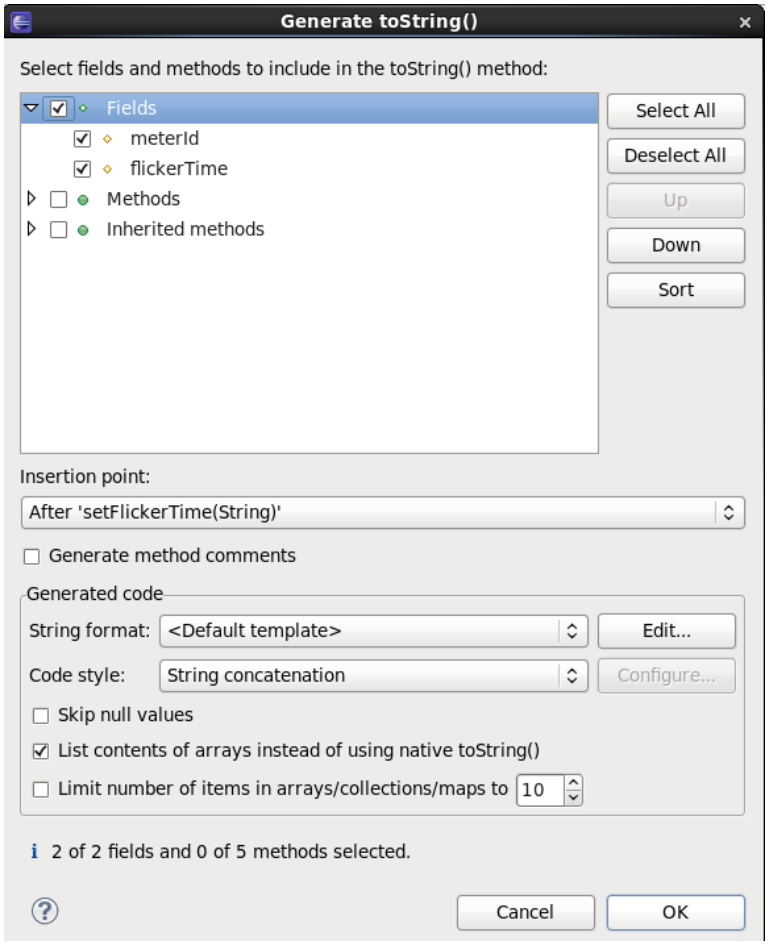
4. You can easily add get and set methods for these properties by right-clicking below the properties and selecting “Source”, “Generate Getters and Setters”.

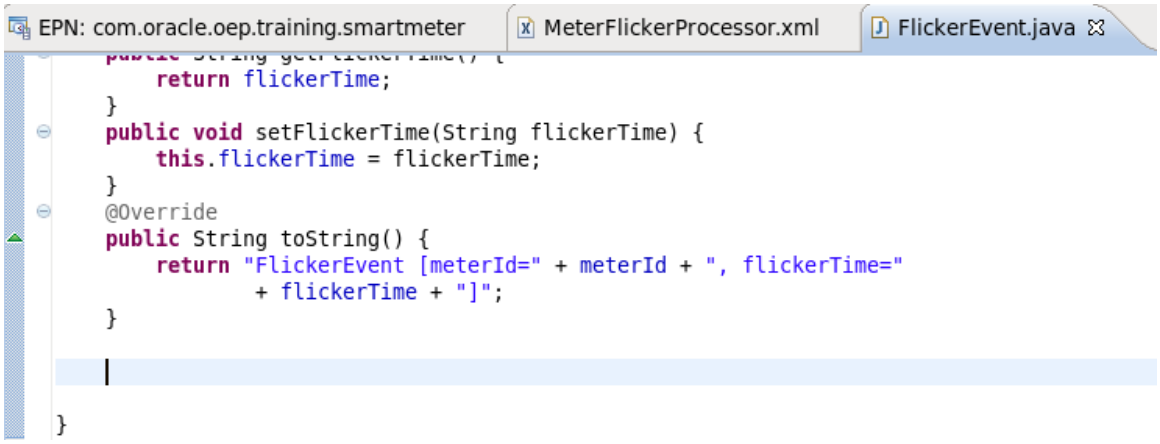


5. Make sure that all of the boxes are checked. (You can use the "Select All" button) and Click OK. The "Getter" and "Setter" source code will be generated for you. Use CTRL-S or the disk icon in the menu bar to save the changes.



6. From the "Source" menu generate a "toString()" method in your new event.





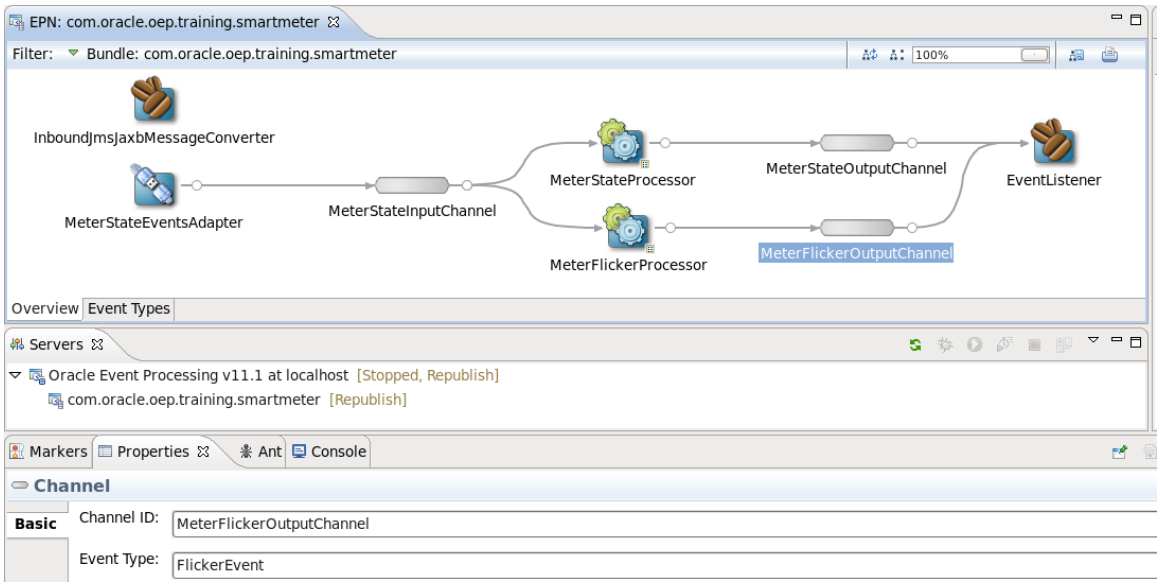
7. In the "META-INF/spring" folder, double-click the "com.oracle.oep.training.smartmeter.context.xml" file to open it.

8. Add the newly created event to the event type repository. You can '*cut-and-paste*' the previous event that you added and make the necessary changes. Notice that the package name is different.

```
    <wlevs:event-type-repository>
        <wlevs:event-type type-name="MeterStateEvent">
            <wlevs:class>com.oracle.oep.demo.jaxb.event.MeterStateEvent</wlevs:class>
        </wlevs:event-type>
        <wlevs:event-type type-name="FlickerEvent">
            <wlevs:class>com.oracle.oep.event.FlickerEvent</wlevs:class>
        </wlevs:event-type>
    </wlevs:event-type-repository>
```

9. Add the "FlickerEvent" to the "MeterFlickerOutputChannel". Remember that you can use CTRL + SPACE_BAR in the IDE to enter text instead of typing. (OR click on the channel and use the properties tab).

```
    <wlevs:processor id="MeterFlickerProcessor">
        <wlevs:listener ref="MeterFlickerOutputChannel" />
    </wlevs:processor>
    <wlevs:channel id="MeterFlickerOutputChannel" event-type="FlickerEvent">
        <wlevs:listener ref="bean" />
    </wlevs:channel>
</beans>
```



Part 3: Enhance the Event Bean

To see the output of your CQL query with the new event type, enhance the "Event Listener".

1. Right-click on the "Event Listener" icon and select "Go to Java Source".

2. Import the new event.

EPN: com.oracle.oep.training.smartmeter | *EventListener.java

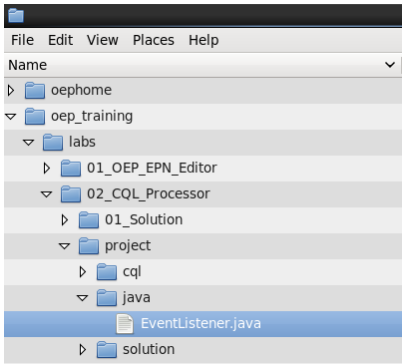
```
package com.oracle.oep.training.listeners;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.bea.wlevs.ede.api.EventRejectedException;
import com.bea.wlevs.ede.api.StreamSink;

import com.oracle.oep.demo.jaxb.event.MeterStateEvent;
import com.oracle.oep.event.FlickerEvent;

public class EventListener implements StreamSink {
```

3. Add some lines to show the details of the "FlickerEvent" when it arrives.

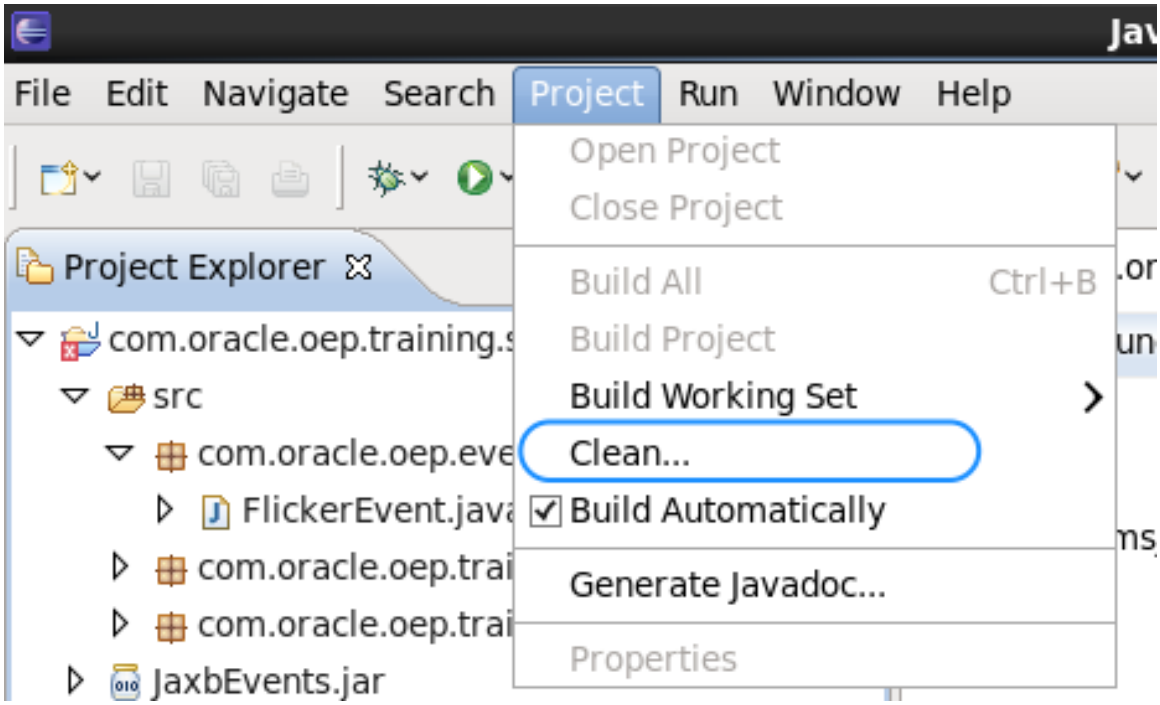
EPN: com.oracle.oep.training.smartmeter | EventListener.java

```
    public void onInsertEvent(Object event) throws EventRejectedException {
        log_.debug("onInsertEvent():" + event);

        if(event instanceof MeterStateEvent) {
            MeterStateEvent theEvent = (MeterStateEvent) event;
            String id = theEvent.getMeterId();
            String state = theEvent.getState();
            String transformer = theEvent.getTransformerId();
            String cust = theEvent.getCustomerType();
            //log_.info("EventListener: Received 'MeterStateEvent' for meter id: " + id + " state:" + state + "\n");
            System.out.println("EventListener: Received 'MeterStateEvent' for meter id: " + id + " state: " + state + " transf
        }

        if(event instanceof FlickerEvent) {
            FlickerEvent theEvent = (FlickerEvent) event;
            System.out.println(theEvent.toString());
        }

    }
```

OR: If you want, you can replace the "EventListener.java" file with the file in "<OEP_TRAINING\labs\02_CQL_Processor\java"

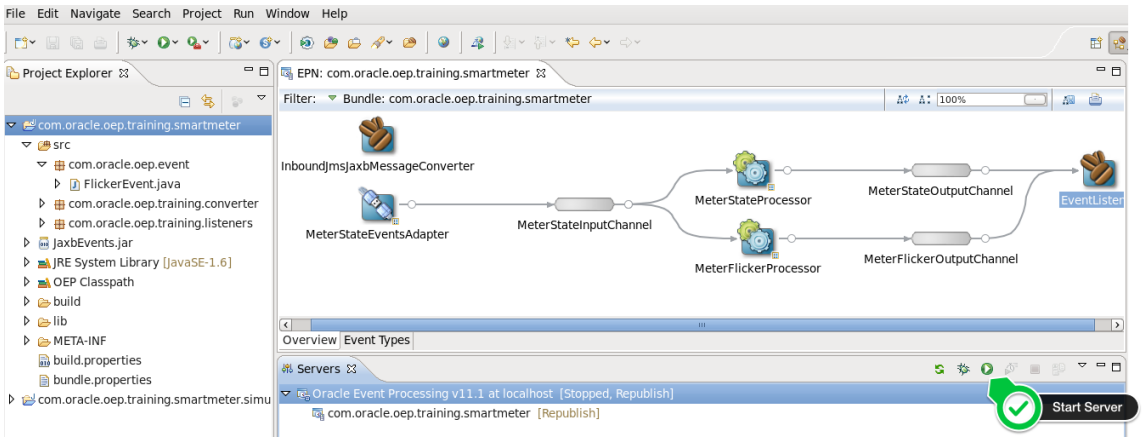


Choose "Clean" from the project menu. Notice that "Build Automatically" is checked. This will make sure your project is compiled properly before you deploy.

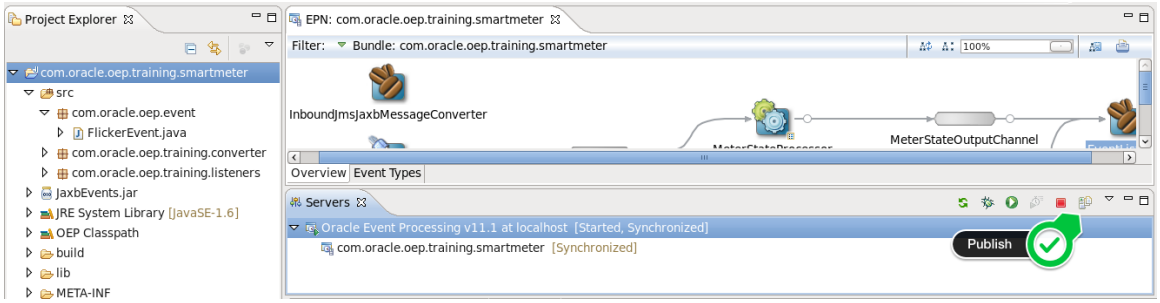


Part 4: Deploying the Application

Finally, you are ready to deploy the application to the server.
Start the server and add the project, if necessary.



1. If the application is already on a running server, you simply need to "publish" the new version. If there are no errors, you should see a "deployed successfully" and "started successfully" messages when the application is added to the server and the publish task is complete (with no more error messages):

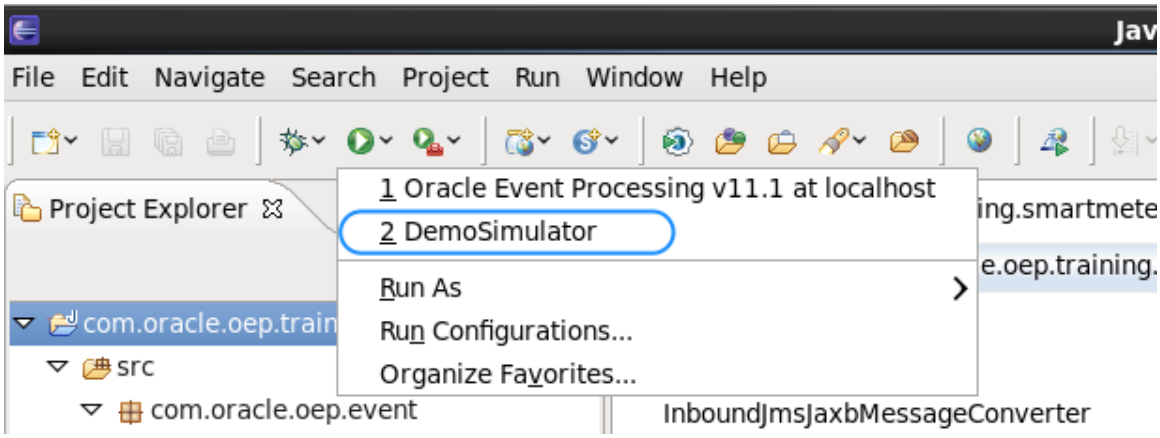



```
> <Notice> <Deployment> <BEA-2045000> <The application bundle "com.oracle.oep.training.smartmeter" was deployed successfully to
> <Notice> <Spring> <BEA-2047000> <The application context for "com.oracle.oep.training.smartmeter" was started successfully>
> <Notice> <Server> <BEA-2046000> <Server STARTED>
> <Notice> <Spring> <BEA-2047001> <The application context "com.oracle.oep.training.smartmeter" was shutdown successfully>
54 PM] Socket: WL-000402: There are: 3 active sockets, but the maximum number of socket reader threads allowed by the configurat
> <Notice> <Spring> <BEA-2047000> <The application context for "com.oracle.oep.training.smartmeter" was started successfully>
```

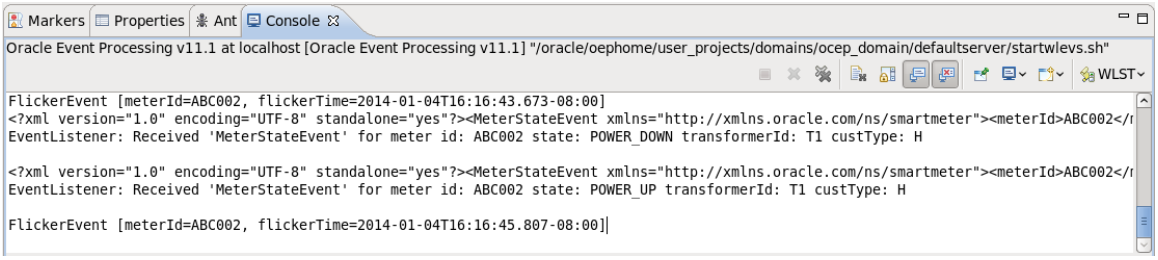
Part 5: Testing

You should now be ready to run the application and test that the query is working.
To test, we simply run the JMS simulator.

1. You simply need to run the JMS simulator that you ran in the previous tutorial. Since you've run the "DemoSimulator", you can run it from the green arrow in the menu bar.



When you run the application, you'll see that when a "POWER_UP" follows a "POWER_DOWN" for the same meterId (because we partitioned by meter id), you'll receive a "FlickerEvent".



Summary:

At the end of these exercises the participants understand how to:

- Create a CQL Processor
- Create a new event type that originates from the CQL processor