ADF Code Corner

89. How-to conditionally switch model driven LOV in ADF forms and tables

twitter.com/adfcodecorner

Abstract:

In Oracle ADF Business Components, developers can define model driven list of values for consistent data lists and UI rendering at runtime. In addition, model driven list-of-values can be configured to be dependent so they present a master-detail relationship of the list data.

A not so well known feature of model driven list-of-values is that the LOV definition can be switched at runtime, which usually is triggered by data in a related view object attribute.

This articles shows an example of a list of values that changes its data list from credit card types to currencies dependent on the selected payment mode, which can be CREDIT or CASH.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
10-AUG-2011

Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

Introduction

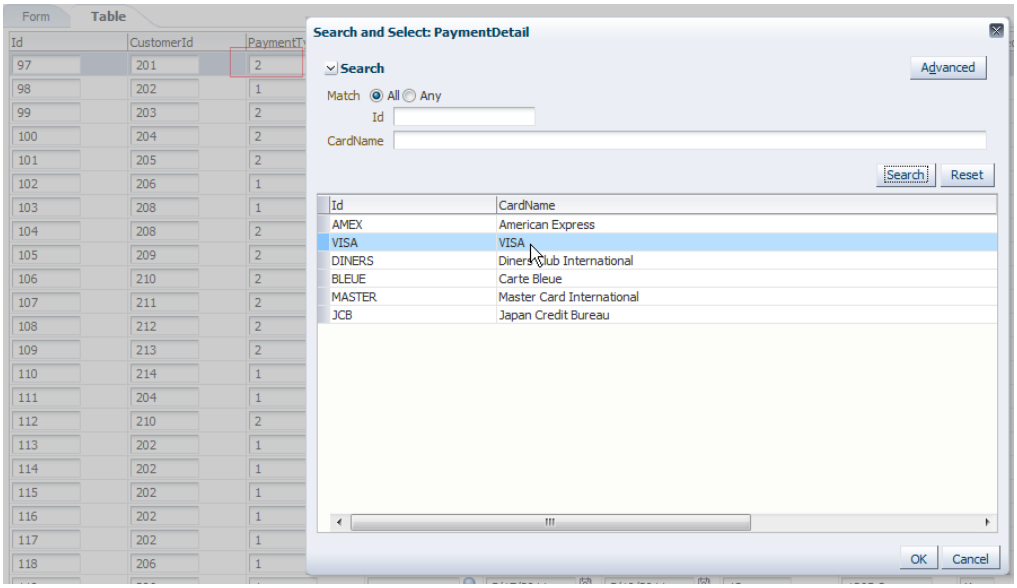
Some use cases require a list of values to show different list data based on an external condition. In the example created for this article, the condition is the value of the *PaymentId* attribute, which, if set to 1, means a customer pays cash, and, if set to 2, that a customer pays with credit card.

Dependent on the type of payment chosen for an order, the list of values shows a list of accepted currencies or a list of accepted credit cards to choose from. The database schema used in this sample is **ADF Summit**, which you can download from here:

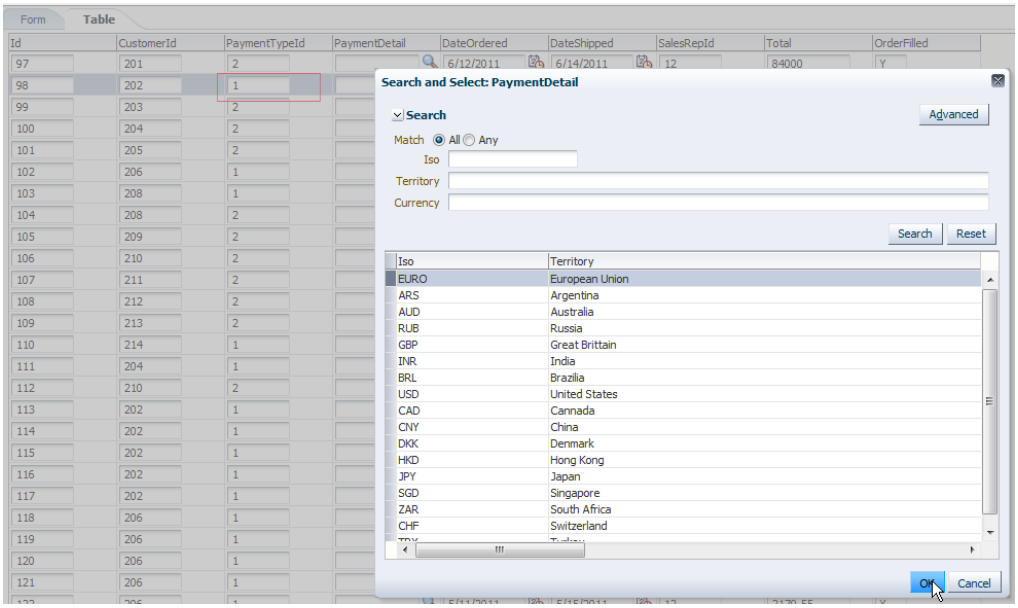
http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/SummitADF/SummitADFV1_0_08072011.zip

The ADF Summit schema can be installed on the Oracle XE and enterprise database.

The image below shows an order table with a list-of-values defined for the **PaymentType** column. The LOV data, as explained before, depends on the value of the **PaymentId** column. This means that the data – currencies or credit cards – displayed in the LOV changes on a row-by-row basis.



The image above shows the list-of-values for a **PaymentId** value of 2 (credit card). The image below shows the same LOV for a PaymentId value of 1 (cash). Instead of showing the payment id values, I could have used a single select one choice, which I did not for clarity reasons.



In the following, this article explains how the use case of varying data lists can be implemented declaratively using model driven list-of-values, which I think is a great proof point for how ADF simplifies web development, as the same use case would require many lines of code to be written in other programming languages or frameworks.

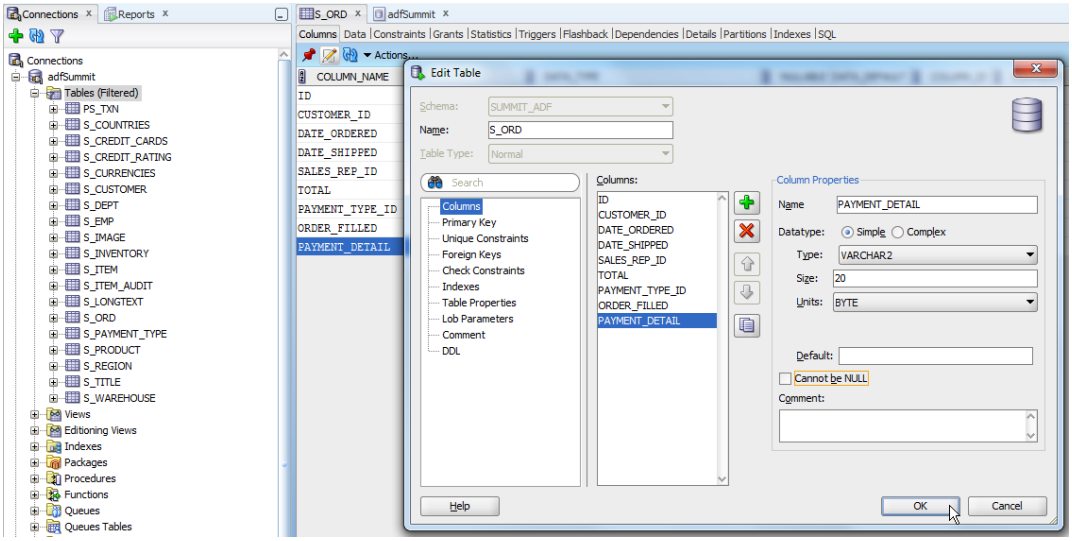
The Oracle JDeveloper 11.1.1.4 example workspace used in this article can be downloaded as sample 089 from the ADF Code Corner website:

http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html

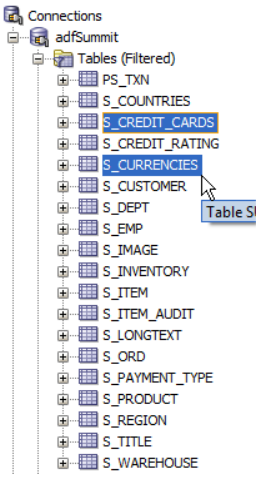
Database

The default **ADF Summit database schema** needs to be patched for this used article, for two tables, *S_CREDIT_CARDS* and *S_CURRENCIES* to be added. The *S_ORD* table that holds orders also needs to be enhanced with an additional *PAYMENT_DETAIL* column.

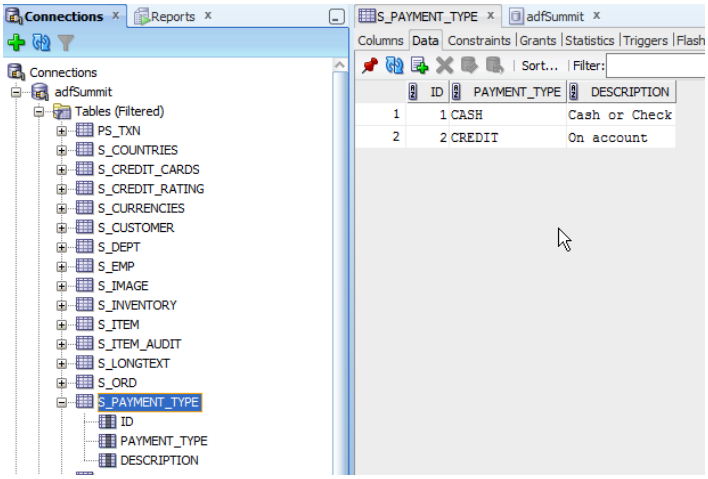
The sample download contains a SQL script that you can run against the **SUMMIT_ADF** schema to create these changes. The script is in the **SUMMIT_ADF_SCHEMA-PATCH** folder of the sample workspace.



The image below shows the ADF Summit schema with the patch script installed.

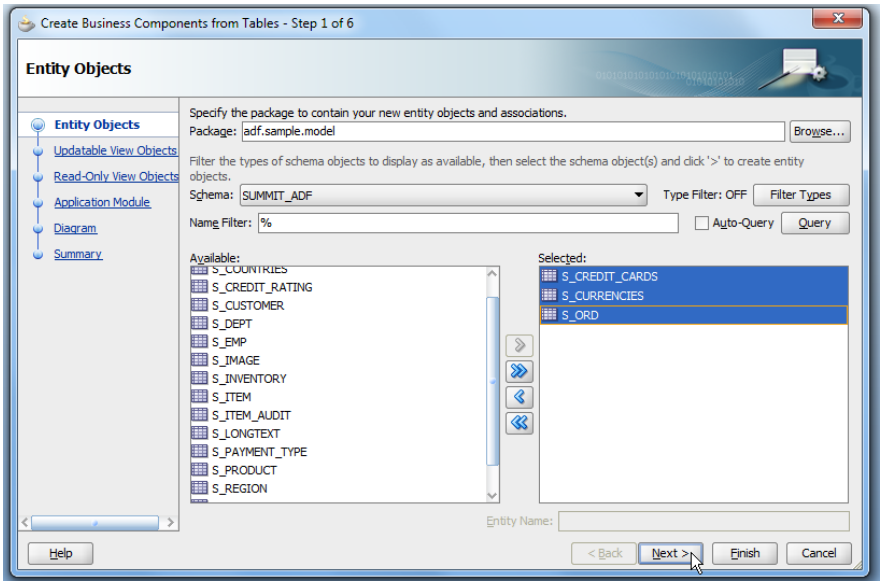


The ADF Summit *S_PAYMENT_TYPE* table knows of two values, CASH and CREDIT for which we need to create list of values for an ADF Business Components view object.



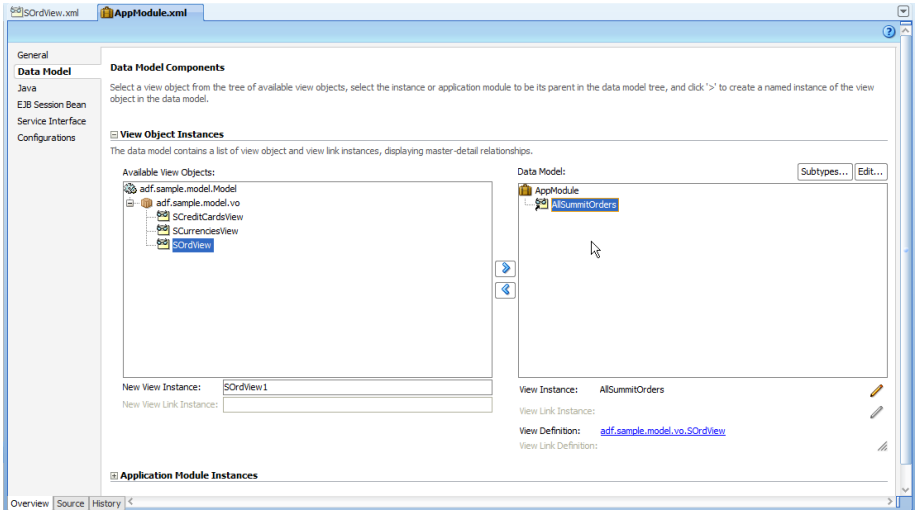
ADF BC Model

As shown in the image below, an ADF Business Components model needs to be created for the three tables *S_ORD*, *S_CURRENCIES* and *S_CREDIT_CARDS*. Only the *S_ORD* view object is later exposed to application developers. The other two entity objects and view objects are used internally to populate the LOV.

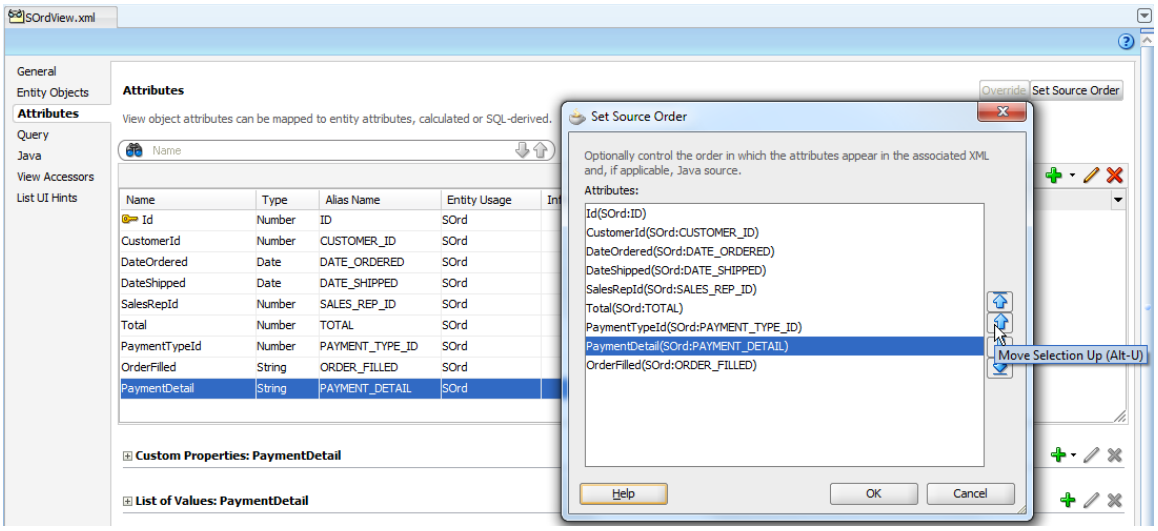


The ADF Business Component model is built choosing **New | Business Tier | ADF Business Components | Business Components from Tables** from the context menu that opened by a right mouse click onto the Model project for an application that was created using the Fusion Web Application (ADF) template. In the opened wizard, a database connection needs to be defined to access the ADF Summit schema - **summit_adf**.

After creating the entities, which is shown in the image above, view objects are created for each entity. The application module then is edited to only expose the *S_ORD* view object. The view object instance in the example also was renamed to **AllSummitOrders**.



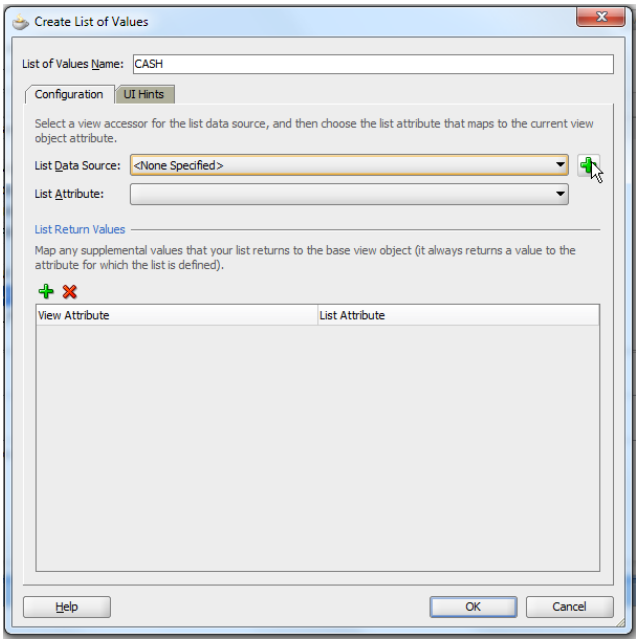
To create the model driven LOVs, the **SOrdView** view object is opened in the ADF Business Components editor, which is done by a double click on the view object definition in the JDeveloper Application Navigator.



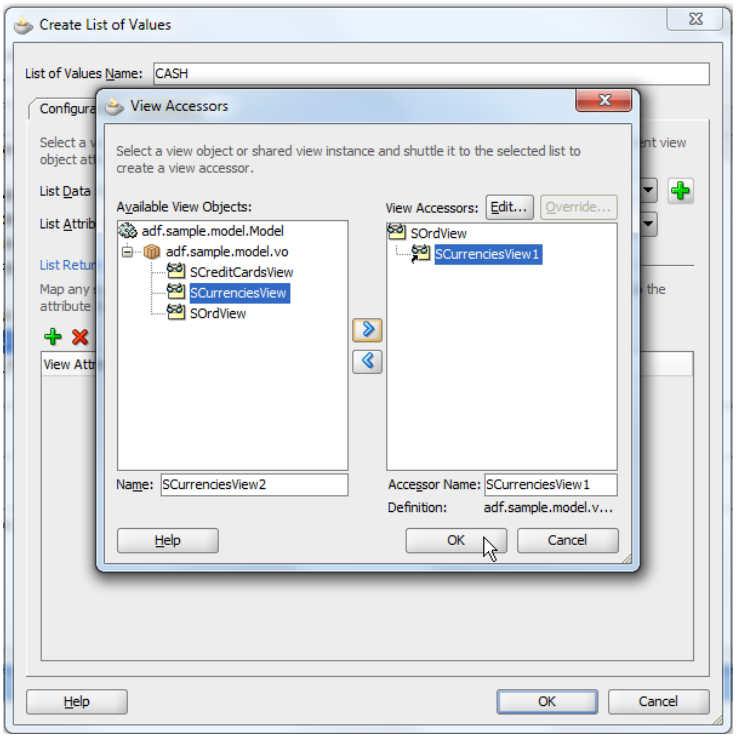
The image above shows how the order attributes in a generated view object can be changed using the **Set Source Order** button.

In the example, reordering is used to show the **PaymentTypeId** and the **PaymentDetail** columns next to each other.

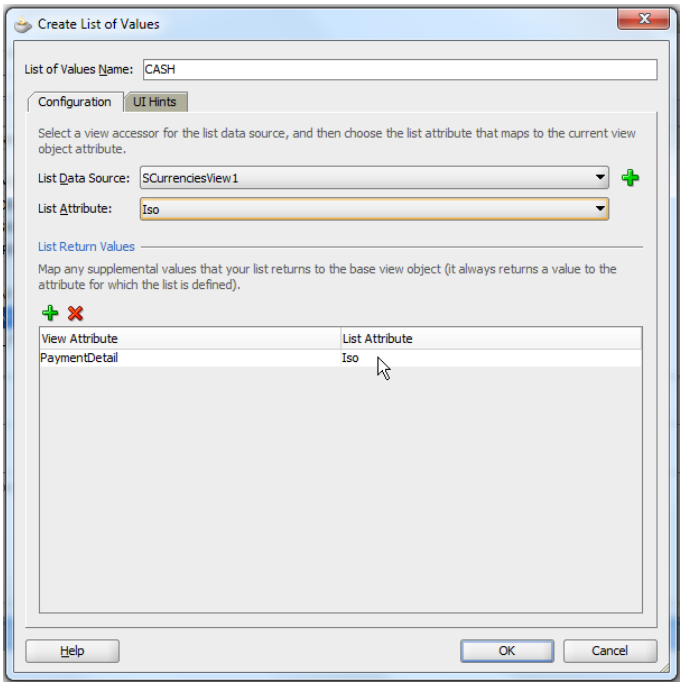
To create the list-of-values, the **PaymentDetail** attribute is selected and the green plus icon that is next to the **List of Values: PaymentDetail** label pressed. To define the model driven list the green plus icon next to **List Data Source** is pressed in the **Create List of Values** dialog.



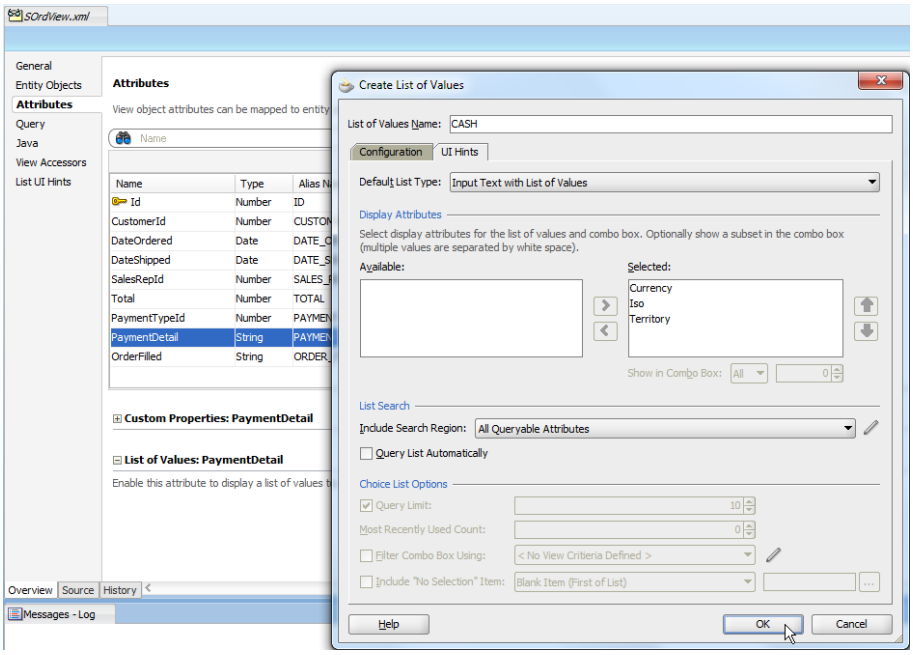
In the list of available view objects the **SCurrenciesView** is selected and moved to the list of selected **View Accessors**. This is how model driven list-of-values are created.



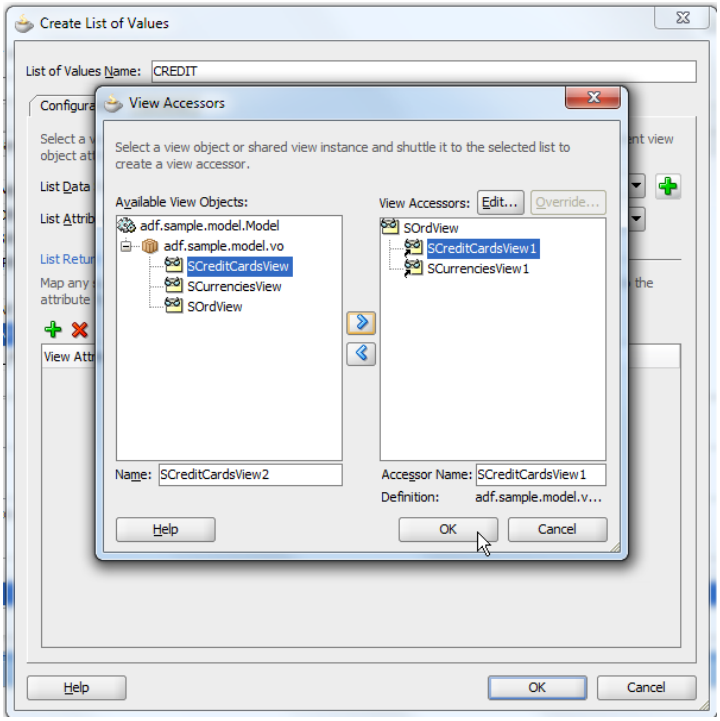
Next, as shown in the image below, the **PaymentDetail** attribute is mapped to the **Iso** attribute of the currencies view object. This ensures the currency ISO value of the selected list value is copied to the **PaymentDetail** attribute at runtime.



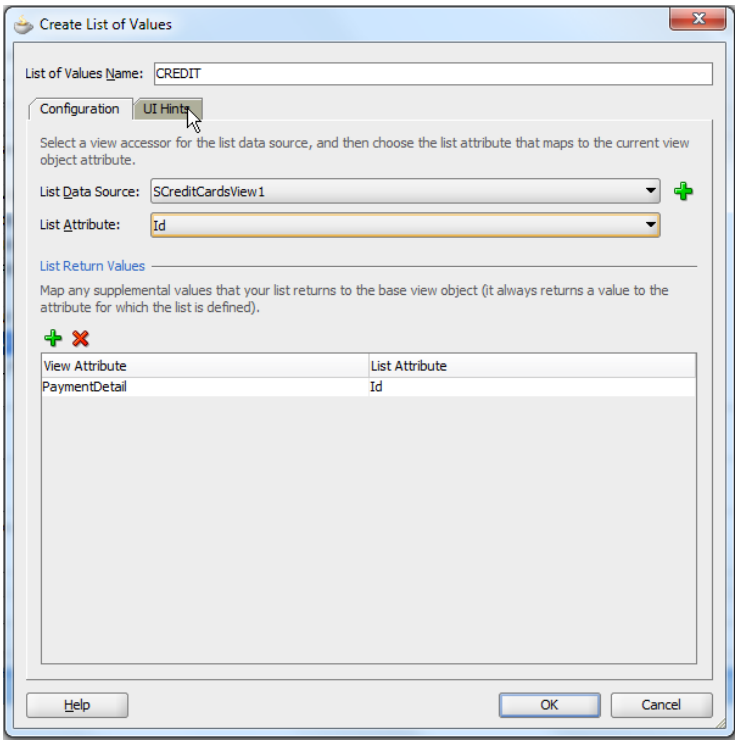
It is important to note that the **List of Values Name** is chosen to be **CASH** to match with the payment type name for payment type Id 1. This way we don't need to further transform the payment type value to the name of a matching LOV.



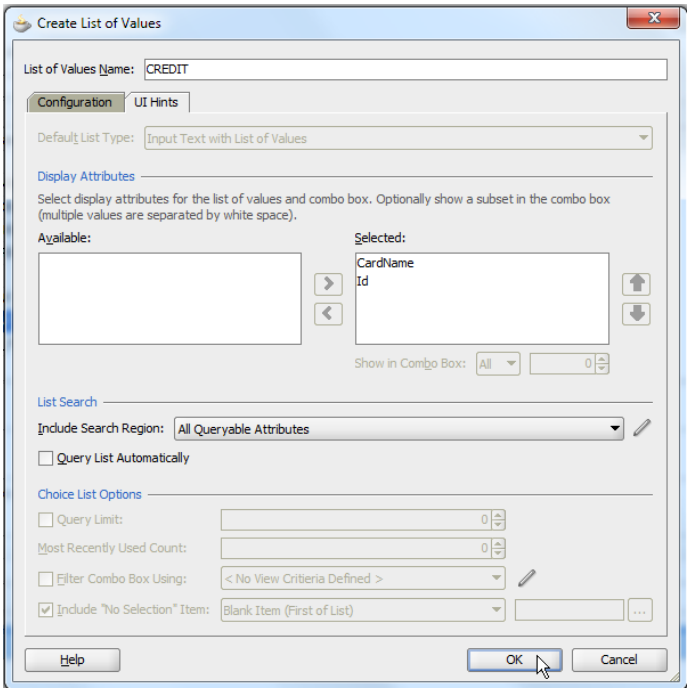
Next, a second LOV needs to be created for the same attribute – *PaymentDetail* – but this time for the CREDIT payment type.



In both cases, the LOV is built as **Input Text with List of Values**. Note that you cannot have two different list types.

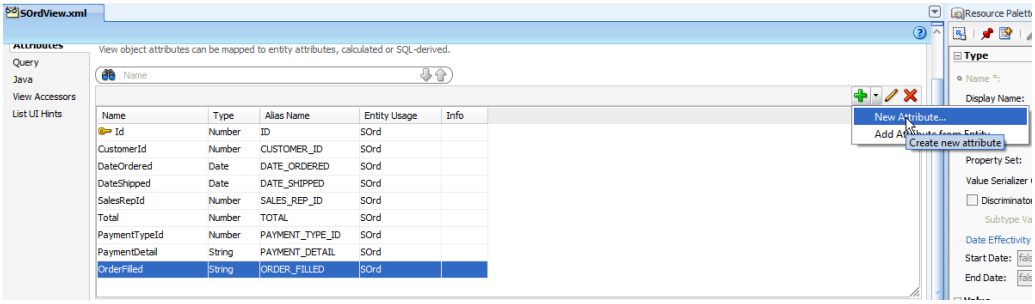


The **List of Values Name** is set to **CREDIT** and the **PaymentDetail** attribute is mapped to the Id value of the **CreditCard** view object. This time, at runtime, when selecting a value of the LOV, the payment detail attribute is updated with the abbreviation of a credit card.



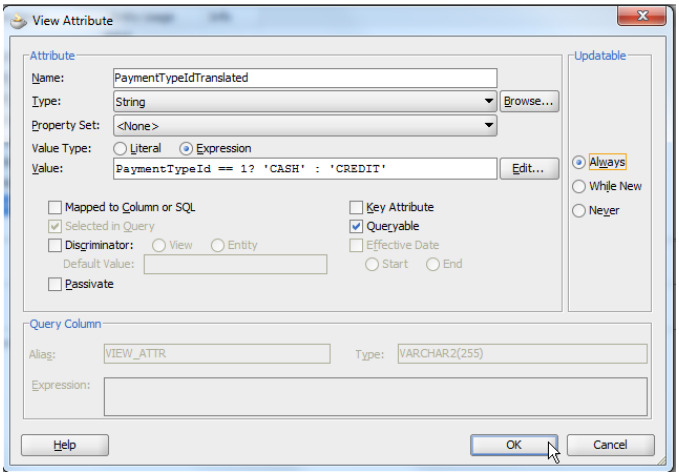
The *S_ORD* table only has a **FK reference** to the *S_PAYMENT_TYPE* table *ID* column. To switch between the different LOV data lists, we need a discriminator column to return a value CASH or

CREDIT. This can be achieved easily with a transient attribute in the View Object as shown in the image below.



The transient attribute is named **PaymentTypeIdTranslated** and uses a Groovy expression to return a value of "CASH" for PaymentTypeId 1 and "CREDIT" for PaymentTypeId 2.

Note that the view attribute's **Value Type** property must be set to **Expression** for Groovy strings to be executed at runtime.



What if you need this expression to be more generic – feeling groovy ?

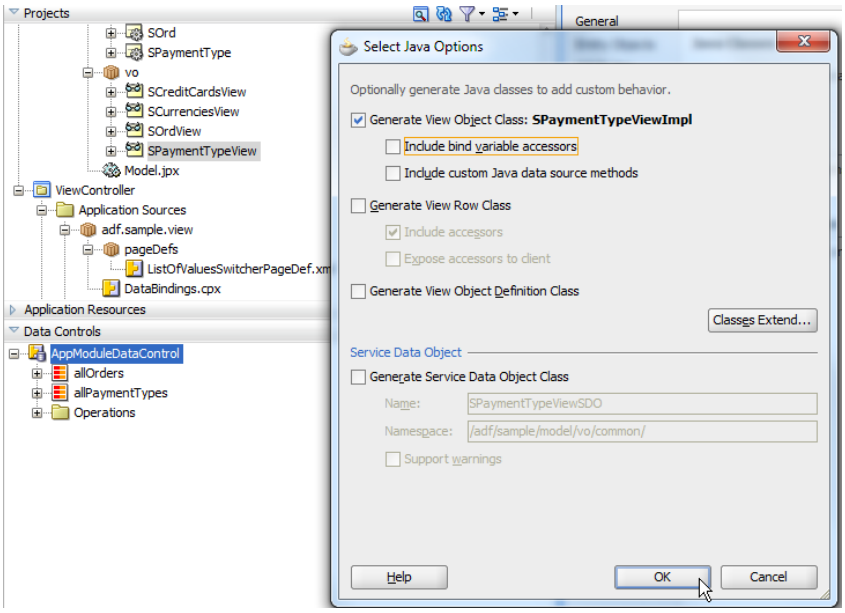
The image above has the value strings CASH and CREDIT hard-coded, which is not optimal. To define the Groovy string more generic, we want to read the translated payment type from a method exposed on a view object created for the S_PAYMENT_TYPE table in the ADF Summit schema.

Note: If you are okay with the hard-coded values, then you may skip the following section and continue with "What we've done so far " on page 12.

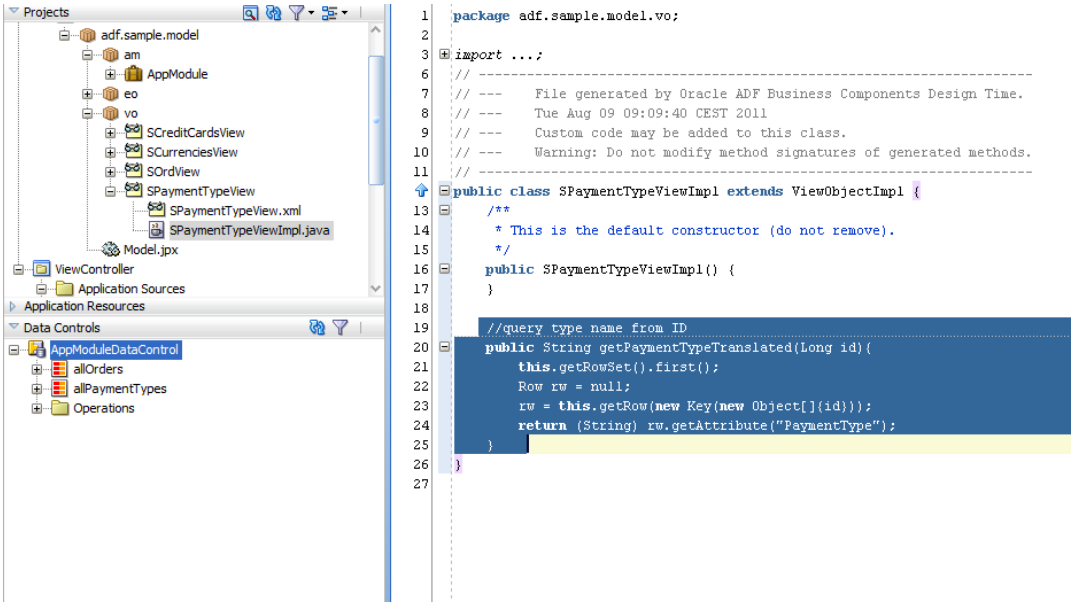
The **SPaymentTypeView** view object is also created using the **ADF Business Components from Table** entry in the Oracle JDeveloper **New Gallery**.

The **SPaymentTypeView** view object is opened in the ADF Business Components editor by a double click on its entry in the Application Navigator. A Java implementation class is created for the view object by selecting the Java menu option in the view object editor and pressing the **pencil** icon next to the **Java Classes** label.

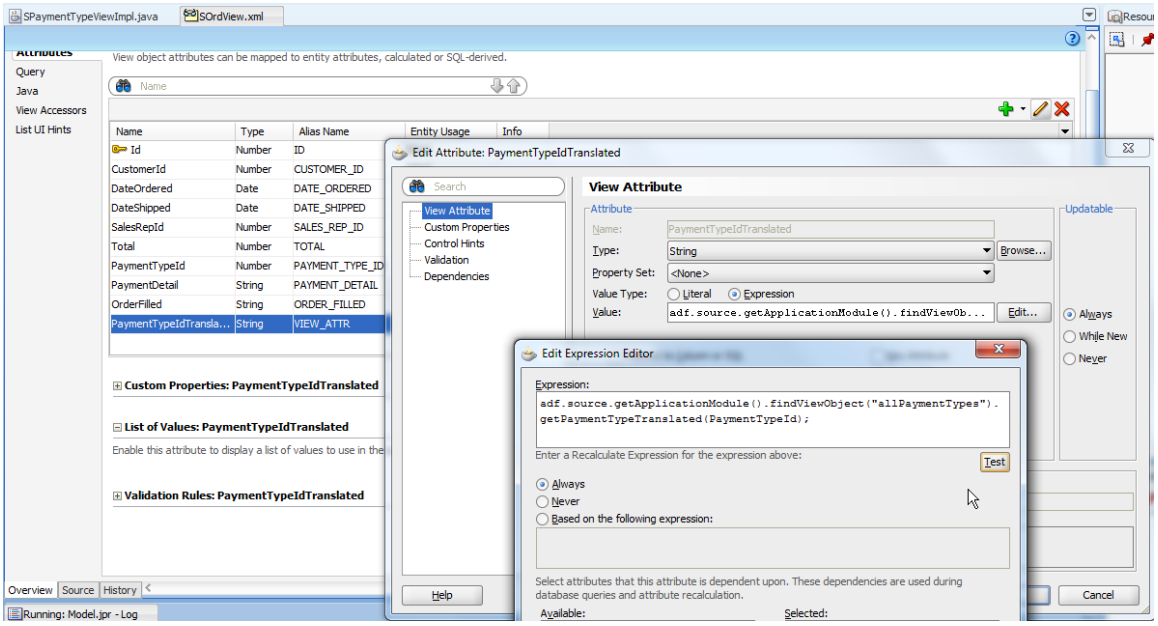
The dialog shown in the image below is displayed to create the **SPaymentTypeViewImpl** class.



In the view object implementation class, a single public method is defined that, when invoked, queries the **SPaymentTypeView** RowSet for the **PaymentType** name matching the ID passed as an argument.



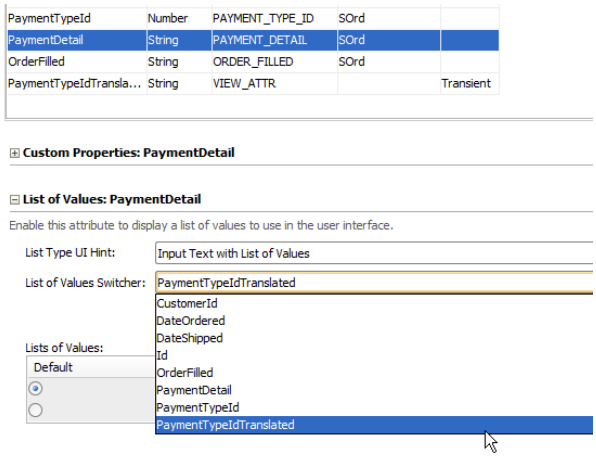
The **PaymentTypeIdTranslated** value expression (Groovy) is then changed to query the public method on the "allPaymentTypes" View Object, which is the instance of the **SPaymentTypeView** view object configured in the Data Model of the Application Module.



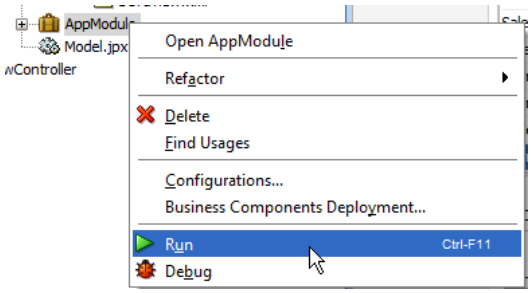

```
//query type name from ID
  public String getPaymentTypeTranslated(Long id){
      this.getRowSet().first();
      Row rw = null;
      rw = this.getRow(new Key(new Object[]{id}));
      return (String) rw.getAttribute("PaymentType");

  }
```

What we've done so far:

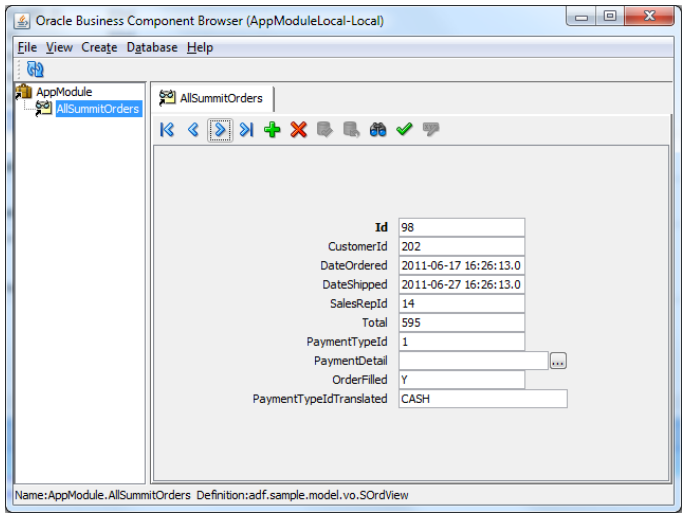
- Create a View Object for the S_ORD table entity
- Create two LOV for the PaymentDetail attribute
- Create a transient attribute to hold the values CASH, CREDIT based on the *PaymentTypeId* attribute value
- Create a public method to translate the PaymentTypeId to its type name. A groovy string is used to update the *PaymentTypeIdTranslated* transient attribute



In a last step, the **PaymentTypeIdTranslated** attribute is configured as the value of the **List of Values Switcher** property. This defines the **PaymentTypeIdTranslated** as the discriminator that – at runtime - decides which list data is displayed in the LOV.



To test the LOV switcher, select the **AppModule** object in the Application Navigator and choose **Run** from the context menu.

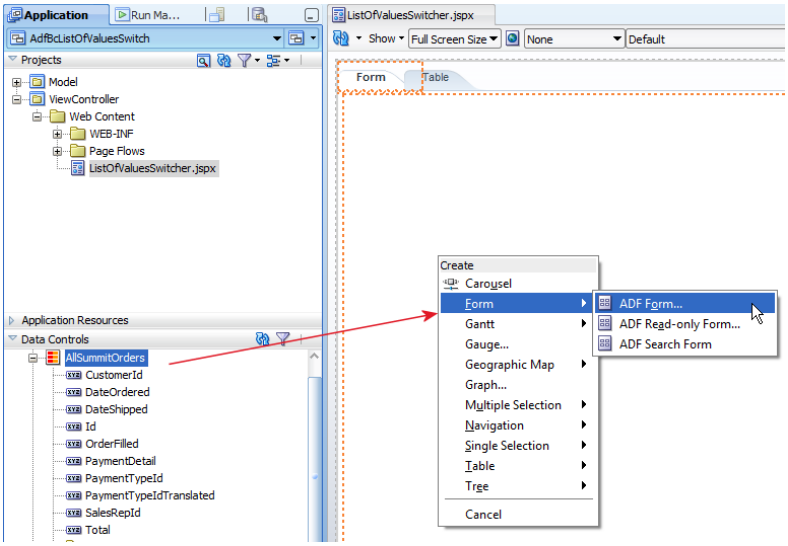


In the ADF BC tester, you can now test the LOV, which, dependent on the **PaymentTypeId** attribute value, shows credit card or currencies values. The list-of-values data also changes when you change the PaymentTypeId from a value of 1 to 2.

ADF Faces View

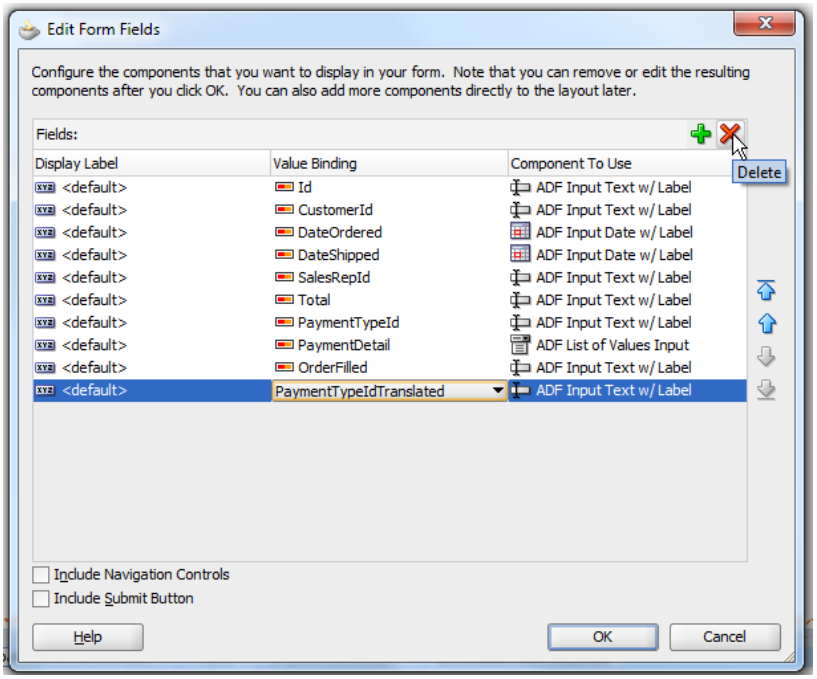
We did the obligatory short program, now it is time for free skating.

The real work has been done in the model already and the ADF Faces UI defined next is only needed to show how to display the LOV switcher in a web environment.

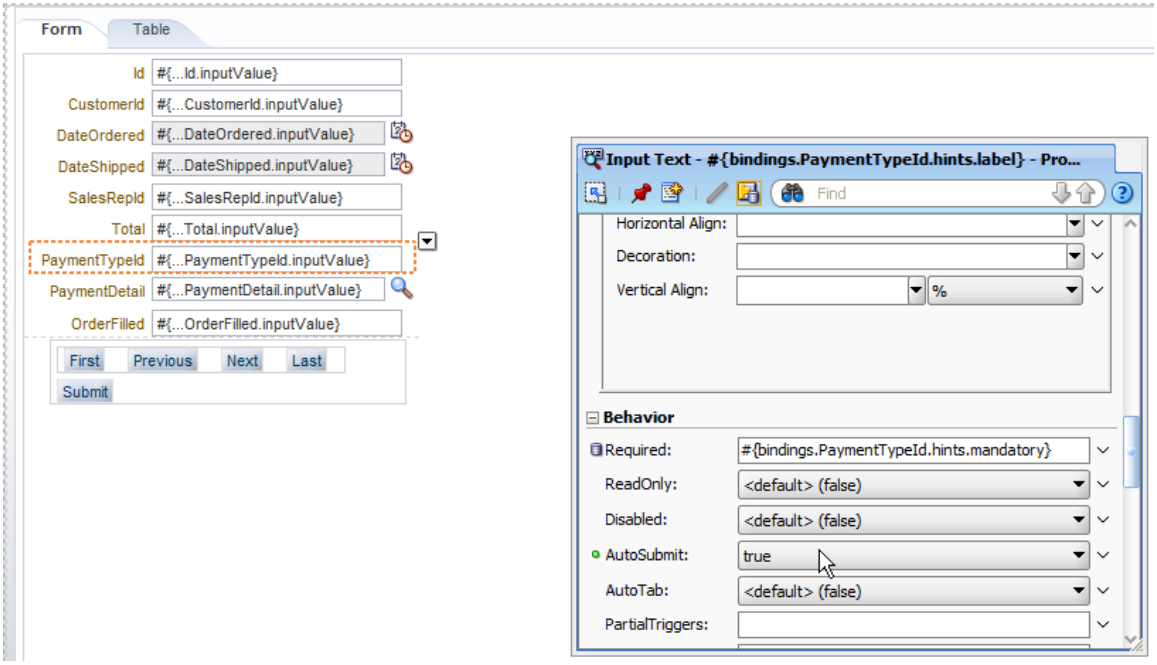


To create an ADF form, we drag the *AllSummitOrders* collection onto the JSF page and choose **ADF Form** from the context menu.

The **PaymentTypeIdTranslated** attribute is internal detail information and not needed to display in the form. It can be deleted from the view as shown in the image below.

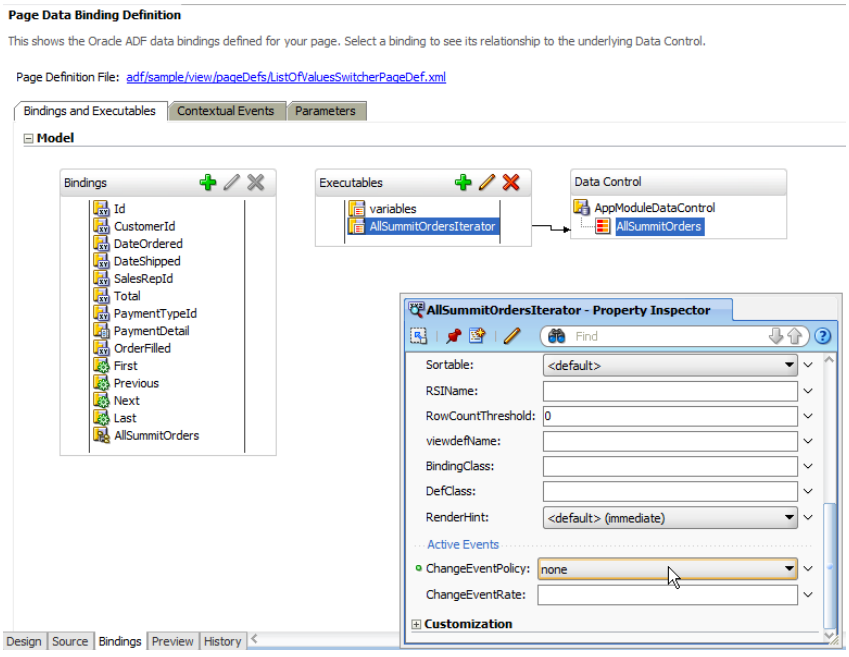


The **PaymentTypeId** field in the form needs to have its **AutoSubmit** property set to **true.** This ensures that a change of the payment Id type is immediately saved in the model so that the list-of-values data list can switch accordingly.



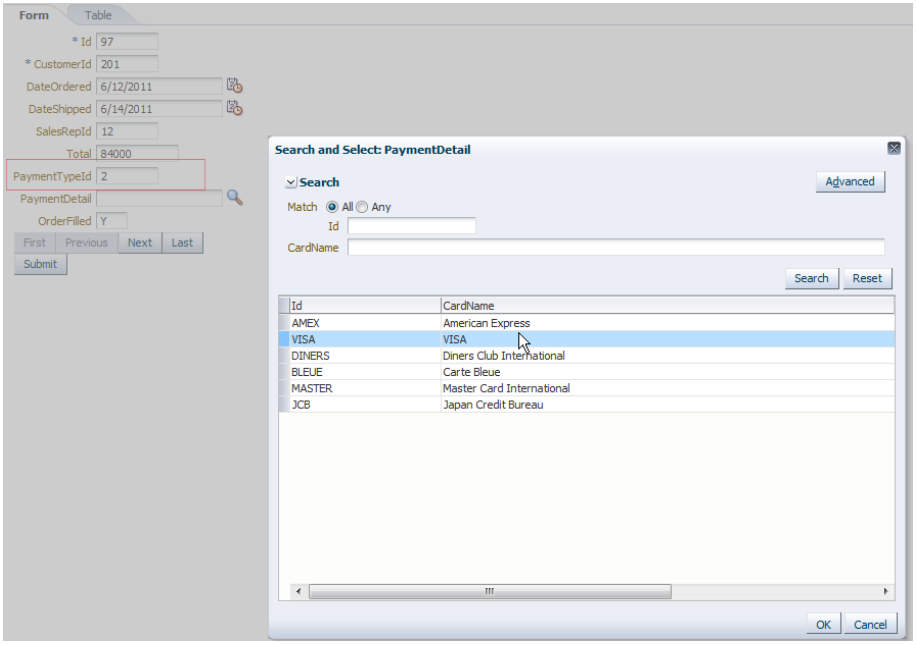
Next, we change the **ChangeEventPolicy**, on the **AllSummitOrdersIterator** in the PageDef file of the JSPX document from **PPR** to **None**.

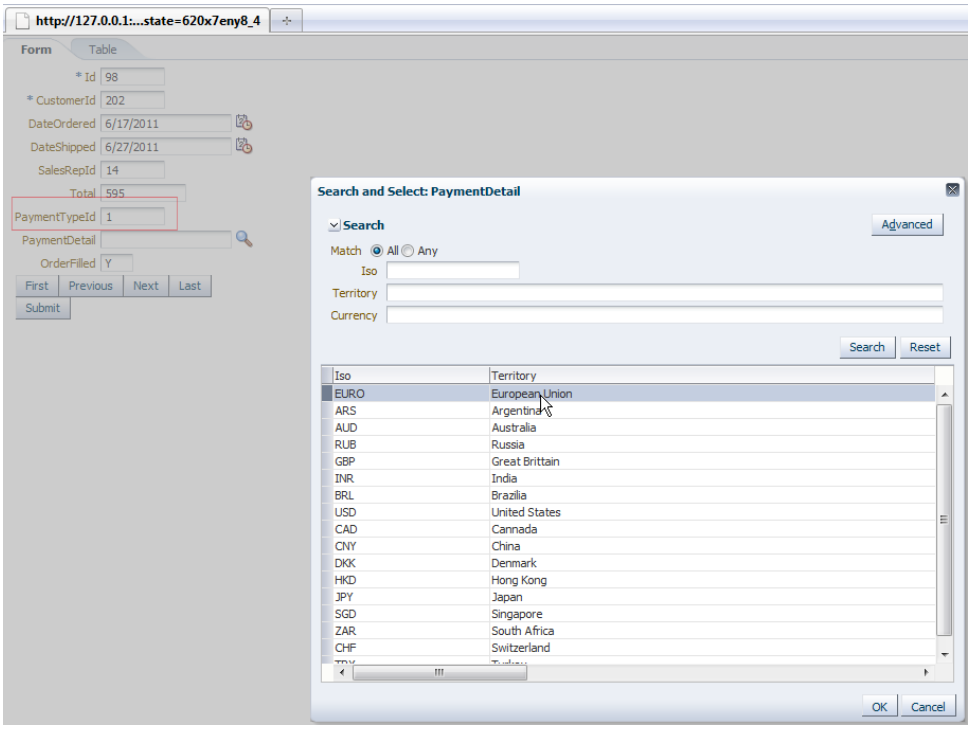
ChangeEventPolicy set to PPR fires a partial refresh on each change of the row currency, which we don't need for this sample. The PageDef file can be accessed from the **Bindings** tab shown at the bottom of the JSPX document in the visual page editor. The image below shows the PageDef view.



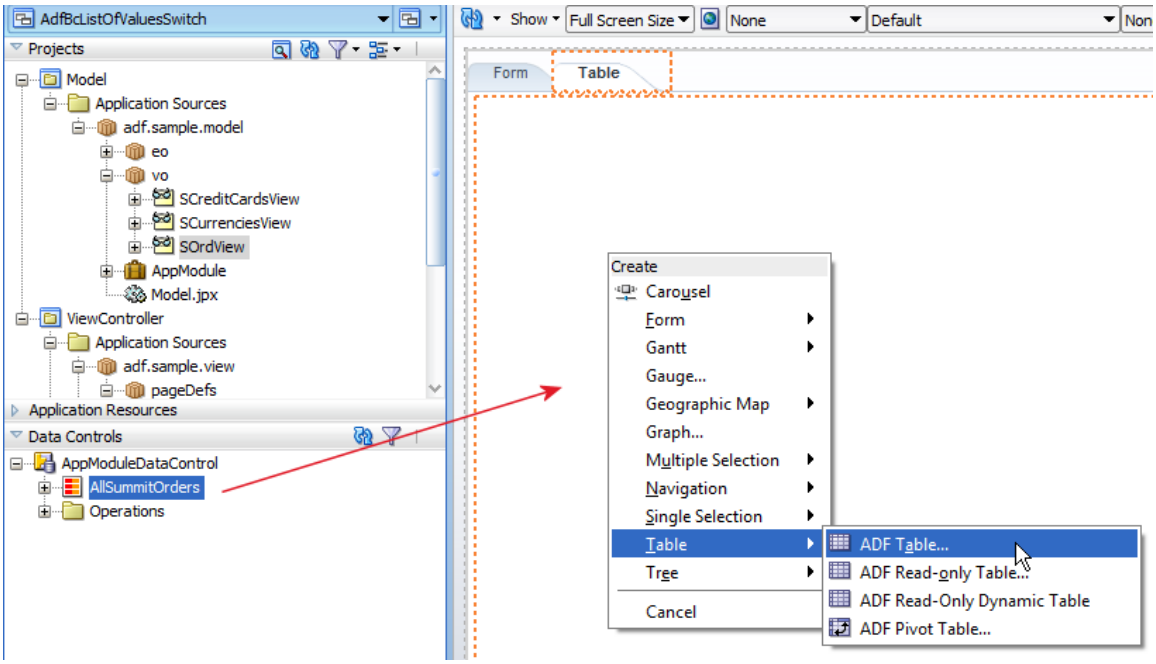
Now it is time for a first test of list-of-value switching in an ADF Faces web environment. Select the JSPX document in the JDeveloper Application Navigator and choose run from the context menu.

When the page comes up in a browser, navigate the ADF form data and click the LOV button for different **PaymentTypeId** values to see the change in the list data shown in the two images below.

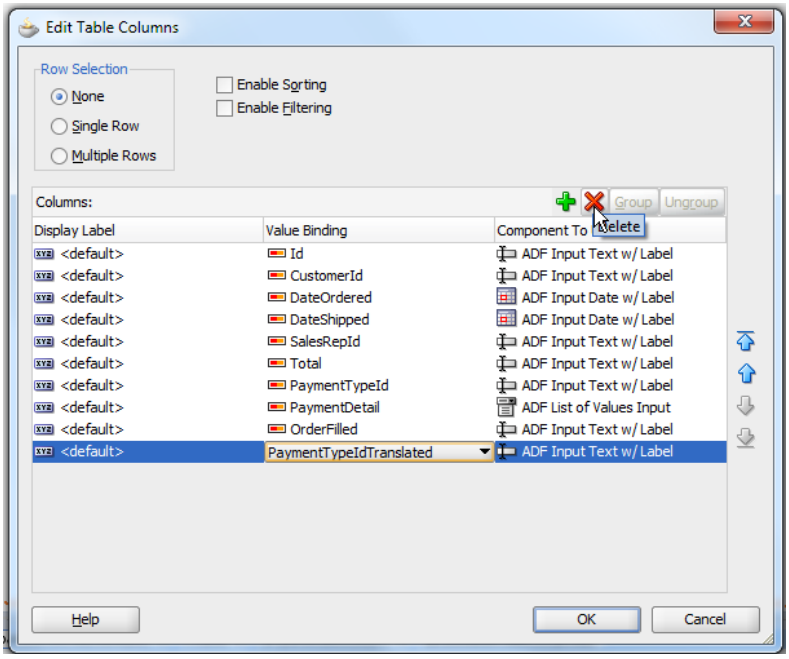




The same LOV switch works in ADF tables too. Drag the **AllSummitOrders** collection to the JSF page and choose **Table | ADF Table** from the context menu.

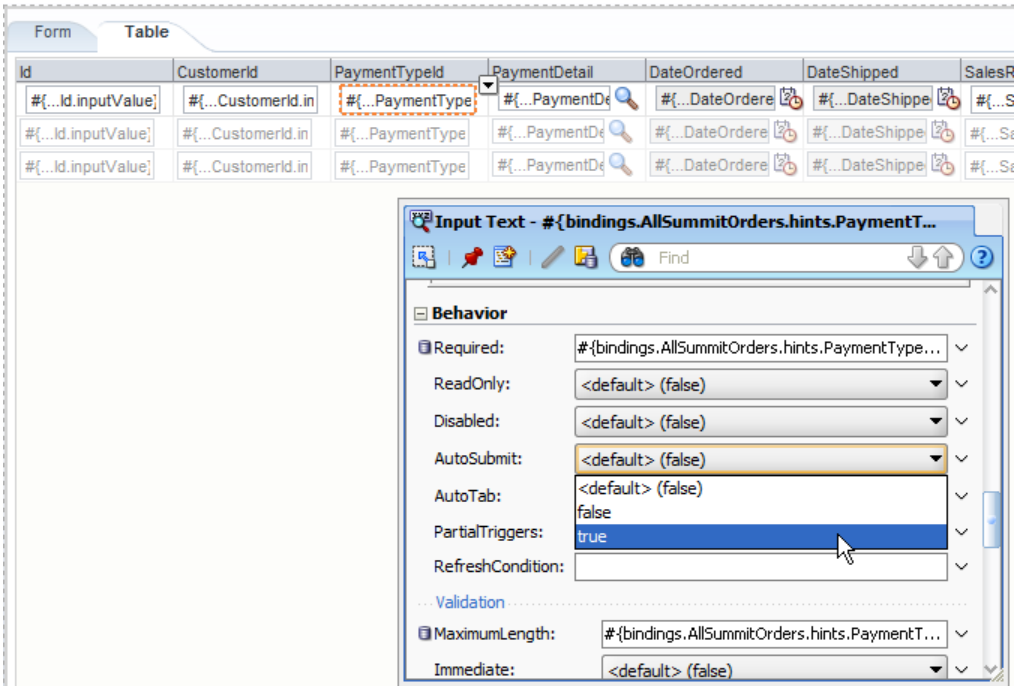


Again, the **PaymentTypeIdTranslated** attribute can be deleted from the display as this is not needed in the table.

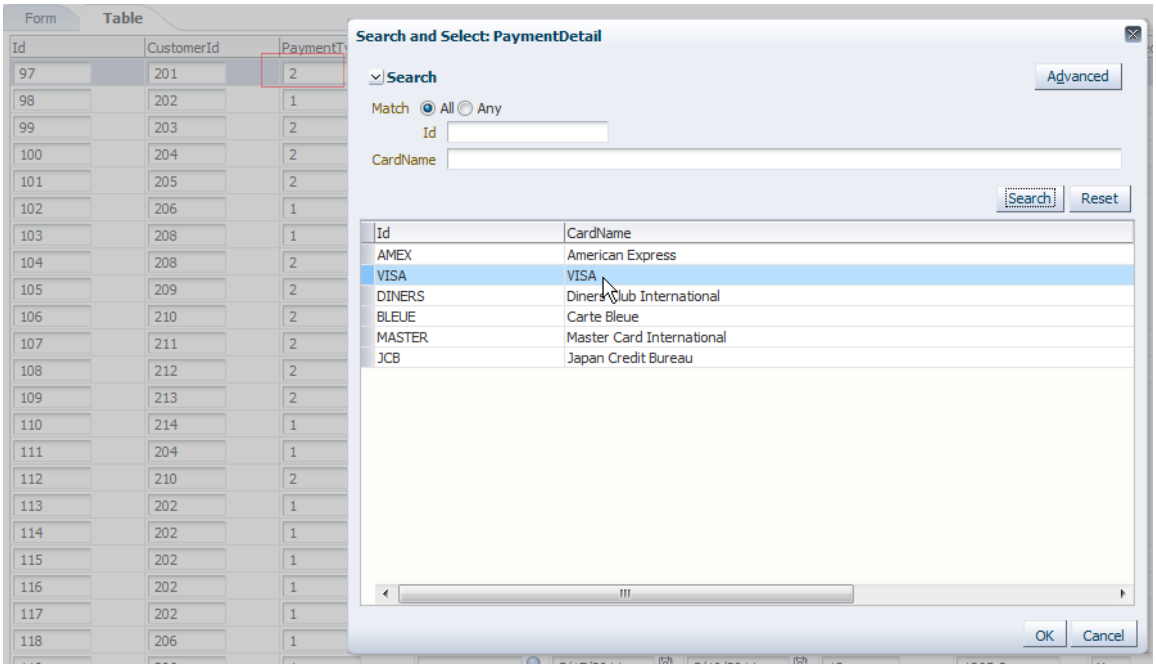


Note: Make sure the ADF table is configured for **single row selection** after dragging the collection onto the page. For this set the **Row Selection** property shown in the image above from **None** to **Single Row**.

Set the **AutoSubmit** property of the `af:inputText` component in the PaymentTypeId column to **true**.



Run the JSPX document to test the LOV for different table rows.



Conclusion

Model driven LOVs in AD Business Components can be configured to conditionally show different sets of data. This article explained how to change the LOV data lists dependent on whether the payment type is CASH or CREDIT. The database schema used in this sample is ADF Summit. You can download the ADF Summit sample and schema scripts from

http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/SummitADF/SummitADFV1_0_08072011.zip

You can download the Oracle JDeveloper 11.1.1.4 sample workspace from ADF Code Corner website where it is sample 89

http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html

Make sure you run the SQL script contained in the SUMMIT_ADF_SCHEMA-PATCH folder to add the credit cards table and currencies table used in this sample. Configure the database connect information in the sample to point to a database of yours that has the ADF Summit schema installed.

RELATED DOCOMENTATION

⊠	ADF Summit application and schema http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/SummitADF/SummitADFV1_0_08072011.zip
⊠	
⊠	