Oracle Complex Event Processing: Tutorial: Building an advanced EDA Application

An Oracle Tutorial
Updated June 2009
# TABLE OF CONTENTS

INTRODUCTION .............................................................................................................. 3
  Starting the BAM WLS Servers .................................................................................. 4
Advanced Topics Tutorial: Configuring Adapters ............................................................. 6
  Smart Meter Demo Application Key Points ............................................................... 6
  Environment Setup ..................................................................................................... 8
  Run the Smart Meter demo application ................................................................ 8
  Replace the placeholder with a JMS adapter instance .......................................... 9
  Configure the JMS adapter instance ..................................................................... 10
  Redeploy and run the application ....................................................................... 14
Advanced Topics Tutorial: CQL ...................................................................................... 15
  Scenario Overview .................................................................................................. 15
  Tutorial Steps .......................................................................................................... 16
INTRODUCTION

This tutorial accompanies the “TutorialSmartmeter.zip” Eclipse project archive which you must first import into your Eclipse IDE.

For the foundational understanding of the architecture, creation, deployment and testing of an EDA (Event Driven Architecture) based CEP (Complex Event Processing) application this tutorial includes a sample “SMART METER” solution, referenced during various sessions.

During the course of the 2 day training we will be adding a JMS adapter and CQL statements to a processor for this solution, which will subsequently highlight a powerful integration with Oracle BAM. (NOTE: You must have Oracle BAM 11g installed to use the EMS integration methods highlighted.)
Starting the BAM WLS Servers

To use BAM, you must first start a WebLogic Admin Server instance.

When you see the SERVER RUNNING message, start a WebLogic Managed Server instance.

NOTE: If the Oracle XE DB is not running, errors such as “TOPLINK.....” may appear. If this happens, then CTRL “C” both Servers and ensure the DB services are started (restarted).

Oracle Complex Event Processing – Smartmeter Tutorial
© Copyright Oracle Corporation, 2009
Both WLS Servers indicate a “ready” state, when you see the “RUNNING” mode console message.
Advanced Topics Tutorial: Configuring Adapters

The ability for Oracle CEP to connect to the various types of data sources is a major requirement in all customer implementations. The following tutorial gives you the insight and basics on how to configure and use an EPN Adapter node.

Review the following key points and then work on the tutorial, described later in this document.

The foundation solution for our tutorial is the Smart Meter demo application (oracle.cep.demo.smartmeter-lab).

Smart Meter Demo Application Key Points

The smart meter demo application (the application) consumes events that reflect real-time meter readings, substation voltage, and substation temperature values. The application first joins the meter events to a cache to add contextual data to the meter readings. One of the properties retrieved from the cache is the substation that the meter is attached to. This allows aggregation of the meter readings by substation.

Those aggregate values (by substation) are joined to a cache containing substation baseline voltage and temperature levels. The aggregate view of the substation is then joined to incoming substation temperature and voltage values to provide a consolidated view of substations that is then forwarded on to BAM.

For the purposes of the demo, the data feeds are simulated using the Oracle CEP load generator utility.

The following image describes the smart meter event processing network:
In the demo application, a single CSV file includes meter, temperature and voltage events. This approach allows the order of events to be controlled. Following is a fragment of the CSV file. The properties (in order) are eventType, eventKey, eventValue:

<table>
<thead>
<tr>
<th>eventType</th>
<th>eventKey</th>
<th>eventValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>hr</td>
<td>h1</td>
<td>22</td>
</tr>
<tr>
<td>hr</td>
<td>h2</td>
<td>26</td>
</tr>
<tr>
<td>hr</td>
<td>h3</td>
<td>19</td>
</tr>
<tr>
<td>st</td>
<td>Danville</td>
<td>95</td>
</tr>
<tr>
<td>st</td>
<td>Concord</td>
<td>94</td>
</tr>
<tr>
<td>sv</td>
<td>Danville</td>
<td>7200</td>
</tr>
<tr>
<td>sv</td>
<td>Concord</td>
<td>7200</td>
</tr>
</tbody>
</table>

Within the application, the EventMapper event-bean receives the event stream from the CSV adapter and uses the first property (eventType) of each CSV event to determine the appropriate event type; then creates and sends that event into the EPN.

Event channels in 11g carry one specific type of event. In cases where an adapter produces more than one type of event, a channel selector utility is used to route each event to the correct channel based on the event type defined in each down-stream channel’s event-type property. In this app, the channel selector routes the meter reading, temperature, and voltage events to the correct channel.

In the baseline version of the demo there is a placeholder event sink that represents where events would be output to BAM via JMS. The placeholder is a simple JavaBean that receives and (optionally) prints each event.

The goal of the lab is to replace that placeholder with an outbound JMS adapter instance that is then configured to send JMS Map messages to BAM.
Environment Setup

1. IMPORT The "oracle.cep.demo.smartmeter-lab.zip" IDE archive into your Eclipse workspace.

2. Ensure that the file reference.xml located in the reference-data folder of your Eclipse project is in the root of the server’s directory. The default location is: <installdir>\ user_projects\domains\ocep_domain\defaultserver

3. Ensure that the loadgenerator files in the loadgen folder of your Eclipse project are in the loadgen directory. The default location is: <installdir>\ ocep_11.1\utils\load-generator

FILES: smartmeter.prop, smartmeter.csv

Run the Smart Meter demo application

1. Start the local development server using the green start button or context menu from the “Servers” view.

2. Ensure that the application is deployed to the local development server. Open a command window in the loadgen directory and start the load generator using the following command: runloadgen.cmd smartmeter.prop

DOCUMENT RESULTS FOR A TUTORIAL BASELINE
(Review the Output messages provided in the IDE Console Window)

3. Stop the load generators by typing Ctrl-C in the command window.
Replace the placeholder with a JMS adapter instance

Use the EPN view to navigate to the placeholder adapter. Right-click on the JmsOutboundMessage node and select the “Go to Assembly Source” item.

Once in the Assembly file, notice that adapter declaration does not reference the built-in JMS adapter. The adapter declares an instance of a Java class that is an Event Sink and prints each received event to the console. To enable JMS, we need to:

- Comment out the existing adapter declaration
- Create a new <wlevs:adapter> element with the same name (id), but with a jms-outbound provider attribute
- Create and reference a <bean> element that declares the outbound message converter for the JMS adapter.

Comment out the first <wlevs:adapter> element:

```
<--
   Comment out to enable JMS.
   This listener sinks the outbound events
--><wlevs:adapter
  id="JmsOutboundMapMessages"
  class="com.oracle.cep.demo.smartmeter.listener.EventTraceListener"
  <wlevs:instance-property name="enabled" value="false"/>
</wlevs:adapter>
```

Add a <wlevs:adapter> element that specifies the “jms-outbound” provider, and a <bean> element that specifies the outbound message converter

```
<-- Add the following content to enable JMS -->
<wlevs:adapter id="JmsOutboundMapMessages" provider="jms-outbound">
  <wlevs:instance-property
       name="converterBean" ref="EventToJmsMapMessageConverter" />
</wlevs:adapter>
<bean id="EventToJmsMapMessageConverter"
     class="com.oracle.cep.demo.smartmeter.converter.EventToJmsMapMessageConverter" />
```
At this point, the adapter is now declared as an outbound JMS adapter. However, there are no configuration properties bound to the adapter. This can be verified by right-clicking on the adapter in the EPN view and noticing that the “Go to configuration source” option is not enabled. The next step is to provide the required configuration for the JMS adapter.

**Configure the JMS adapter instance**

Use the Project Explorer view in the IDE and open the `JmsAdapterConfiguration.xml` file in the META-INF/wlevs folder.
Note that JmsAdapterConfiguration.xml has the <jms-adapter> element commented out. **Uncomment** the jms-adapter element and take note of the configuration properties. These properties define the information needed to connect to the JMS provider and to locate the required JMS resources. The complete set of properties and default values can be found in the on-line product documentation.

If needed, change the value in the `<jndi-provider-url>` to match the host and port of the server where BAM is running.

```xml
<%!  
<jms-adapter>
  <name>JmsOutboundMapMessages</name>
  <jndi-provider-url>t3://localhost:9001</jndi-provider-url>
  <jndi-factory>weblogic.jndi.WLInitialContextFactory</jndi-factory>
  <connection-jndi-name>.jms/QueueConnectionFactory</connection-jndi-name>
  <destination-jndi-name>jms/demoQueue</destination-jndi-name>
  <user>weblogic</user>
  <password>Oracle11</password>
  <session-transacted>false</session-transacted>
</jms-adapter>
-->
```

Verify that the adapter configuration data is “linked” to the adapter instance.

Hold down the Ctrl key and hover over the JmsOutboundMapMessages text within the `<name>` element. You should see the pointer change to a hyperlink. If you left-click, you will navigate to the adapter definition in the assembly file.

The linking works in both directions. From the assembly file, hold down the Ctrl key and select the adapter name. You will navigate back to the configuration element for the adapter.

The linking to the configuration view is also supported from the EPN view. Note that the JmsOutboundMapMessages icon now includes a small badge on the lower right side of the icon. This indicates that that element of the EPN has configuration data. The context-menu now includes options for both “Go to Assembly Source” and “Go to Configuration Source”.

Review the implementation of the outbound message conversions.

The outbound JMS adapter will provide automatic conversion of Map based events to a JMS MapMessage. However if your application uses JavaBean based events, you must provide a conversion of outbound events to the desired JMS Message type. This is accomplished by implementing the OutboundMessageConverter API:

Oracle Complex Event Processing – Smartmeter Tutorial
© Copyright Oracle Corporation, 2009
public List<Message> convert(Session jmsSession, Object event) throws MessageConverterException, JMSException {

There are two Java types in the com.oracle.cep.demo.smartmeter.converter package:

- EventToJmsMapMessageConverter – implements the OutboundMessageConverter for each JavaBean based event type.
- OutboundEventTypes – Surfaces the JMS property values used to identify the event type represented by each outbound message. This is to enable the use of a single JMS destination with selectors used on the receiving side to identify the type of message.

```java
@SuppressWarnings("unchecked")
private List handleEvent(Session jmsSession, MeterReading event) ... {
    List result = new ArrayList();
    MapMessage msg = createMessage(jmsSession, OutboundEventTypes.METER_READING);
    msg.setObject("timestamp", dateTimeFormatter_.format(event.getDate()));
    msg.setObject("house", event.getHouse());
    msg.setObject("kwh", event.getKwh());
    result.add(msg);
    return result;
}

private MapMessage createMessage(Session jmsSession, String eventType) {...
    MapMessage msg = jmsSession.createMapMessage();
    msg.setStringProperty(OutboundEventTypes.EVENT_TYPE_PROPERTY_NAME, eventType);
    return msg;
}
```
Review the EMS configuration in BAM to see the configuration used to route each message to the correct DataObject. (SEE Section below on Starting the BAM WLS Servers)
Redeploy and run the application

Ensure that BAM is running, then redeploy the application to the server. Check the CEP server console output to ensure that there are no deployment errors (SEE Section below on Starting the BAM WLS Servers)

Restart the loadgenerator (using the same configuration and data files) and note that the BAM dashboard is again receiving data from CEP.
Advanced Topics Tutorial: CQL

Scenario Overview

The demo application joins incoming meter read events to a cache to fetch the substation associated with the meter. As with relational database, if there’s no matching value in the cache, the incoming event will not join and will be discarded.

In some cases that may be the desired result. However for the purposes of this lab, we assume that we want to identify any meter events that do not have a match in the cache, and send those meter events out on a separate stream (i.e. we don’t want to lose them).

In this lab we will make use of the “outer join” support in CQL to ensure that all events are retained. We will then create a new query that identifies meter events with no substation (i.e. substation IS NULL) and output those events to a new channel. One could envisage that events on this channel would trigger a business process to determine why the lookup data was not available.

As is often the case, this new functionality, can be implemented, as an extension to the existing EPN. The following view of the EPN highlights the additional path out of the existing JoinToCustomerCache process that will capture meter events with no match in the cache:
Tutorial Steps

Add (create) the *MeterWithNoSubstation* channel

Use the EPN view right-click context menu to create a new channel. Name the channel: *MeterWithNoSubstation*.

**NOTE:** DBL-Click on the EPN TAB to expand the view for “ease of use” Editing.

Right-click on the newly created channel (*MeterWithNoSubstation*) and select “Go to Assembly Source” option from the EPN context menu. The assembly source will have a simple declaration of the channel:

```
<wlevs:channel id="MeterWithNoSubstation"></wlevs:channel>
```

Update the `<wlevs:channel>` element to create and then set the event-type property to *MeterReading*. The IDE provides content assist for the available XML attribute names and for the set of event types in the event-type attribute.

```
<wlevs:channel id="MeterWithNoSubstation" event-type="MeterReading">
</wlevs:channel>
```

Create a nested `<bean>` element that declares an EventTraceListener instance as a listener on the channel. Note the nesting of the elements using the following example:

```
<wlevs:channel id="MeterWithNoSubstation" event-type="MeterReading">
  <wlevs:listener>
    <bean class="com.oracle.cep.demo.smartmeter.listener.EventTraceListener">
      <property name="logCategory" value="SmartMeter.MeterWithNoSubstation"/>
    </bean>
  </wlevs:listener>
</wlevs:channel>
```
Link the JoinToCustomerCache processor to the new MeterWithNoSubstation channel.

Select the JoinToCustomerCache processor. Hold the left-click button and move the mouse toward the MeterWithNoSubstation channel. You should see the end of the arrow change to a connector. Release the left mouse button when the connector is over the MeterWithNoSubstation channel. This will create the link in the EPN that connects the JoinToCustomerCache processor as an Event Source for the MeterWithNoSubstation channel. The following XML fragment shows the addition of the MeterWithNoSubstation channel as a listener, which is dynamically created.

```
<wlevs:processor id="JoinToCustomerCache" provider="cql">
  <wlevs:listener ref="HouseEvents" />
  <wlevs:listener ref="MeterWithNoSubstation" />
  <wlevs:cache-source ref="CustomerDataCache" />
</wlevs:processor>
```

Update the JoinToCustomerCache processor to route events with no substation to the MeterWithNoSubstation channel.

Return to the EPN view. Double-click on the JoinToCustomerCache processor to navigate to the source view. Note that the processor includes three elements:

- JoinToCustomerCacheView – this performs an outer join of the meter read events to the cache
- MeterWithSubstationQuery – this query selects only those meter read events that had a join in the cache
- Channel element mapping the output events from the MeterWithSubstationQuery to the HouseEvents channel

There are two new elements needed in the processor to route events with no substation match to the new channel:

Add the MeterWithoutSubstationQuery to select those event where the substation property was null (i.e. there was no match in the cache). The source for this query is the existing view JoinToCustomerCacheView.

```
<query id="MeterWithoutSubstationQuery">
<![CDATA[
  SELECT
date,
  name AS house,
kwh
  FROM
  JoinToCustomerCacheView
  WHERE
  substation IS NULL
]]>
```
The above query selects the properties needed to populate the `MeterReading` event. However since a channel only supports a single event type. We also need to add a new channel selector to route the output of this new query to the `MeterWithNoSubstation` channel we created in a prior step.

Navigate to the end of the `<processor>` element and notice the existing channel mapping for the `<MeterWithSubstationQuery>` element. Note that the `<selector>` element references the id of the query.

```
<channel>
  <name>HouseEvents</name>
  <selector>MeterWithSubstationQuery</selector>
  <heartbeat>500000000</heartbeat>
</channel>
```

Add a new `<channel>` element that routes the output of `<MeterWithoutSubstationQuery>` to the `<MeterWithNoSubstation>` channel. Note that the IDE provides completion for the set of available channels.

```
<channel>
  <name>MeterWithNoSubstation</name>
  <selector>MeterWithoutSubstationQuery</selector>
</channel>
```

Redeploy the updated application and re-start the load generator.

The existing data file for the demo app includes an entry that does not have a match in cache. Prior to the changes made in this lab, that event was discarded. When you observe the console output messages, you should now see a message from the `EventTraceListener` showing that the event for `h99` did not have a match. The fact that we’ve printed this event indicates that it’s been captured and can be processed as needed by any Event Sink that’s attached to the channel.

```
smartmeter.csv
hr,h1,22
hr,h2,26
hr,h3,19
hr,h4,29
hr,h99,99
hr,h5,30
st,Danville,95
st,Concord,94
sv,Danville,7200
```

**console output**

```
<SmartMeter.MeterReading> <onInsertEvent()MeterReading [house:h5][kwh:30]>
<SmartMeter.MeterWithNoSubstation> <onInsertEvent()MeterReading [house:h99][kwh:99]>
<SmartMeter.SubstationTemperature> <onInsertEvent()SubstationTemperature [substation:Danville][temperature:95]>
```