duckcreek technologies

Exercise: Connectors

This lab will involve creating a custom connector that will simulate calling a rating service to generate a premium for the Policy Summary page. Getting a rating or premium is a common usage for connectors.

Exercise 1: Create a skeleton custom connector implementation and plug it into the transaction.

1. Create a new class called **com.agencyport.workerscomp.connector.RatingServiceConnector** class that extends ConnectorManager[1].

2. For the execute method, return Outcome.CONDITIONS_NOT_MET. In the body of this method, add a simple "Hello, World" INFO message to the messageMap via `this.getMessageMap()`.

3. For the postProcess method, return true.

4. Open the `/web/WEB-INF/workerscomp/definitions/workerscomp.xml` transaction. We will be working with the page with the id "policySummary". Create a `connectors` element with a `type="display"`. Under the `connectors` element, add a `connector` element with the name attribute set to "RatingServiceConnector" and a classname attribute referencing the class we just created in the first step.

```
<page id="policySummary" title="Policy Summary" type="page"
validateTransactionOnDisplay="true">
    <!-- SDKCLASS Connector Lab -->
    <!-- Add the connector invocation of (simulated) rating service -->
    <connectors type="display">
      <connector type="custom"
className="com.agencyport.workerscomp.connector.RatingServiceConnector" />
    </connectors>
</page>
```

5. Deploy the updates. Open a workitem and choose the Policy Summary page in the menu. The connector should be executed when the page is displayed. You should see your "Hello, World" message at the top of the page.

[1] Note that there are 2 – use the one within the following package: com.agencyport.connector.impl

duckcreek technologies

Exercise 2: Return a sample premium and display it on the Policy Summary page

1. In the RatingServiceConnector execute method, put an Integer into the ConnectorDataBundle that is a Java Integer instance; for now, use a hard-coded numeric value for the actual premium. Use "RATING_PREMIUM" for the key. The DataBundle is a map, so this will look like this:

```
dataBundle.put("RATING_PREMIUM", new Integer(1500));
```

2. Modify the "Hello World" message created in Part I to show this premium instead.
3. Deploy the updates. Open a workitem and choose the Policy Summary page in the menu. You should see your premium value displayed in an info box at the top of the page.

Exercise 3: (Advanced and Optional): Simulate a premium calculation based on the workitem data

1. In the RatingServiceConnector execute method, get the workitem ACORD out of the ConnectorDataBundle using the getDataCollection call.

2. Get the sum of values for the Employer's Liability Limit. Note this is a "split" limit, so multiple xpaths must be used to get the values.

3. Compute the premium as 5% of the Employer's Liability Limit computed in the previous step. This is not intended to represent a realistic premium, but it gives us a way to vary the simulated premium based on the workitem data.

4. Change the storage of the RATING_PREMIUM to store the computed premium from the previous step, rather than the hard-coded value; you'll still use an Integer object to store the premium in the dataBundle.

5. Deploy the updates. Open a workitem and modify the selection for the Employer's Liability Limit; save that page. Jump to the Policy Summary page to see the new

computed premium. Change the Employer's Liability Limit selection to see a different premium.

Exercise 4: (Advanced and Optional): Use a View to read data

1. In the RatingServiceConnector execute method, comment out the existing code that reads the Employers Liability Limit from apData (as created in Part III). NOTE you'll still be computing the premium and storing in the dataBundle.

2. Write the equivalent code that instead uses ViewRepository.getView(<viewId>).read(apData …), reading the data using the view created for the Employer's Liability Limit in a prior lab. NOTE when reading the data in this way, the result will be in the form of the data value from the underlying Employer's Liability Limit optionList.

3. Update the premium computation code to use the split limit values acquired from the view.

4. Deploy the updates. Open a workitem and modify the selection for the Employer's Liability Limit; save that page. Jump to the Policy Summary page to see the new computed premium. Change the Employer's Liability Limit selection to see a different premium.

What are some of the advantages and disadvantages of using the view to acquire the data?

We've now covered the basic operations of any Connector:

- The connector is wired in to the transaction as part of the page definition, either in the transaction file (for custom pages) or in the pageLibrary (for dataEntry or roster pages).
- The connector implementation does "work" in the execute method. The postProcess method is traditionally used to update the messageMap or otherwise indicate the result of the work.
- The ConnectorDataBundle is used to both retrieve information about the current context, such as the workitem ACORD, and return information for use in downstream code.

NOTE: In the Custom JSP lab, we'll display the premium in the body of the Policy Summary page, rather than as an info header message.