THE TIE THAT BINDS: AN INTRODUCTION TO ADF BINDINGS

Peter Koletzke, Quovera

Normally, connecting database data to the user interface of a web application requires a lot of work. Oracle offers frameworks in JDeveloper 10.1.3 to help with this task for Java-based web applications. The Oracle Application Development Framework (ADF) in JDeveloper provides access to a framework called ADF Bindings that allows you to more easily connect the user interface objects on a web page to the database. ADF Bindings is part of the ADF Model layer, which is the most innovative and remarkable technology in the ADF stack.

This white paper briefly reviews where ADF Bindings fits in the ADF architecture and how it allows you to quickly connect components from any user interface library such as ADF Faces and JSF Reference Implementation to business services such as ADF Business Components (ADF BC). It then gives examples of various bindings and how to automatically bind data elements to visual elements; what types of bindings are available; and where binding code appears. It also explains the basics of the expression language used to bind Model layer components to View layer components.

Note

Bindings are not part of the Java EE specification, but an effort is being made to include it using the Java Community Process (JCP). For more information, search jcp.org for JSR #227 and this article: www.theserverside.com/news/thread.tss?thread_id= 20018.

ADF

The Java EE specifications define a design pattern called "Model-View-Controller" (MVC). This design pattern forms the basis for ADF architecture (depicted in Figure 1).

MVC splits the application into three logical layers: *Model*—to manage data to and from the database; *View*—to manipulate and render the user interface; and *Controller*—to interpret user events such as button clicks into data transfers between the View and Model layers and to determine page flow). ADF adds a fourth layer, Business Services, which defines data sources of various styles. Each layer supports one or more Java frameworks, such as JSF, Struts, and ADF BC.

The main purpose of ADF (represented as the JDeveloper box on the left side of Figure 1) is to provide a common developer interface to any technology within these layers. For example, although you would develop different styles of code when using Enterprise JavaBeans (EJB) and when using ADF BC, the techniques used the code in other layers to these business services will be the same.

Note

This white paper concentrates on ADF BC as an example for the Business Services layer because Oracle is using it as part of the Fusion Stack, technologies used to create the next version of Oracle E-Business Suite ("Oracle Fusion Applications").

The Model layer of ADF consists of ADF code libraries that implement the link between data in the Business Services layer and user interface controls in the View layer (through the Controller layer). You interact with this Model layer library code in JDeveloper using two related parts: ADF Data Controls and ADF Bindings.

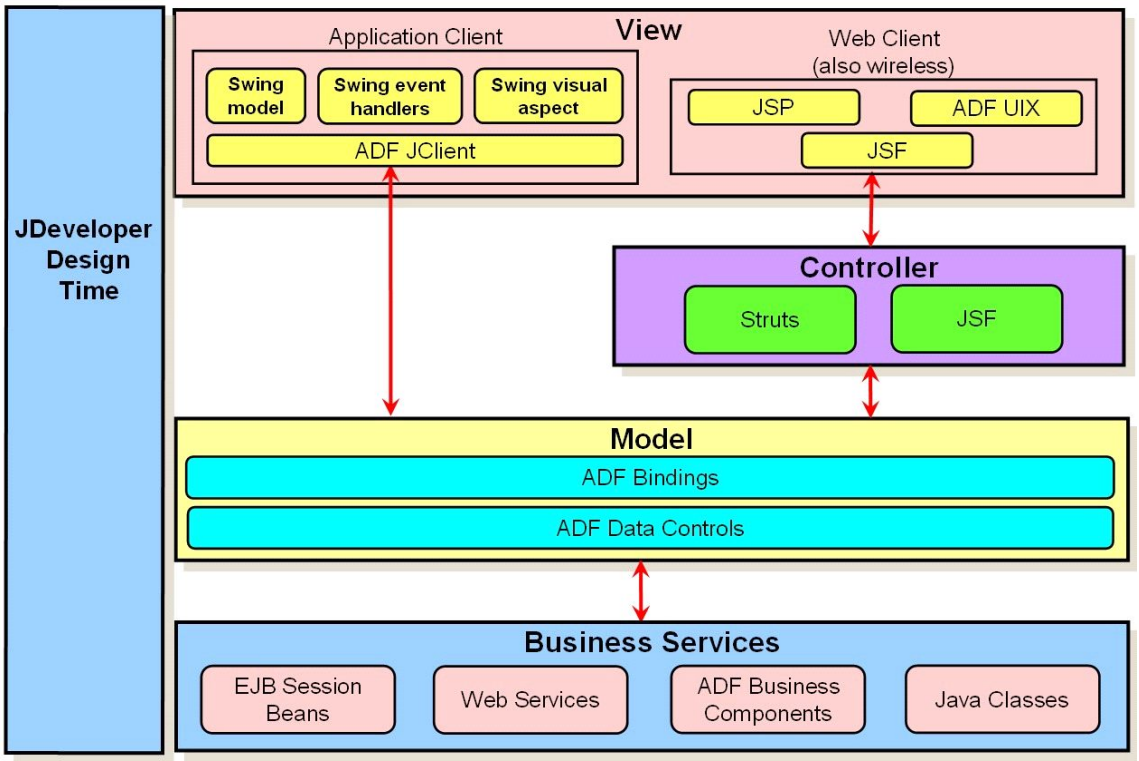


Figure 1. ADF architecture

ADF DATA CONTROLS

Data controls represent the data model in the Business Services layer and provide data to the bindings. The Data Control Palette in JDeveloper displays data model objects define in the Model project. Figure 2 shows the Application Module Editor for a simple data model (containing one view object instance, DepartmentsView) and the corresponding Data Control Palette. The Data Control Palette represents the data control as the top-level node. *Data collections* (DepartmentsView in this example) appear under the data control note. Data collections represent more than one data element—in this case, all attributes in the DepartmentsView view object). Within each data collection, you will see attributes such as DepartmentId and DepartmentName that correspond to the attributes of the ADF BC view object. The data collection also offers *operations* (actions) such as Create, Delete, and Find. In addition, the data control level defines operations for Commit and Rollback.

The Data Control Palette allows you to drag various component types onto the View layer page (.jsp or .jspx, for example). Each level of the data model offers a relevant set of components. For example, dragging a data collection node (such as DepartmentsView) onto a JSP file will offer collection-level components made up of multiple attributes such as tables, forms, trees, and navigation controls. Dragging an attribute node onto the JSP will offer components like input text fields, output text fields (boilerplate text), and pulldown lists.

Data controls are defined in the DataBindings.cpx file found in the ViewController project's view package; each ViewController project will contain one DataBindings.cpx file for all data controls. In JDeveloper's Applications Navigator, this file is displayed under the Application Sources node. JDeveloper creates this file automatically when you first drop a node from the Data Control Palette onto the first page of your application. For each page onto which you drop data controls, JDeveloper adds a reference to this file for the page on which the data control will be used.

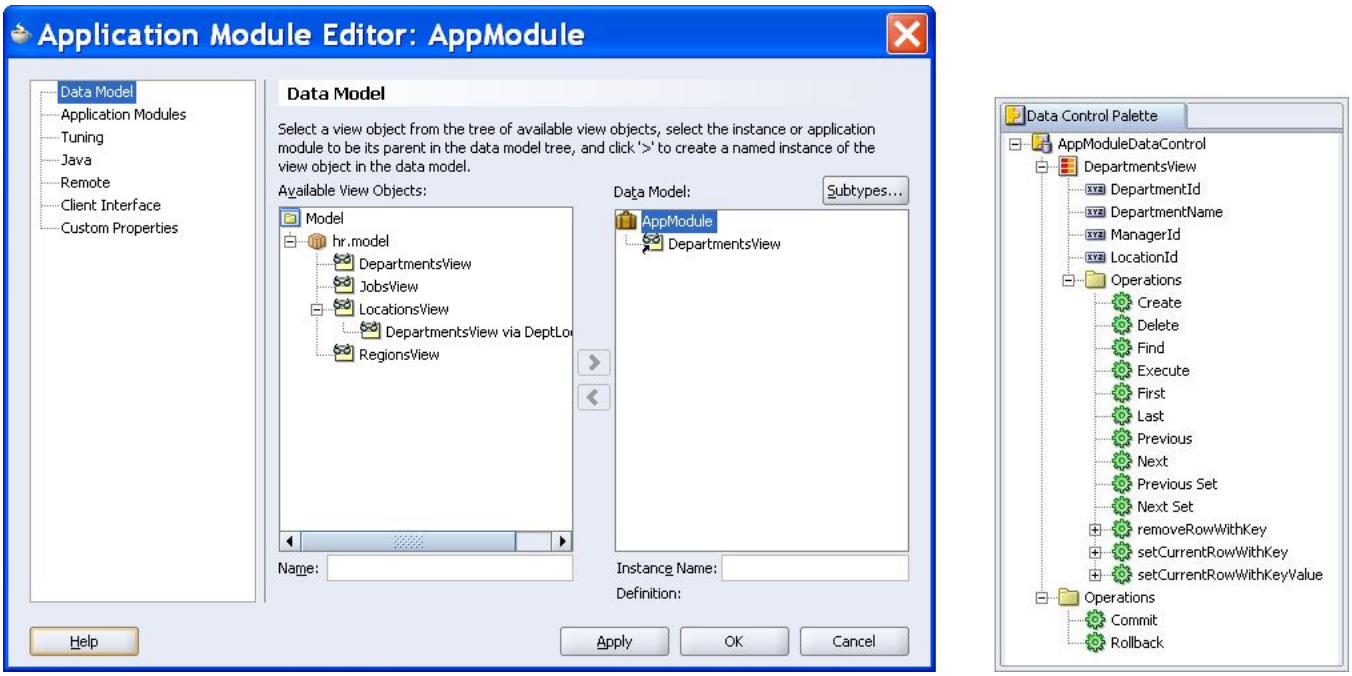


Figure 2. Application Module Editor with a data model and its corresponding Data Control Palette

The following code listing shows the entire contents of a DataBindings.cpx that was created for an application containing one JSP page, dept.jspx.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Application xmlns="http://xmlns.oracle.com/adfm/application"
             version="10.1.3.40.66" id="DataBindings" SeparateXMLFiles="false"
             Package="hr.view" ClientType="Generic">
  <pageMap>
    <page path="/dept.jspx" usageId="deptPageDef"/>
  </pageMap>
  <pageDefinitionUsages>
    <page id="deptPageDef" path="hr.view.pageDefs.deptPageDef"/>
  </pageDefinitionUsages>
  <dataControlUsages>
    <BC4JDataControl id="AppModuleDataControl" Package="hr.model"
                     FactoryClass="oracle.adf.model.bc4j.DataControlFactoryImpl"
                     SupportsTransactions="true" SupportsFindMode="true"
                     SupportsRangesize="true" SupportsResetState="true"
                     SupportsSortCollection="true"
                     Configuration="AppModuleLocal" syncMode="Immediate"
                     xmlns="http://xmlns.oracle.com/adfm/datacontrol"/>
  </dataControlUsages>
</Application>
```

Notice that the file name of the page (dept.jspx) and its *PageDef file* containing its binding definitions (deptPageDef) is listed under the pageMap element. The name of the PageDef file is "<*package name*>_<*JSP file name*>" by default (where "<*package name*>" is the directory where the JSP page is located and "<*jsp file name*>" is the name of the file). The location of the PageDef file is declared in the pageDefinitionUsages element of DataBindings.cpx. JDeveloper updates the .cpx file when you add application modules (for additional BC4JDataControl entries) or pages (for additional pageMap entries).

Note

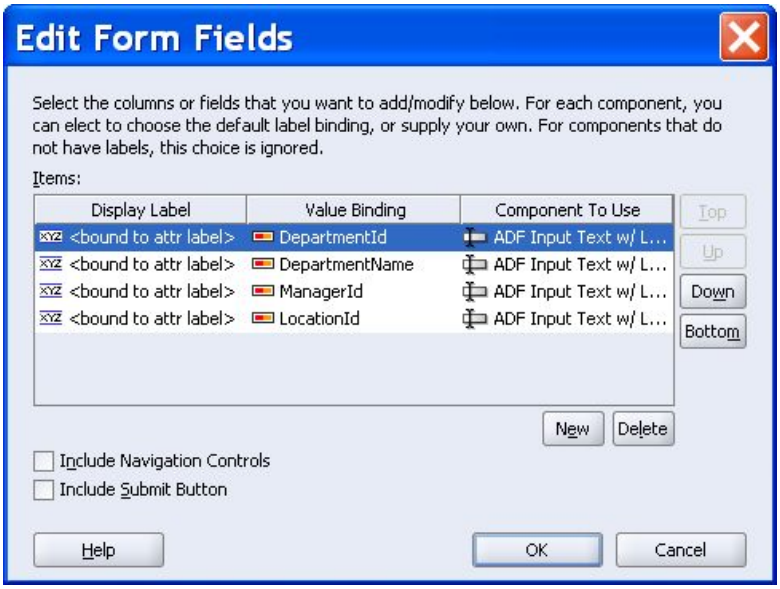
If you rename a JSP file, the JSP and PageDef file names in the DataBindings.cpx file will not automatically be updated. You need to manually change references in this file if you choose to rename a JSP or PageDef file. Also, be sure to remove the JSP file names from the PageDef and DataBindings.cpx files if you delete a JSP file that they reference.

ADF BINDINGS

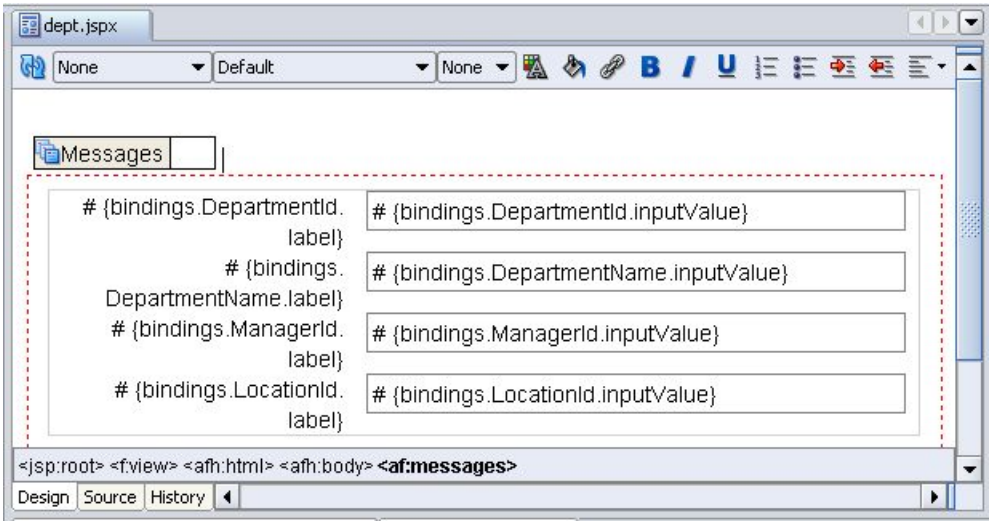
Bindings represent the link between the data control's data objects and the View layer user interface component. They are defined in the PageDef file, which JDeveloper creates automatically for each page in the user interface (ViewController) project. You would use following steps to automatically bind data to user interface objects in dept.jspx.

1. Drag the DepartmentsView node from the Data Control Palette (shown in Figure 2) to the Visual Editor of a JSP page. The following context menu will appear:

2. Select ADF Form. The following dialog will appear.



3. Click OK. The Visual Editor will display the following layout:



These actions create the following code in the PageDef file (deptPageDef.xml) for dept.jspx.:

```
<?xml version="1.0" encoding="UTF-8" ?>
<pageDefinition xmlns="http://xmlns.oracle.com/adfm/uimodel"
                version="10.1.3.40.66" id="untitled1PageDef"
                Package="hr.view.pageDefs">
  <parameters/>
  <executables>
    <iterator id="DepartmentsView1Iterator" RangeSize="10"
              Binds="DepartmentsView1" DataControl="AppModuleDataControl"/>
  </executables>
  <bindings>
    <attributeValues id="DepartmentId" IterBinding="DepartmentsView1Iterator">
      <AttrNames>
        <Item Value="DepartmentId"/>
      </AttrNames>
    </attributeValues>
    <attributeValues id="DepartmentName" IterBinding="DepartmentsView1Iterator">
      <AttrNames>
        <Item Value="DepartmentName"/>
      </AttrNames>
    </attributeValues>
    <attributeValues id="ManagerId" IterBinding="DepartmentsView1Iterator">
      <AttrNames>
        <Item Value="ManagerId"/>
      </AttrNames>
    </attributeValues>
    <attributeValues id="LocationId" IterBinding="DepartmentsView1Iterator">
      <AttrNames>
        <Item Value="LocationId"/>
      </AttrNames>
    </attributeValues>
  </bindings>
</pageDefinition>
```

Just as JDeveloper reveals data controls the Data Control Palette, it reveals data bindings in the PageDef file. A *binding context* defines the data controls and bindings available to the application. Although the mechanics of how the application interacts with the binding context are largely automatic, you can write Java code to customize the binding context or to retrieve values and run operations available to the binding context.

The communication relationships between ADF layers and its parts are shown in Figure 3.

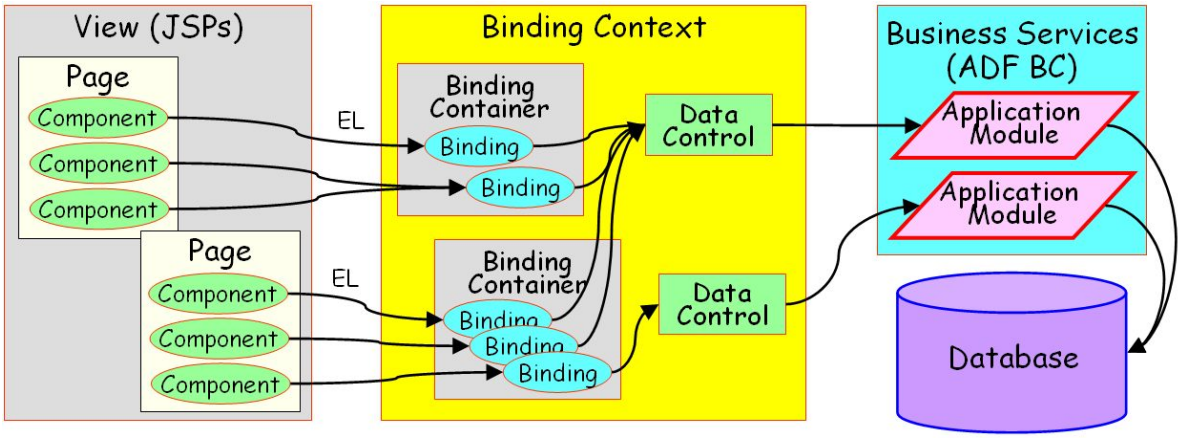


Figure 3. Data flow through the ADF layers of the application

In this diagram, the components on the JSP pages are linked to the binding context through a *binding container* (defined in the PageDef file for each JSP page) using Expression Language (EL, as described in the next section). More than one component can share the same binding. The bindings are linked to data controls defined in the DataBindings.cpx file; each data control represents an ADF BC application module in the Business Services layer. (Other types of Business Services code such as EJB will be associated with data controls in the same way as ADF BC.) The application modules access the database through view objects in the ADF BC layer.

BINDING ACTIONS (OPERATIONS)

In addition to accessing data, bindings can access operations—actions or methods defined in the Business Services layer. For example, the JSP page might require a Save button to commit changes to the database. The bindings for the page would link the Commit operation of the application module to this button. The Data Control Palette shown in Figure 2 contains a node for Commit that represents the Commit operation available to the application module. Service methods you add to the application module Java implementation file (for example, HRServiceImpl.java) will also appear as data controls so you can bind them to user interface components. The technique for creating an operation binding is the same as for a data control: drag the operation to the page and select the control to represent it (for example, a button or a link).

The Data Control Palette shown in Figure 2 also contains operations such as Create, Next, Previous, and Delete on the data collection level. These operations are specific to a data collection so they are repeated for each data collection node in the Data Control Palette. The Commit and Rollback operations are global to all view object instances in the application module so they only appear once in each data control.

EXPRESSION LANGUAGE

Expression Language (also called "JSP Expression Language" or "EL") is described in the Java EE standards for JavaServer Pages Standard Tag Language (JSTL). EL allows you to code procedural logic tags such as forEach, if, and choose within a JSP file. Many technologies other than JSP can use it as well.

All EL expressions use the form "#{<*expression*>}" or "${<*expression*>}" where "<*expression*>" is the text that represents values or procedural logic. JavaServer Faces (JSF) JSP files use the former prefix ("#") to represent component properties and concentrates on values not on procedural logic constructs like forEach. The path to an element property value uses the Java dot (".") separator syntax. For example, an af:inputText (text field component) could be coded as follows:

```
<af:inputText
  value="#{bindings.DepartmentId.inputValue}"
  label="#{bindings.DepartmentId.label}"/>
```

The expressions for the *label* and *value* properties retrieve data from the binding context (appropriately called "bindings"). The path to these values contains the context "bindings" pointing to the PageDef file associated with the JSP page. The path also contains the name of the attribute binding in the PageDef file (in this case, DepartmentId). Therefore, "bindings.DepartmentId" refers to the data control attribute DepartmentId representing a view object attribute DepartmentId available through the application module (as depicted in Figure 3). The final part of these expressions is the property name

that the expression accesses, for example, *inputValue* (for the data value of the attribute) and *label* (for the label associated with the attribute). If a control hint label is defined in the ADF BC view object or entity object for this view object instance, that control hint will be returned by the expression "#{bindings.DepartmentId.label}." Otherwise, the label returned will be the default assigned by ADF BC (the attribute name).

Expressions can also contain operators such as "!" (not), "|" (or), and "&" (and). Here is an example of an EL expression containing operators:

```
#{bindings.EmployeesViewIterator.findMode ? '* Find Mode' : ''}
```

This example uses the ternary operator ("? :") to return a blank value or the string "* Find mode" based on whether the user has placed the form into Find Mode (like Enter Query mode in Oracle Forms). In this case, the Boolean value of the *Find Mode* property of the EmployeesViewIterator is evaluated by the ternary operator.

CONSTRUCTING EL

When you drag and drop a data control onto a JSP page, JDeveloper will create the relevant binding in the PageDef file. It will also assign the relevant properties of the component to expressions. For example, after dragging the DepartmentsView collection onto the page as an ADF Form, the following component tags will be written into the JSP file for the DepartmentId attribute:

```
<af:inputText value="#{bindings.DepartmentId.inputValue}"
              label="#{bindings.DepartmentId.label}"
              required="#{bindings.DepartmentId.mandatory}"
              columns="#{bindings.DepartmentId.displayWidth}">
  <af:validator binding="#{bindings.DepartmentId.validator}"/>
  <f:convertNumber groupingUsed="false"
                   pattern="#{bindings.DepartmentId.format}"/>
</af:inputText>
```

Notice that the *value*, *label*, *required*, and *columns* properties of the DepartmentsId af:inputText component have been automatically filled in. JDeveloper also creates the PageDef file for the page (if it doesn't exist) and adds binding definitions to it. These binding definitions will be accessed by the JSP attribute EL expressions at runtime.

You can change any of these expressions by typing values into the Property Inspector or code editor. Alternatively, the Property Inspector (shown in Figure 4) offers assistance in entering or changing expressions using the *binding editor*, a dialog that assists in constructing EL.

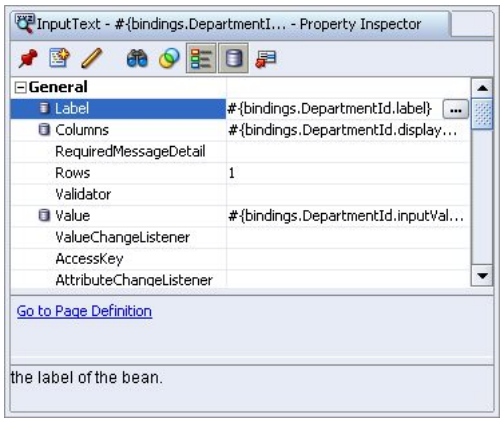


Figure 4. Property Inspector displaying EL expressions

To access the binding editor, select a property containing an expression and click the "…" button in the property's value field.

If the property has no binding, the "…" button will not be available, so click the **Bind to data** button  instead. The binding editor shown in Figure 5 will appear.

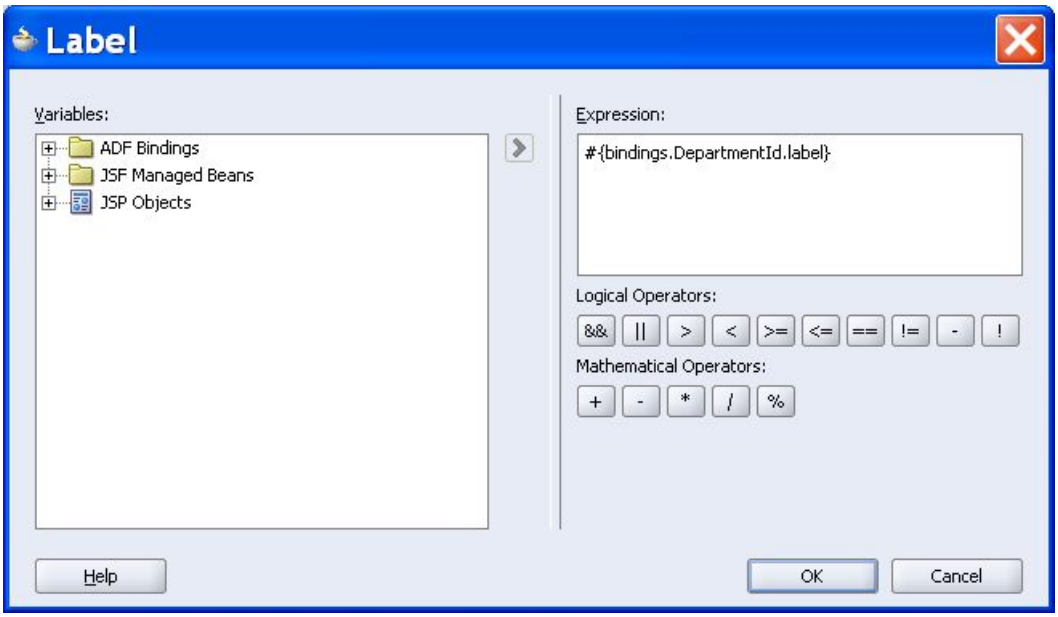
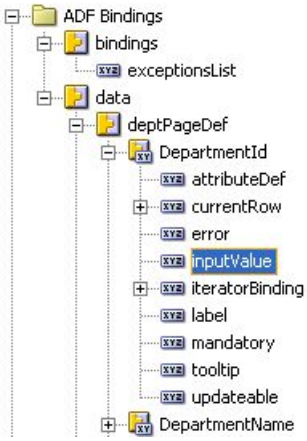


Figure 5. Binding editor

The navigator under the *Variables* prompt allows you to find available bindings and select them. When you click the ">" button after selecting a binding, the expression will be constructed in the *Expression* field. You can also enter operations in th expression by using the operator buttons under the *Expression* field (although you will probably find it easier to type in the operators). Clicking OK returns the expression to the Property Inspector. When a property value is bound to an expression, the **Bind to data** button will be selected in the Property Inspector toolbar.

For example, if you wanted to add a binding to the inputValue property of the DepartmentId attribute and were unclear of the exact syntax, you could use the navigator in the binding editor to find and construct the bindings.DepartmentId.inputValue node as shown here:



Caution

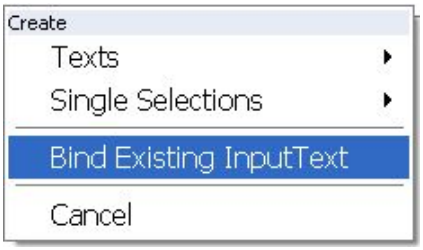
The **Bind to data** button () will be selected for a property once you assign an expression to the property. If you click this button while it is selected, a non-EL variation of the expression will replace the EL.

Clicking the **Reset to default** button () will remove the expression completely. (Use Undo (Ctrl-Z) if you mistakenly remove an expression.)

Do not click **Bind to data** to edit an EL expression. Use the "…" button in the property value field instead.

BINDING EXISTING COMPONENTS

In addition to creating bound components, you can use the Data Control Palette to add bindings to existing components. This is useful if you prototype screens by dropping components on the page from the Component Palette, for example, because the data model is not defined well enough to support the items. To bind an existing component, drag the relevant node from the Data Control Palette onto the component. For example, if you drag an attribute node from the palette onto an af:inputText component, you will see the context menu on the right. If you select **Bind Existing InputText** from this menu, JDeveloper will add EL binding expressions to all appropriate properties and will add relevant bindings to the PageDef file.



Tip

JDeveloper maintains the link between bindings and components while you are using the Visual Editor. For example, if you delete a component by selecting it in the Visual Editor and clicking the Delete key, JDeveloper will remove the corresponding binding from the PageDef file. This is true when editing using the Structure Window as well. However, if you use the code editor to delete the code that defines the component, JDeveloper will not remove the associated binding code.

You can use this effect to your advantage. For example, if you want to create a binding that you will only use programmatically, you can drop an item on the page so the binding is created. Then you can remove the item in the code editor. This is usually faster than creating the binding manually.

THE PAGEDEF FILE

The PageDef file contains sections for parameters you pass to the page and executables (actions that are automatically run when the page loads) as well as bindings. You can display the PageDef file by double clicking it in the Applications Navigator or by selecting **Go to Page Definition** from the right-click menu of the JSP page. Since the file has no diagrammatic representation, the code will then appear in the Source tab of the code editor. You can edit the file using the code editor and Property Inspector, which will display properties of the element in which the cursor is placed. You can alternatively use the right-click menus of the Structure Window (shown in Figure 6) to add, update, or move bindings.

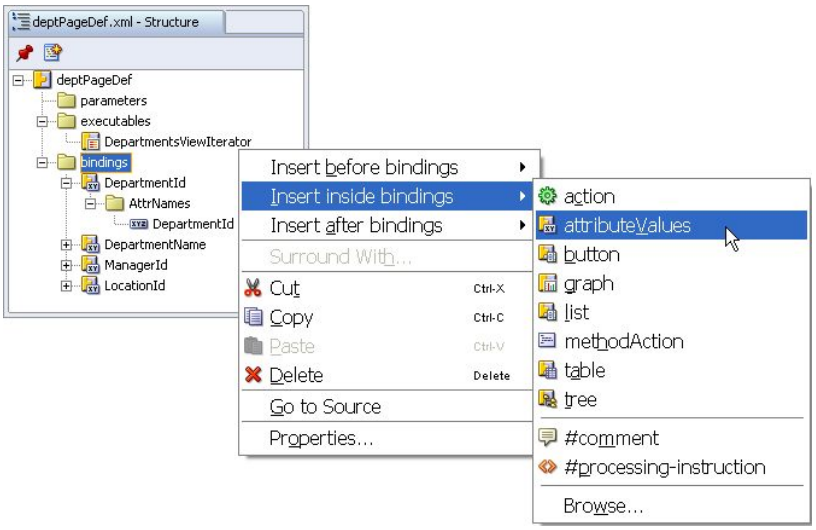


Figure 6. Structure Window right-click menu for a PageDef file

JDeveloper adds the PageDef file to the project when you drop the first data control on the first page. It creates a subdirectory, called "pageDefs" by default, in the view directory to hold all PageDef files. You can change this default directory name using the Project Properties dialog's ADFm Settings page.

The following excerpt from the PageDef file defines an attribute binding for the DepartmentId attribute:

```
<attributeValues id="DepartmentId" IterBinding="DepartmentsView1Iterator">
  <AttrNames>
    <Item Value="DepartmentId"/>
  </AttrNames>
</attributeValues>
```

Tip

You will probably find that maintaining binding definitions is easiest and most accurate when you use the Structure Window (shown in Figure 6). You can use the insert options from the right click menu to add elements This action will display a property editor window where you can set properties of the new element. You can edit properties of an existing binding by selecting Properties from the right-click menu on the element's node in the Structure Window or by using the Property Inspector after selecting the node in the Structure Window.

ITERATORS

In the preceding code snippet, the attributeValues *id* property matches the ADF BC name (referenced by the AttrNames element). The *IterBinding* property refers to an *iterator*, a definition in the executables section of the PageDef file that points to the current row in the view object instance's view cache. You would create multiple iterators for any view object instance if you need to have more than one current row in the same view object cache. Iterators work much in the same way as cursors in PL/SQL. Both iterators and cursors provide access to rows of data in a query result set and both include the concept of a current row to which the structure points. Here is an example of an iterator definition for DepartmentsView from the PageDef file:

```
<iterator id="DepartmentsViewIterator" RangeSize="10"
          Binds="DepartmentsView" DataControl="AppModuleDataControl"/>
```

One important property of an iterator is *rangeSize* that defines how many rows are displayed in table components. This property is set to 10 by default but you can modify it using the Property Inspector for the applicable iterator in the executables section of the PageDef file or by editing the PageDef code in the code editor.

TYPES OF BINDINGS

The DepartmentId *attribute binding* shown in the code snippet above is used for a component that requires a single value such as data from one column in one row of a table or database view. Other types of components require other types of bindings. The following table briefly describes the usage of the attribute binding and other types of bindings.

Binding Type	Use
Attribute	This is used for the attribute value of the attribute in the current row of the iterator.
Table (or range)	Table bindings (also called *range bindings*) are used for table components (displaying rows and columns) that are bound to collections. They expose data from all rows in an iterator and all or some of the attributes. The value of a table control is defined in the *collectionModel* property. A row in the collection is identified with a variable name (by default, "row").
List	This is used for data-bound list elements of pulldown components. The value is loaded from a dynamic list (loaded from a view object) or a static list (values coded into the binding). When you select a list item for an attribute you drop from the Data Control Palette, a list binding editor will appear to allow you to define the properties for the list binding.
Action	This is used for standard operations like Commit, Create, and Delete. When you drop an action binding on the page, you can represent it as a button or link.
Navigation List	This us used to manipulate the current row in a set. This binding can be used to change the current row in an iterator (for the Next and Previous buttons, for example).
Method	This is used for custom methods you write. When you drop a method binding on the page, you can represent it as a button or link. If the method has parameters, you can select to add fields to the page for those parameter values as well as the button or link.
Boolean	Use this binding for checkbox components.
Tree	This is used to represent a set of master-detail data in hierarchical components.

Note

In earlier versions of JDeveloper, iterators were considered a type of binding. Although they are defined in the PageDef file, you need to create other types of bindings to access their features.

CONCLUSION

This white paper has described ADF and the parts of the ADF Model layer. It explained how ADF Bindings interact with the ADF Data Controls and ADF BC frameworks. You can rely on JDeveloper to create bindings when you drop data controls on the page, and it is a good exercise to follow the data communication thread through the various files. This will help you become more comfortable with this powerful feature of ADF and will allow you to better add, modify, and debug bindings for web applications you build with JDeveloper

###

Peter Koletzke is a technical director and principal instructor for the Enterprise e-Commerce Solutions practice at Quovera, in Mountain View, California, and has 25 years of industry experience. Peter has presented at various Oracle users group conferences more than 220 times and has won awards such as Pinnacle Publishing's Technical Achievement, Oracle Development Tools Users Group (ODTUG) Editor's Choice, ECO/SEOUC Oracle Designer Award, ODTUG Volunteer of the Year, and NYOUG Editor's Choice. He is an Oracle Certified Master, Oracle ACE Director, and coauthor of the Oracle Press Books: *Oracle JDeveloper 10g for Forms & PL/SQL Developers* (with Duncan Mills); *Oracle JDeveloper 10g Handbook* and *Oracle9i JDeveloper Handbook* (with Dr. Paul Dorsey and Avrom Roy-Faderman); *Oracle JDeveloper 3 Handbook*, *Oracle Developer Advanced Forms and Reports*, *Oracle Designer Handbook*, *2nd Edition*, and *Oracle Designer/2000 Handbook* (all with Dr. Paul Dorsey).