

---

## Understanding The Basics of ADF and Jdeveloper From an Oracle Forms Perspective

Author: Thomas Korbecki



Tom Korbecki has worked with Oracle Applications for more than 15 years. His first assignment on Oracle Applications project was to develop custom applications in Release 10. He currently is solution architect/developer that designs and develops versatile, customer-centric solutions for Oracle Application Products.

White Paper Presentation  
OAUG Connection Point - Apps Tech 2013  
Pittsburgh, PA

## Abstract

This presentation was created to help Oracle Forms Developers transition their current Oracle Forms skill set to the next version of forms development called Application Development Framework (ADF). ADF has been around since 2004 but most Oracle Forms Developers are just now having to learn ADF because Oracle is developing future applications and products using a new technology stack which supports ADF. ADF is based on the Java 2 Platform, Enterprise Edition (J2EE) set of standards. This means that Oracle Forms Developers will need to transition their existing skill sets and/or acquire new skill sets to develop applications and forms in a Web based environment. In others words, Oracle Forms Developers will need to be become Web Developers and this white paper will help you start your journey into becoming a Web Developer via ADF.

In order to build ADF applications, Oracle has provided a new integrated development environment (IDE) called JDeveloper. JDeveloper is similar to Oracle Forms in that its purpose is to simplify application development by providing a visual and declarative approach to building your application or form. Unlike the Oracle Forms tool which can build Oracle Forms applications, JDeveloper includes a development environment that supports SQL, PL/SQL, Java, XML, HTML, JavaScript, BPEL, and PHP. Since JDeveloper is a multi-purpose IDE tool, you will be intimidated by the layout, menu options, profile options and wizards, as compared to the existing E-Business Suite environment where there is a separate IDE for each Oracle Forms, Oracle Reports, Oracle Workflow and XML Gateway.

From my experience, some of your Oracle Forms skill sets can transition very easily while others will be a challenge. In addition, there are several new skills to learn and based on your past experience and/or education with Web Development, the learning curve can be fairly short but in most cases it will require an extended amount of effort. Remember when you first started developing in Oracle Forms, it took only a few days to build a simple form but it took you months or years to master it. The same can be applied to building ADF forms and applications. In fact, it will take longer to master ADF because there are hundreds of more components and services to learn.

This white paper will highlight the skill sets required to build your ADF application, steps required to set up your development environment, basic navigation of the Oracle JDeveloper software, building an ADF application with comparison to Oracle Form development and provide some understanding on how to deploy an ADF application.

## Objective

1. What are the challenges with transitioning from Oracle Forms to Oracle ADF?
2. How to learn the required programming skill set to build ADF applications?
3. What are the basic ADF concepts?
4. Provide side by side comparisons between Oracle Forms and Oracle ADF.
5. Provide links to relevant websites explaining key concepts.

## **What are the challenges associated with the transition from Oracle Forms to Oracle ADF?**

The same basic building blocks required in Oracle Forms development are still required with Oracle ADF development. The basic building blocks include gathering the requirements, designing the application User Interface (UI) and Data Model, building the application and then releasing the application in a production environment; however, there are new challenges encountered along the way.

Your first challenge will be designing your new application because it will take time and experience to understand the ADF framework of Model, View and Controller. The ADF framework separates the database layer (Model) from the UI layer (View) while introducing a new navigation concept called a Task Flow (Controller).

Task Flow is a modular and reusable unit of business navigation between views and non-visual activities like routers and methods. Task Flow can be compared to an Oracle EBS workflow. An Oracle EBS workflow can execute PL/SQL code and route a transaction, whereas, an ADF Task Flow can execute Java code, route a transaction; as well as call other ADF Pages or call another Task Flow.

Your second challenge will be navigating within the JDeveloper software and selecting the correct set JDeveloper configuration for your ADF application. The confusion is due to JDeveloper being a multipurpose development tool and ADF is just one of tools within the same software.

For comparison, it would be like Oracle combining Oracle Forms, Oracle Reports and Oracle Workflow into one common development tool. When you start your new development task, the Developer has to select the correct configuration so the development software only offers the tools available for that configuration.

Your third challenge will be learning the Java programming language and how to research and use the pre-built Java APIs. It will take time to build your understanding of the most commonly used Java APIs and Oracle specific APIs. I would compare it to learning how to research and use the correct Oracle EBS public APIs. For non-Java Developers, it is initially overwhelming; however, it is only a matter of time before it all makes sense.

Your final challenge will be deploying and administering your ADF applications in the WebLogic Server. JDeveloper provides two ways to deploy ADF applications: 1) Deploy directly from the JDeveloper application; 2) Deploy using an Enterprise Application Archive (EAR File). In addition, basic navigation skills are required for the WebLogic console and enterprise manager screens.

## **How to learn the required programming languages and software tools to build an application in ADFs?**

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

The most difficult and frustrating part is trying to find the equivalent Oracle Forms action in ADF. Initially, it might take hours or days trying to understand how to perform a certain task in ADF. I would rely on my database knowledge to solve a certain problem in PL/SQL when the same task could have been performed in ordinary Java. In order to help minimize the learning curve, I will attempt to draw comparisons between Oracle Forms (PL/SQL) and ADF (Java).

Oracle Forms programming language is based on PL/SQL; whereas, ADF is based on Java. Most ADF applications will require some programming so a basic understanding of Java or Groovy (Java-like Syntax) is required. Most of the programming concepts transfer from PL/SQL to Java, but the syntax will be different. Based on my experience, the following Java topics/concepts are required to build basic ADF applications:

Primitive data types
Data types / variables
Strings
Variable scope
Operator
Ternary operator
Relational and conditional operations
If statement
Loops
Arrays / Lists
Maps
What causes Java null pointer exception
Try Catch Finally
Logging
Import statements

These concepts can be found in many online sites and here are a couple of sites for reference:

Oracle Java Tutorial	<a href="http://docs.oracle.com/javase/tutorial/">http://docs.oracle.com/javase/tutorial/</a>
IBM Java Tutorial	<a href="http://www.ibm.com/developerworks/java/tutorials/j-introjava1/j-introjava1-pdf.pdf">http://www.ibm.com/developerworks/java/tutorials/j-introjava1/j-introjava1-pdf.pdf</a>

Oracle provides pre-built Java APIs just like Oracle EBS provides pre-built PL/SQL APIs, so take advantage of them. In order to use the pre-built Java APIs, the class needs to be added to the code via the import statement. Oracle's pre-built Java APIs can be referenced at this link:

Java APIs	<a href="http://docs.oracle.com/javase/6/docs/api">http://docs.oracle.com/javase/6/docs/api</a>
Oracle APIs	<code>oracle.jbo.*</code> package

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

In addition to Java programming language, there is a Java-like scripting language called Groovy. Groovy is a dynamic language for the Java platform that is checked and executed at runtime as opposed to Java, which is checked at compile-time. In ADF applications, Groovy is typically used to validate business rules, set a database sequence on a table, build error messages, referencing built-in calls like setting current date and time (i.e. sysdate), and aggregate functions in view objects (i.e. sum, count, min, max, etc.).

Introduction to Groovy Support in JDeveloper and Oracle ADF 11g	<a href="http://www.oracle.com/technetwork/developer-tools/jdev/introduction-to-groovy-128837.pdf">http://www.oracle.com/technetwork/developer-tools/jdev/introduction-to-groovy-128837.pdf</a>
---	---

On occasion, you will need JavaScript to react to some client event to get information from the Clients device. JavaScript was originally implemented as part of the web browser so the client-side scripts could interact with the User, control the browser, communicate asynchronously, and alter content that is displayed. In the case of ADF applications, I have used JavaScript to determine the preferred language in the Web browser, determine the Users Time Zone and store and retrieve cookies.

Introduction To JavaScript	<a href="http://www.w3schools.com/js/js_intro.asp">http://www.w3schools.com/js/js_intro.asp</a>
----------------------------	---

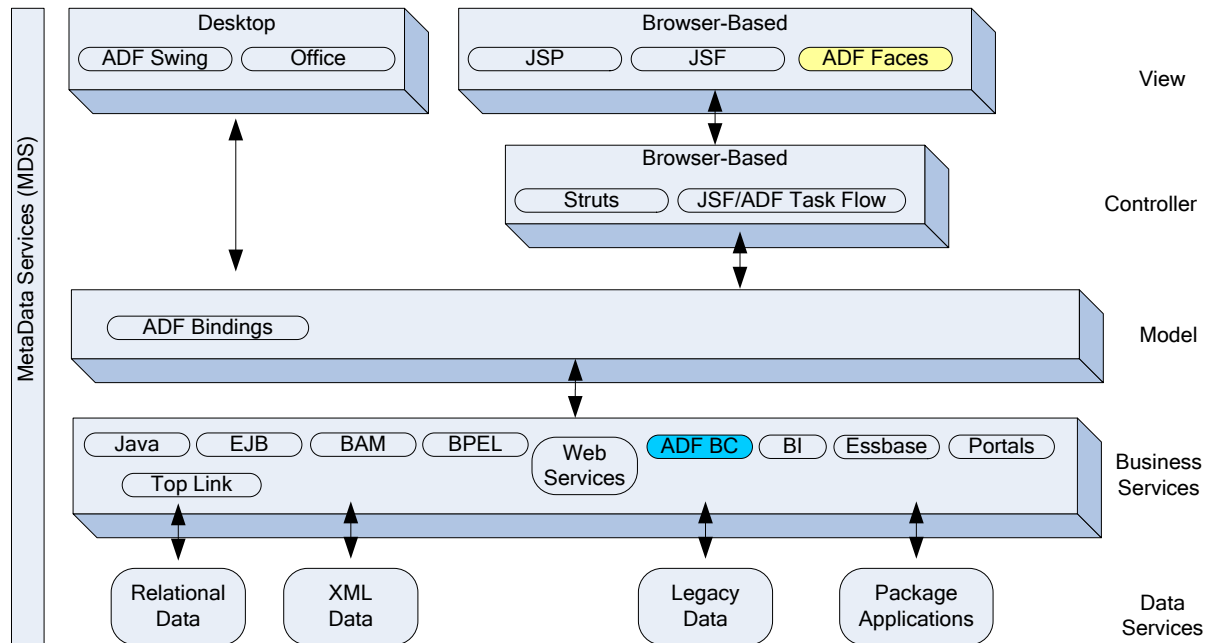
### What are the basic ADF concepts?

#### ADF Concepts

What is the ADF Framework?

Oracle Application Development Framework (ADF) is a Java-based development tool (much like Oracle Forms is a PL/SQL-based tool) designed to take full advantage of Java Enterprise Edition or Java EE. ADF Technology simplifies interaction with “Java” EE and Oracle’s Fusion Middleware.

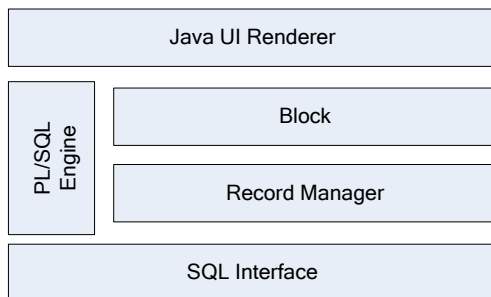
## The Basics of ADF and JDeveloper From an Oracle Forms Perspective



- The view layer contains the UI pages used to view or modify that data
- The controller layer processes user input and determines page navigation
- The model layer represents the data values related to the current page
- The business service layer handles data access and encapsulates business logic

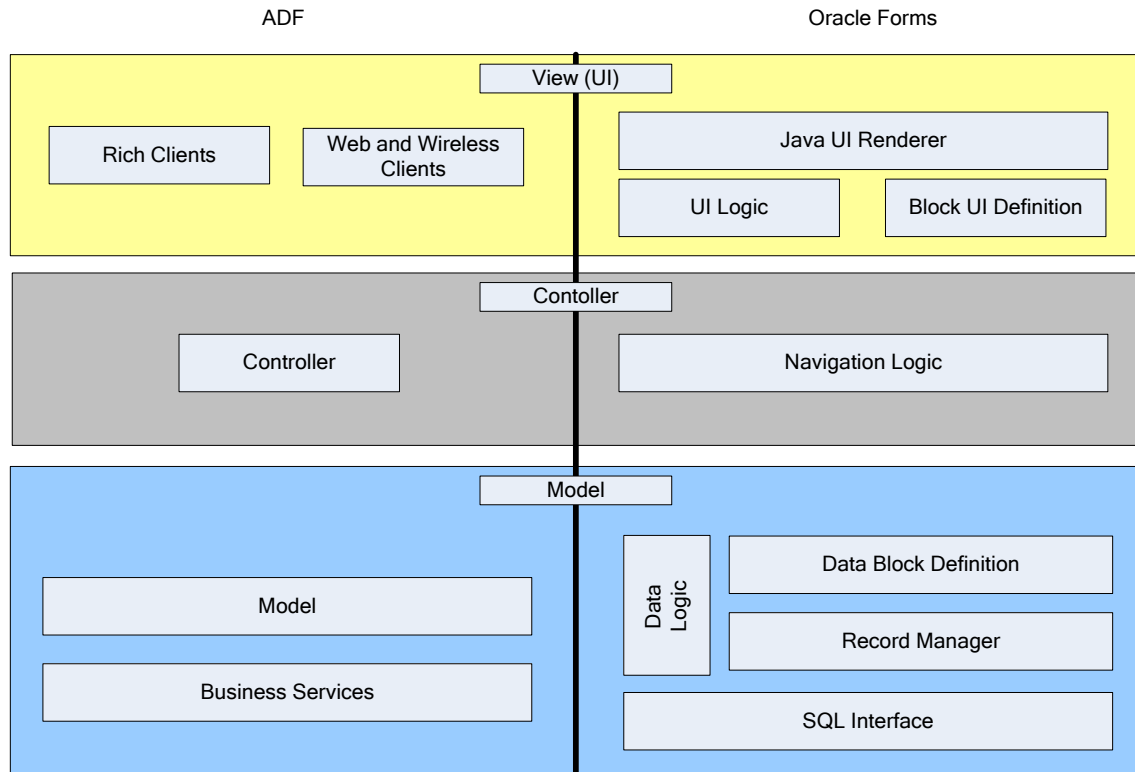
ADF has many components, but the ADF BC (Data & Links) and the ADF Faces (UI) can be thought of as Form Data Blocks (Data & Relationships) and Canvases (UI).

What is the Oracle Forms Framework?



# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

## Compare ADF vs Oracle Framework



### Model

Entity object – For the context of this white paper, an entity object is a database table on which DML operations are performed. In Oracle forms, this is the Record Manager.

The full ADF definition of an entity object is business components that encapsulate the business model, including data, business rules (When-Validate-Record), and persistence behavior for items/columns that are used in your application. Entity object definitions map to single objects in the data source and it is where DML operations are performed. In most cases, it is a database table or snapshot in a database, but it can be a spreadsheet, XML or a flat file.

View object – For the context of this white paper, a view object can be a single database table or a group of database tables linked together via relationships (i.e. parent table and child table) or a view only database view that can be used in a “List of Values” (LOV). In Oracle Forms, this could be considered the Data Block. For example, in a Forms Data Block, we specify what columns we are using and we can also set the “where by” or “order by” clause.

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

The full ADF definition of a view object is a group of business components (ie. Database columns) that are a collection of data from a data source. It can represent an individual table or a group of tables as well as a database view used by a LOV. View objects must have a process for retrieving data from the data source (database) and a method of retrieving the data (SQL based query). Oracle ADF Business Components can automatically use JDBC API to pass this query to the database and receive the result.

View Link– For the context of this white paper, a view link is a relationship between two view objects that are joined by one or many database columns. This enforces business rules related to the data. A view link is used to keep data in-sync in the View Layer. Note: If an association link exists, then use that relationship in the view link, so you join based on the association link and not the individual field(s).

Association “Link” – This is a relationship between two entity objects, which can be represented by a primary – foreign key relationship in a database. If the relationship is defined in the database, then ADF will automatically build an association link between the two entity objects. If the relationship is not defined, then you have the option to create it manually. The association link is used to keep data in-sync in the Model layer.

The Data Model is somewhat comparable to Data Blocks in Oracle Forms. The Data Block controls create, read, update and delete (CRUD) operations as well as link the UI data fields.

### View

For the basis of the white paper, the View layer is the user interface that displays data from the Model layer. The View layer, in its simplest form, could be a page with input fields, buttons, input boxes and tables to display data; however, ADF provides other components that allow for the creation of rich and reusable user interface.

ADF Faces provides over 100 rich components, including hierarchical data tables, tree menus, in-page dialogs, accordions, dividers, and sortable tables. ADF Faces also provides ADF Data Visualization components, which are Flash- and SVG-enabled components capable of rendering dynamic charts, graphs, gauges, and other graphics that can provide a realtime view of underlying data. Each component also supports customization and skinning, along with internationalization and accessibility.

In order to understand these components, Oracle provides a tool called “ADF Faces Rich Client Demos”. The tool is located at:

Oracle ADF Faces Rich Client	<a href="http://jdevadf.oracle.com/adf-richclient-demo/faces/index.jspx">http://jdevadf.oracle.com/adf-richclient-demo/faces/index.jspx</a>
------------------------------	---

### Controller



## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

The Controller layer processes user input and determines page navigation. JDeveloper provides a declarative way, or Task Flow, to pass application control between different types of activities such as pages, methods within managed beans, transaction support, save points, or calls to other task flows. In terms of Oracle Forms, it is the ability to navigate within your application using PL/SQL code to navigate to a block (GO\_BLOCK) or to open another form (CALL\_FORM).

There are two types of Task Flows:

**Unbounded Task Flow** is essentially the entry point into your application or home page. There can be only one unbounded task flow per application

**Bounded Task Flow** is a modular and reusable application flow with a defined entry point (i.e. default activity), but can have zero to many exit points. Additional information about Task Flows is as follows:

Task Flow Design Fundamentals (An Oracle White Paper April 2011)	<a href="http://www.oracle.com/technetwork/developer-tools/jdev/adf-task-flow-design-132904.pdf">http://www.oracle.com/technetwork/developer-tools/jdev/adf-task-flow-design-132904.pdf</a>
--	---

Oracle® Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework 11g Release 1 (11.1.1.6.0)
--

- |   |
|---|
| <ul style="list-style-type: none"><li>- Section 14 Getting Started With Task Flows <a href="http://bit.ly/adfdevguide111160s14">http://bit.ly/adfdevguide111160s14</a></li><li>- Section 16.4 Sharing Data Control Instances <a href="http://bit.ly/adfdevguide111160s164">http://bit.ly/adfdevguide111160s164</a></li><li>- Section 18 Introduction to Complex Task Flows <a href="http://bit.ly/adfdevguide111160s18">http://bit.ly/adfdevguide111160s18</a></li><li>- Section 18.3 Managing Transactions <a href="http://bit.ly/adfdevguide111160s183">http://bit.ly/adfdevguide111160s183</a></li></ul> |
|---|

### Additional Concepts

#### Application Module (AM)

This is a collection of business rules and transactions (view objects and view links defined in the Model layer) that are related to a certain function or use case. When building your application, the view and view links need to be added to a container or application module (AM) so they can be exposed as “data controls” and consumed by the View layer.

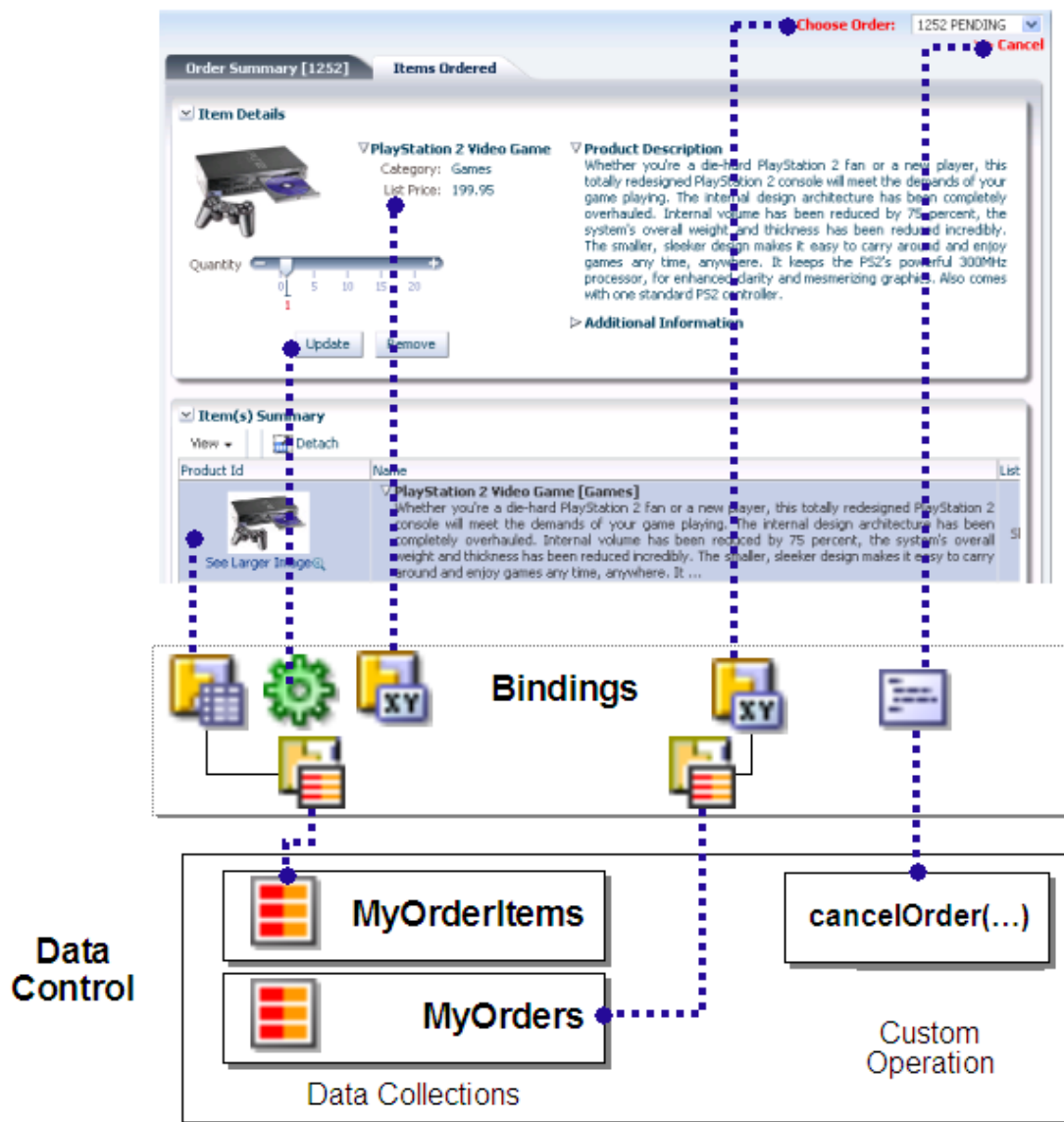
An additional feature of ADF is the ability to expose the collection of views as a service so it can be consumed by other applications; thus reusing code and enforcing same business rules across multiple applications.

As for Oracle Forms, the Form itself is an application module because it is a collection of business rules and transactions that are combined together to archive a business requirement.

#### What is a Data Binding?

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

Data binding is a connection between UI components to data control exposed via the application module. Once the UI consumes a view object or method exposed in the data control, a binding is created between the Model layer and View layer. The following diagram illustrates the data binding concept.



### Object Scope Lifecycles

When you run your application, parameters are usually passed to the application and that data is stored in an object scope. Once you place an object in a scope, it can be accessed from the scope using an expression language. For example, you might create a managed bean named `myPageRequestBean` and define the bean to live in the Request scope.

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

To access that bean, you would use the expression language `#{requestScope.myPageRequestBean}`. For a Forms Developer, this is a way of calling the procedure or function in a database package.

There are three types of scopes in a standard JSF application:

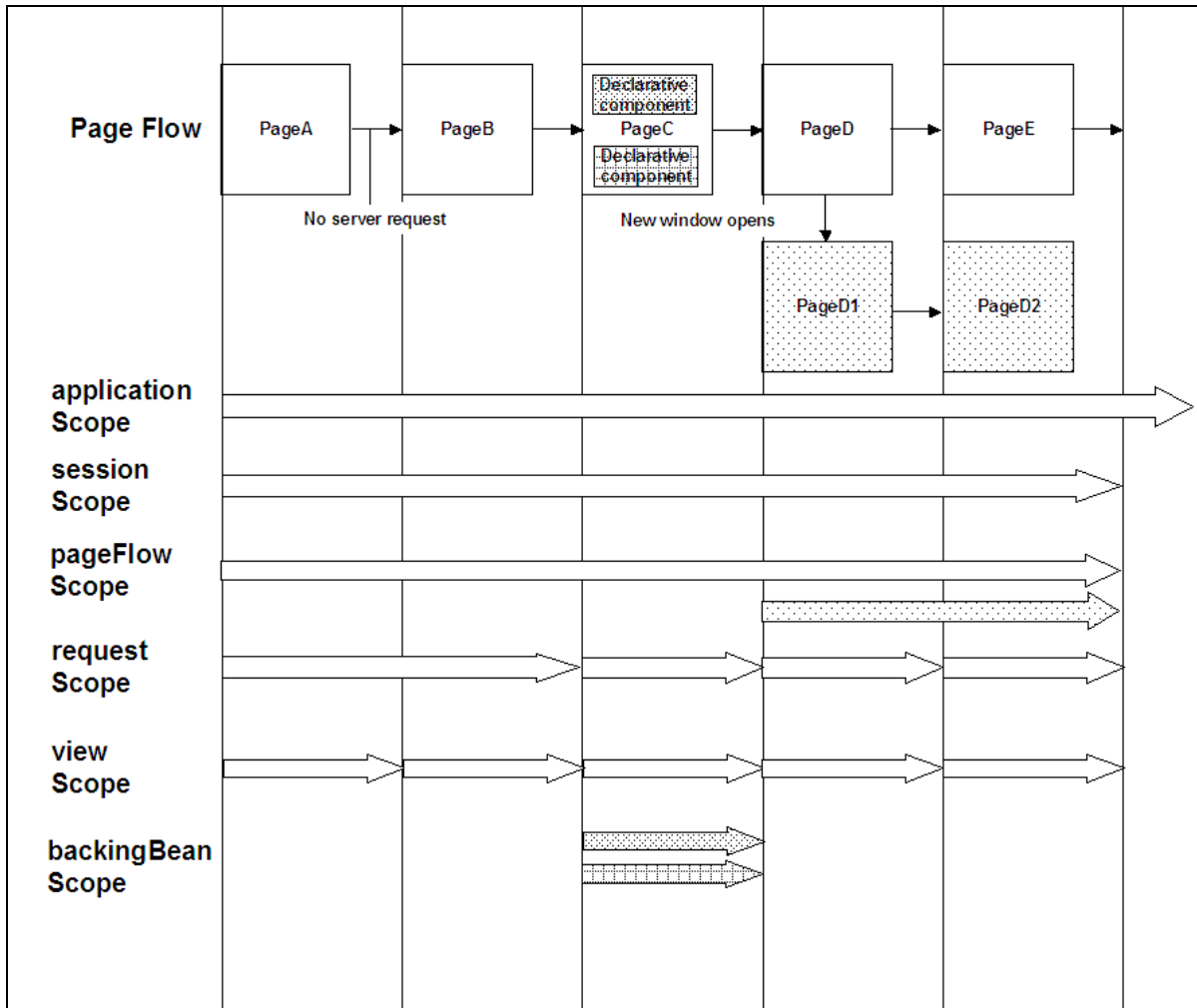
applicationScope	The object is available for the duration of the application.
sessionScope	The object is available for the duration of the session.
requestScope	The object is available for the duration between the time an HTTP request is sent until a response is sent back to the client.

In addition to the standard JSF scopes, ADF Faces provides the following additional scopes:

pageFlowScope	The object is available as long as the user continues navigating from one page to another. If the user opens a new browser window and begins navigating, that series of windows will have its own pageFlowScope. Store your application parameters in this scope.
backingBeanScope	Used for managed beans for page fragments and declarative components only. The object is available for the duration between the time an HTTP request is sent until a response is sent back to the client. This scope is needed because there may be more than one page fragment or declarative component on a page, and to avoid collisions between values, any values must be kept in separate scope instances. Use backingBeanScope for any managed bean created for a page fragment or declarative component.
viewScope	The object is available until the ID for the current view changes. Use viewScope to hold values for a given page.

Object scopes are analogous to global and local variable scopes in programming languages. The wider the scope the higher the availability of an object. During their lifespan, these objects may expose certain interfaces, hold information, or pass variables and parameters to other objects.

## Relationship between Scopes and Page Flow



For your applications, you will most likely choose the `pageFlowScope` to store your Task Flow parameters. These variables are stored in a hash map that can be referenced in the UI managed bean or in the Application Module Java Methods. Referencing the hash map object related to a specific object scope is one way to reference variables between the Model layer and View layer in your application. In your application, you will most likely use `backingBeanScope` to store your methods used by your page fragments.

### Expression Language

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

In Java Server Faces (JSF), you can use a simple expression language (EL) to access application data stored in Java Bean components. The syntax of EL is as follows:

```
#{<Binding or Bean Name>.<Variable/Method> <Operation>}
```

For example, if we want to display a credit card number field only when the payment method is “Credit Card”, then we could apply the following EL to the visible property on the credit card field in the UI:

```
{bindings.PreferredPaymentMethod.attributeValue == 'Credit Card'}
```

In the reference document below, the tutorial uses the \$ sign instead of the # sign. For our ADF development, we will be using the # sign. The following explains the difference between the \$ and # sign.

\$ - \$ syntax executes expressions eagerly/immediately, which means that the result is returned immediately when the page renders.

# - # syntax defers the expression evaluation to a point defined by the implementing technology. In general, JSF uses deferred EL evaluation because of its multiple lifecycle phases in which events are handled. To ensure the model is prepared before the values are accessed by EL, it must defer EL evaluation until the appropriate point in the lifecycle. I have provided a EL tutorial:

The J2EE 1.4 Tutorial	<a href="http://docs.oracle.com/javaee/1.4/tutorial/doc/JSPIntro7.html">http://docs.oracle.com/javaee/1.4/tutorial/doc/JSPIntro7.html</a>
-----------------------	---

Note: Some Developers try to avoid using expression language statements because the statement is checked at runtime as opposed to a managed bean where the statement is checked at compile time. It is similar to calling dynamic SQL where you don't know the value of the variable you are referencing until runtime.

### Managed Bean

At some point in your application, you will need a more powerful tool than expression language (EL) to programmatically modify the UI or control the behavior or your application so you will need to create a managed bean. A managed bean is a reusable software component for Java that represents a manageable resource including components, applications or devices. They are used to encapsulate many objects into a single object (the bean), so that they can be passed around as a single bean object instead of as multiple individual objects.

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

Managed beans are Java classes that you register with the application using various configuration files. When the JSF application starts up, it parses these configuration files and the beans listed within them are made available. The managed beans can be referenced in an EL expression, allowing access to the beans' properties and methods.

Below is an example of a change listener method that is controlled by a managed bean:

```
public void PreferredPaymentMethodChangeListener(ValueChangeEvent valueChangeEvent) {

    valueChangeEvent.getComponent().processUpdates(FacesContext.getCurrentInstance());
    Integer newValue = (Integer) valueChangeEvent.getNewValue();

    BindingContext bctx = BindingContext.getCurrent();
    BindingContainer bindings = bctx.getCurrentBindingsEntry();

    // Get The List Of Values Related To The Selected Payment Method
    JUCtrlListBinding list = (JUCtrlListBinding) bindings.get("PreferredPaymentMethod");

    // Find The Selected Values In The List Of Values
    Row selectedRow = (Row)list.getSelectedValue();
    String selectedValue = list.getAttributeValue().toString();

    if (selectedValue.equals(SiConstants.CreditCard)){
        // Turn On Credit Card Panel Group
        this.creditCardPanelGroupLayout.setVisible(Boolean.TRUE);
    } else {
        // Turn off Credit Card Panel Group
        this.creditCardPanelGroupLayout.setVisible(Boolean.FALSE);
    }
}
```

As comparison to Oracle Forms, a managed bean would contain all the PL/SQL related UI behavior. For example, we can display or hide the credit card number field based on a When-Validate-Item of the payment method field.

Note: When you create your managed bean you need to specify a scope for the bean. (See Object Scope Section).

### Where Should Your Custom Code Reside?

There are various Java classes and “levels” to write your custom code and based on the requirement it should be written at a certain “level”. When I first started building applications, I was never quite sure where to place my custom code. I used the following guide to help me determine where to write the code.

Location	Guide
Application Module	Application module class as the place where you can write your service-level application logic.  Execute database command or block of PL/SQL Access Existing View Objects Access DBTransaction Object
EntityImpl Class	The EntityImpl class is the base class for entity objects, which encapsulate the data, validation rules, and business

	<p>behavior for your business domain objects.</p> <ul style="list-style-type: none"> <li>• Get an attribute</li> <li>• Set an attribute</li> <li>• Validate data before posting to the database</li> <li>• doDML Operations (Like Triggers)</li> </ul> <pre>// In PLSQLEntityImpl.java protected void doDML(int operation, TransactionEvent e) { // super.doDML(operation, e); if (operation == DML_INSERT) callInsertProcedure(e); else if (operation == DML_UPDATE) callUpdateProcedure(e); else if (operation == DML_DELETE) callDeleteProcedure(e); }</pre> <p>In most cases, Developers user DML operations to set “who” columns and retrieve database sequences. For example, translate the user login information to a User Id.</p> <p>Note: Add custom code to set a value after it is retrieved from the database and before it is passed to the view object or to override the view object value before storing the value into the database.</p>
ViewObjectImpl Class	<p>A view object is a base class for view objects</p> <ul style="list-style-type: none"> <li>• Set Where By Clause</li> <li>• Set Order By Clause</li> </ul> <p>For Form Developers, this is where your block triggers exists.</p> <p>Note: I have also seen Developers set the Where By and Order By in an Application Module method</p>
ViewRowImpl Class	<p>A view object is a base class for view row objects</p> <ul style="list-style-type: none"> <li>• Get Attribute</li> <li>• Set Attribute</li> </ul> <p>Note: I normally use this class to set transient values attributes before they are rendered to the page and to process values from the page before passing the row-set back to the Entity class.</p>

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

Managed Bean	<p>This is where you store all your UI based code that controls the page as well as call methods in the application module.</p> <ul style="list-style-type: none"><li>• Initialize code for your session</li><li>• Call methods defined in the application module</li><li>• Conditionally Rendering</li><li>• Set variables</li></ul> <p>For Form Developers, this is where all your UI events triggers</p>
--------------	---

Note: Oracle provides a comprehensive list of what typical code is placed in each area:

Document	Chapter
Developer's Guide for Oracle Application Development Framework	Most Commonly Used ADF Business Components Methods

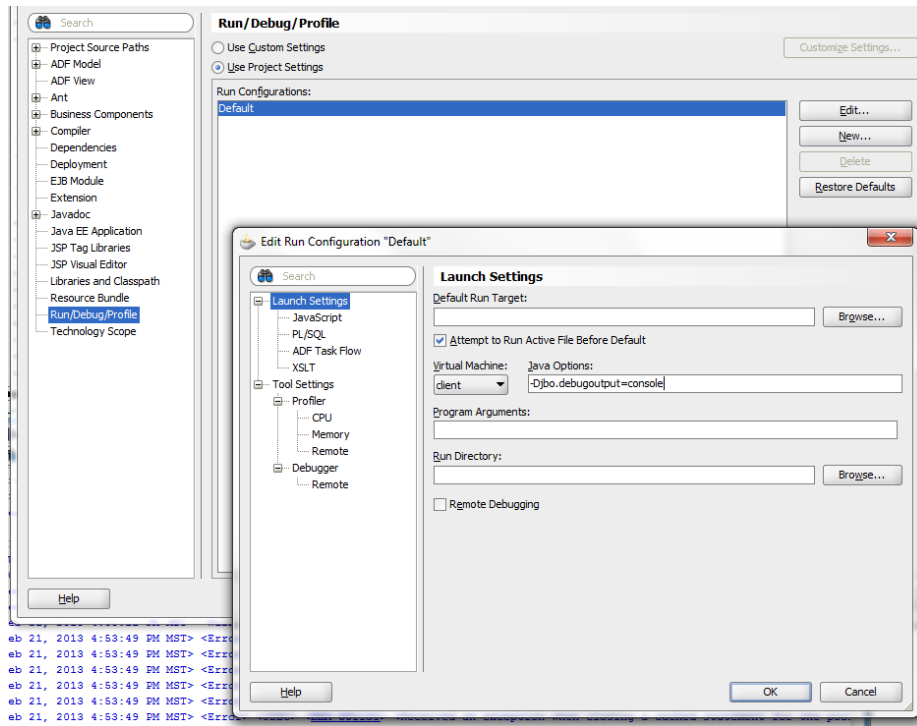
### Debugging (JDeveloper Debug / SOP / ADF Logger)

ADF and JDeveloper provide several ways to debug your code.

1. System.out.println, or SOP is equivalent to dbms\_output in PL/SQL. It provides debugging capability while developing your application but ADF Logger should be used when the application is deployed to production. By default the SOP messages are displayed in the Log Window (Figure A: JDeveloper Navigation).
2. ADF Logger is a logging mechanism, which is built into the ADF framework. It is a wrapper around the java.util.Logging APIs with a few convenience methods thrown in, and most importantly, some specific features integrated into both JDeveloper and Enterprise Manager.
3. The JDeveloper tool has a few ways to debug your application.
  - a. When you want to debug your application in detail, then add the following string to your application: -Djbo.debugoutput=console and run your application in debug mode. The debug messages will be displayed in the Log Window (Figure A: JDeveloper Navigation).



## The Basics of ADF and JDeveloper From an Oracle Forms Perspective



- b. ADF Declarative Debugger provides declarative breakpoints that you can set at the ADF object level (such as task flows, page definition executables, method and action bindings, ADF lifecycle phases), as well as standard Java breakpoints. How to use the debugging utility can be the entire presentation, so reference the ADF Application Developers Guide – Chapter “31.7 Using the ADF Declarative Debugger” to learn how to use this feature.

### Naming Conventions

You should determine how to name your packages and the granularity.

Example Base: com.<application short name>.<project name>

Sample: com.xx.myproject

### UI

Default Package	com.xx.myproject.view
Task Flows	com.xx.myproject.view.pageflows
Pages	com.xx.myproject.view.pages
Page Fragments	com.xx.myproject.view.pagefrgments
Managed Beans	com.xx.myproject.view.bean
Overriden Component Class	com.xx.myproject.view.overriden
Page Definitions	com.xx.myproject.view.pagedefs

## Model

Default Package	com.xx.myproject. <b>model</b>
Entities	com.xx.myproject. <b>model.entities</b>
Associations	com.xx.myproject. <b>model.assoc</b>
View Links	com.xx.myproject. <b>model.vo.link</b>
Updatable Views	com.xx.myproject. <b>model.vo</b>
Read Only Views	com.xx.myproject. <b>model.vo.readonly</b>
Application Module (AM)	com.xx.myproject. <b>model.am</b>
Diagram	com.xx.myproject. <b>model.diagram</b>

## How to set up your local PC to develop ADF applications?

Now that you have been introduced to the concept of ADF and JDeveloper, it is time to set up your local PC. You can find the download page by referencing the JDeveloper home page:

Oracle JDeveloper Main Page	<a href="http://www.oracle.com/technetwork/developer-tools/jdev/overview/index.html">http://www.oracle.com/technetwork/developer-tools/jdev/overview/index.html</a>
-----------------------------	---

### What Version?

There are several versions of JDeveloper to download, so choose the correct version for your environment. There is a one-to-one relationship between JDeveloper and Weblogic Server and you have to install a JDeveloper with the same ADF runtime version as the Weblogic Server. There is no backward or forward compatibility. EBS Forms Developers are familiar with this concept because they have to install the compatible Oracle Forms software related to their EBS technology stack.

If you are NOT planning on deploying to a Weblogic Server (WLS), then download the latest JDeveloper version. In all other cases, reference the compatibility link below:

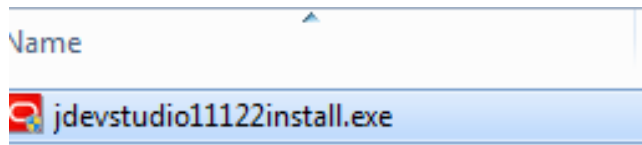
ADF Runtimes vs WLS versions as of JDeveloper 11.1.1.6.0	<a href="https://blogs.oracle.com/onesizedoesntfitall/entry/adf_runtimes_vs_wls_versions">https://blogs.oracle.com/onesizedoesntfitall/entry/adf_runtimes_vs_wls_versions</a>
--	---

### Installation

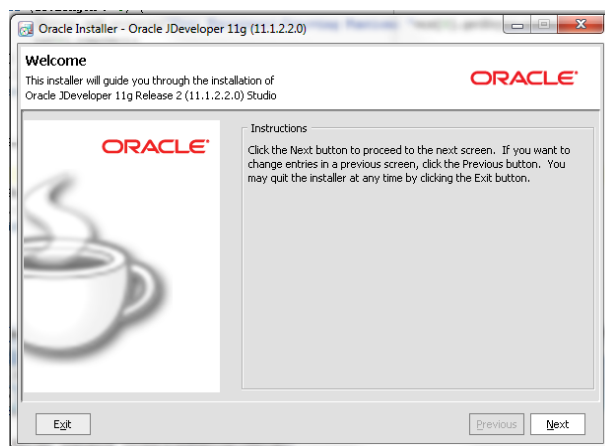
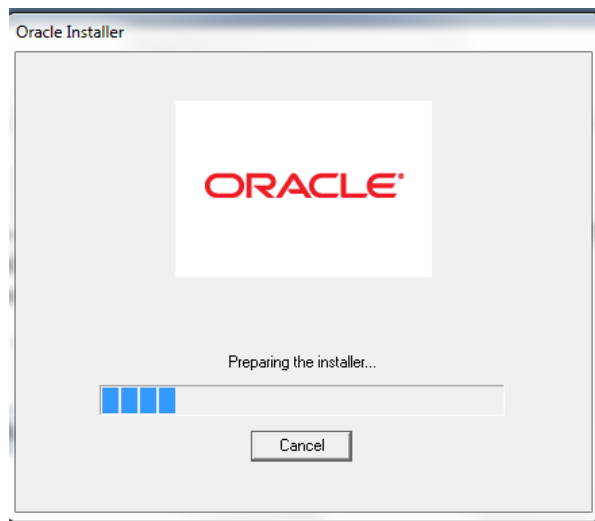
Oracle provides detailed instructions on how to install a local version of. I will just highlight a few steps:

1. Software - Download the correct version of the software (See What Version?)
2. Prerequisites – Administrative Access
3. Installation is quite simple

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

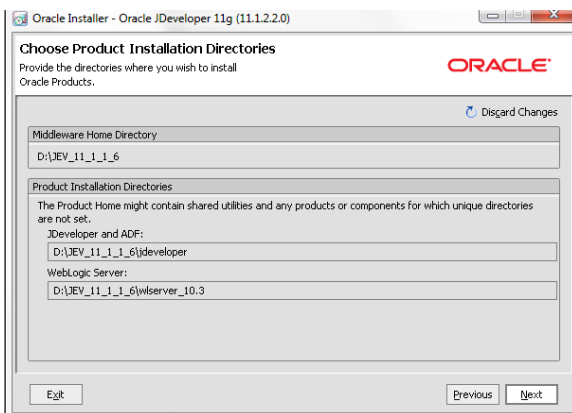
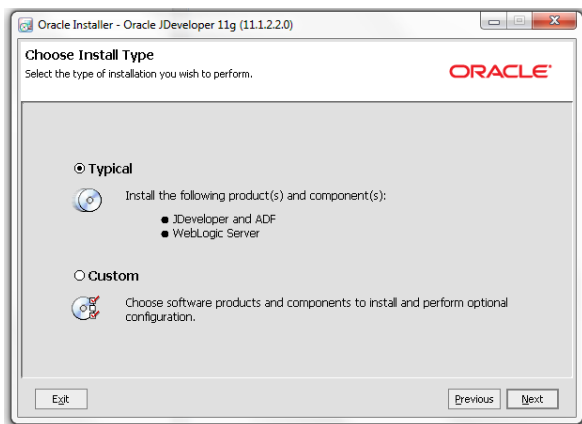
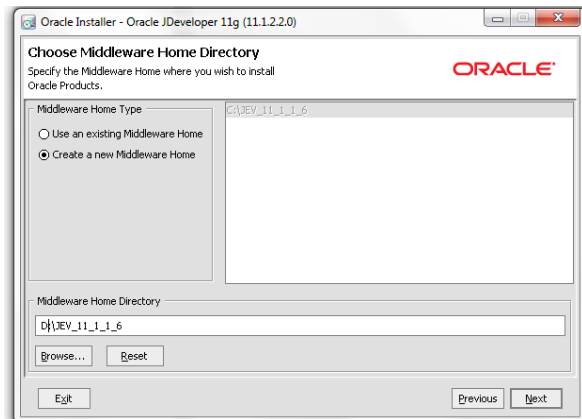


Click On jdevstudio11122install.exe

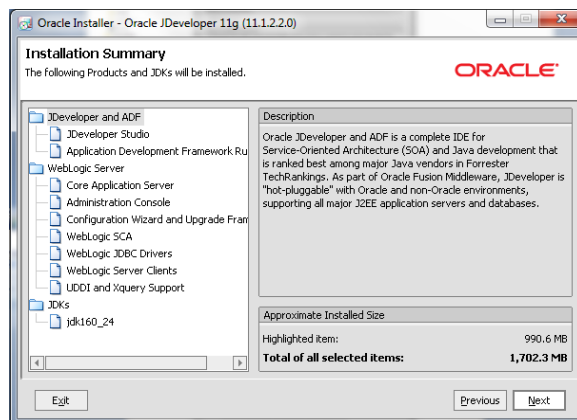
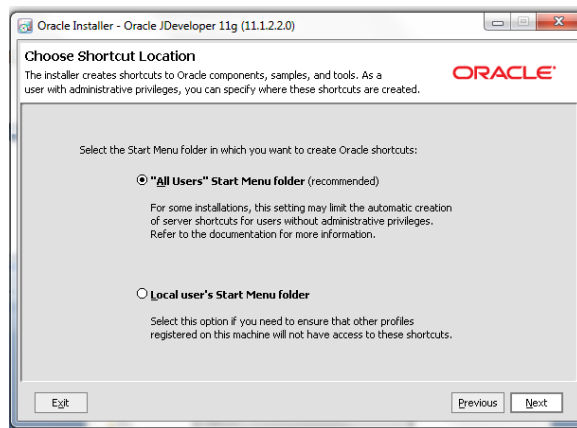


It is recommended you always install each JDeveloper installation in its own directory.

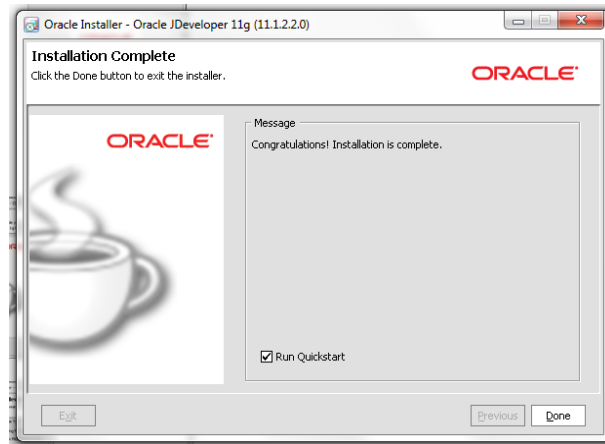
# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



If you need additional installation references, then refer to the following references:

Oracle Install Guide – Screen Shots	<a href="http://docs.oracle.com/cd/E35521_01/install.111230/e17074/toc.htm">http://docs.oracle.com/cd/E35521_01/install.111230/e17074/toc.htm</a>
Oracle Install Guide – Video	<a href="https://blogs.oracle.com/workingwithadf/entry/installing_and_configuring_jdeveloper_with_adf">https://blogs.oracle.com/workingwithadf/entry/installing_and_configuring_jdeveloper_with_adf</a>

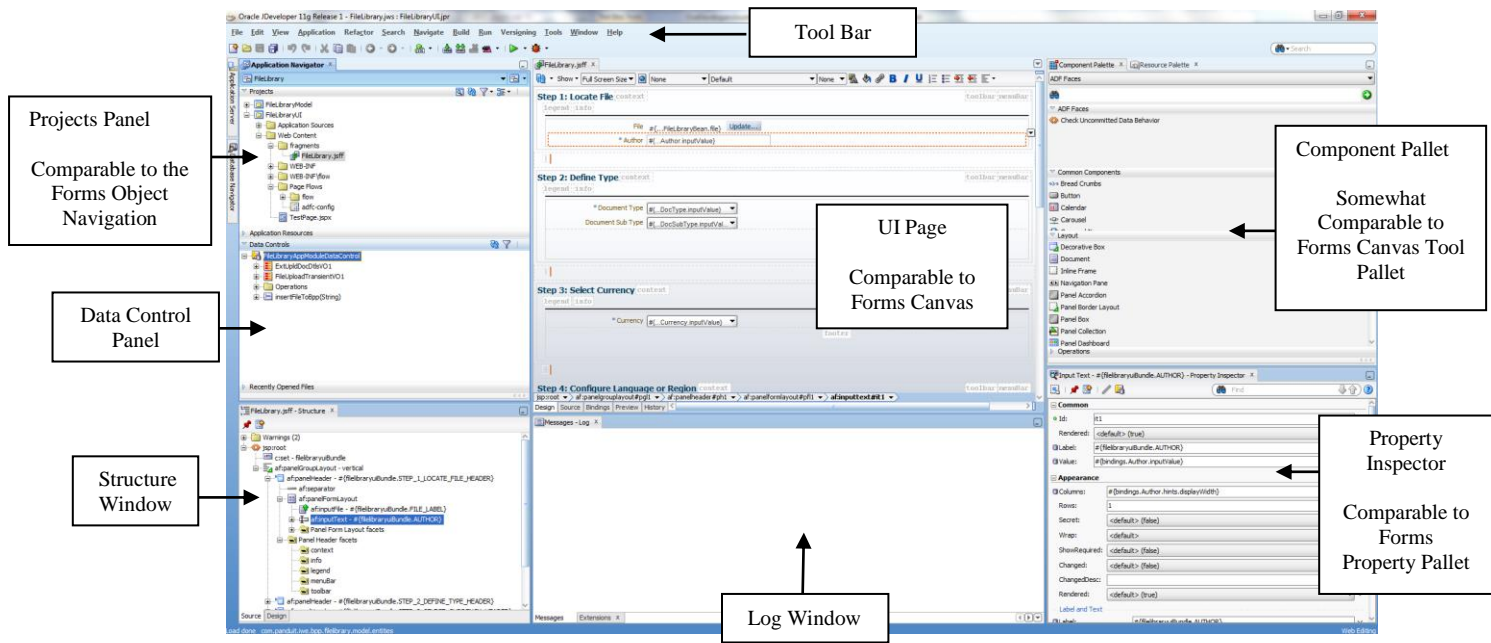
## Building Your First Applications

Now that you have installed JDeveloper, take a few hours navigating within the tool and review the Oracle tutorial on how to build an ADF application.

1. Understand basic navigation within the JDeveloper tool.

Figure A: JDeveloper Navigation

# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



Here are some navigation short-cuts:

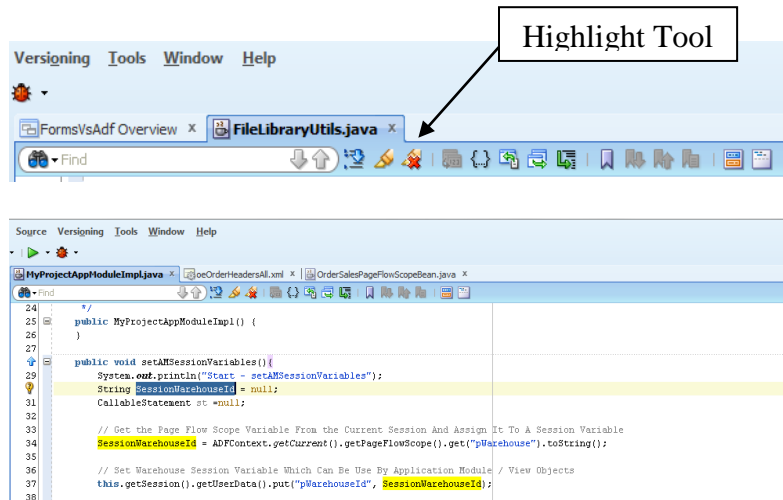
Ctrl + Mouse Click	Go To Declaration (Find view associations)
Ctrl + -	Go To Class
Ctrl + Alt + -	Go To File (Used when it's a large project)
Alt + Home	Locate file in app navigator
App Navigator	Find as you type (Similar to Oracle Forms)
Ctrl + F	Find
Ctrl + Shift + F	Find In Files (Similar to "Find" in Oracle Forms)
Ctrl + Alt + H	Highlight Fields

Note: In JDeveloper Version 11.1.1.6, the associations are not visible in the declarative entity object so open the source tab and search for "AccessorAttribute". Once you find the association section, then hold down the Alt key on the accessor line, and next click on the hyperlink to view the association.

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective



Highlight is a great way to find text within in your custom code and I use this feature all the time.



2. Review an Oracle tutorial on how to build your first application. I referenced this URL multiple times when I was building the first applications.

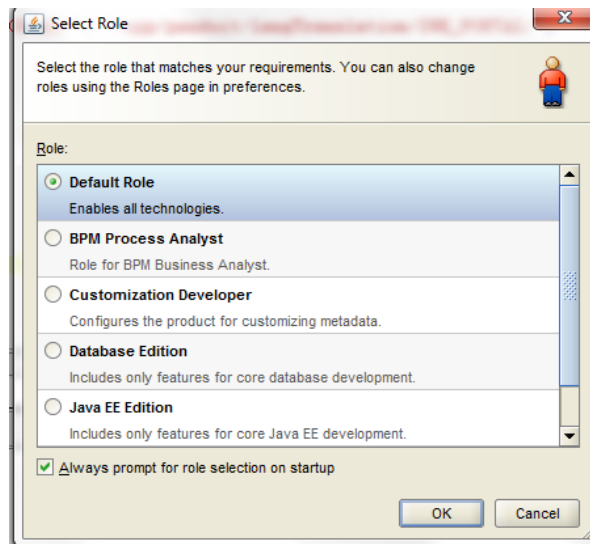
Developing Rich Web  
Applications With Oracle ADF

[http://docs.oracle.com/cd/E18941\\_01/tutorials/jdtut\\_11r2\\_55/jdtut\\_11r2\\_55\\_1.html](http://docs.oracle.com/cd/E18941_01/tutorials/jdtut_11r2_55/jdtut_11r2_55_1.html)

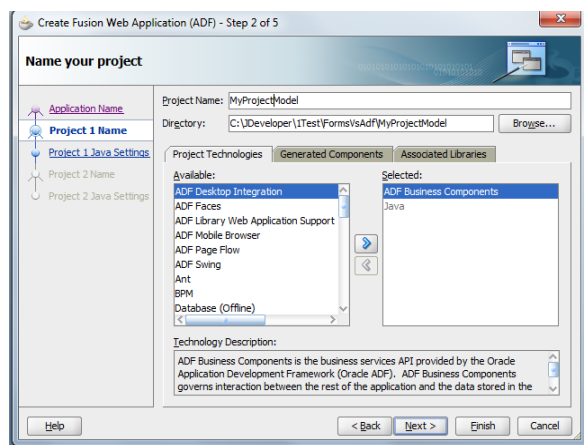
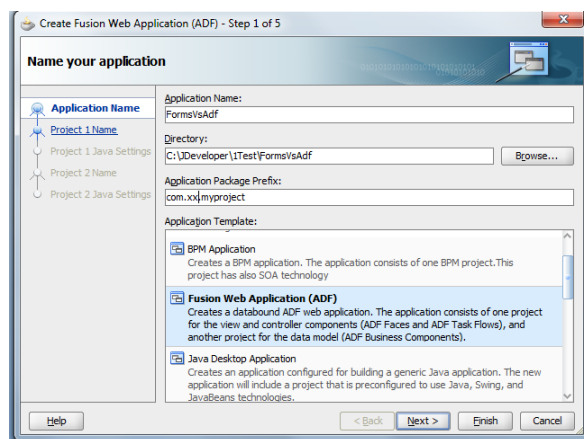
Now it is time to start building your first application. Once you start JDeveloper, it will prompt you to select a role. A role is a way to only display tools and components related to your project. Select the default role until you have researched how this option works.



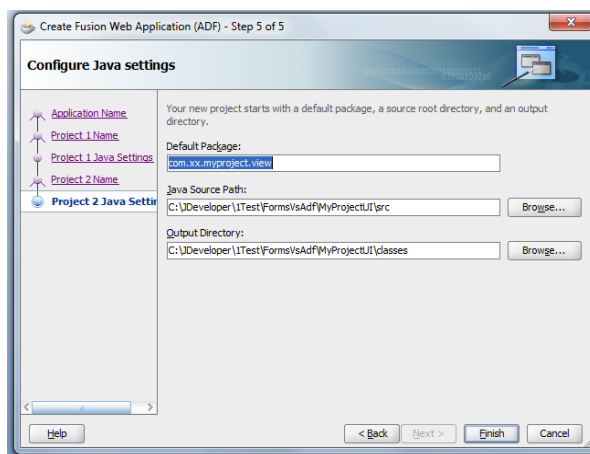
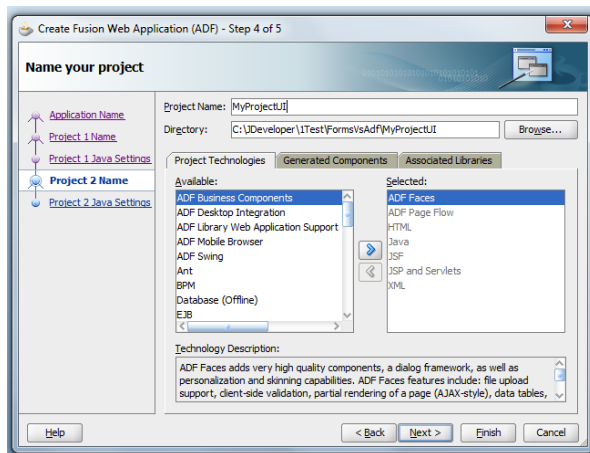
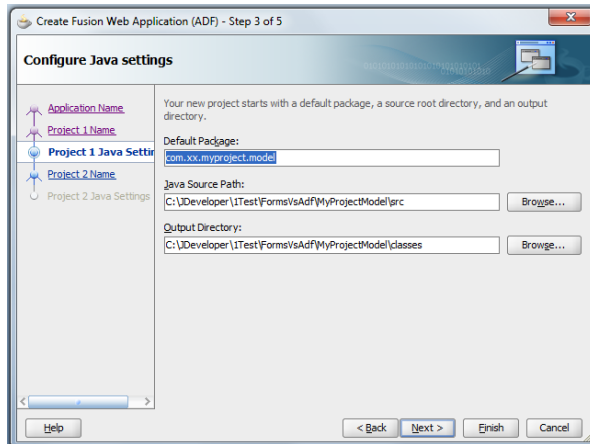
## The Basics of ADF and JDeveloper From an Oracle Forms Perspective



Create a new application:



## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

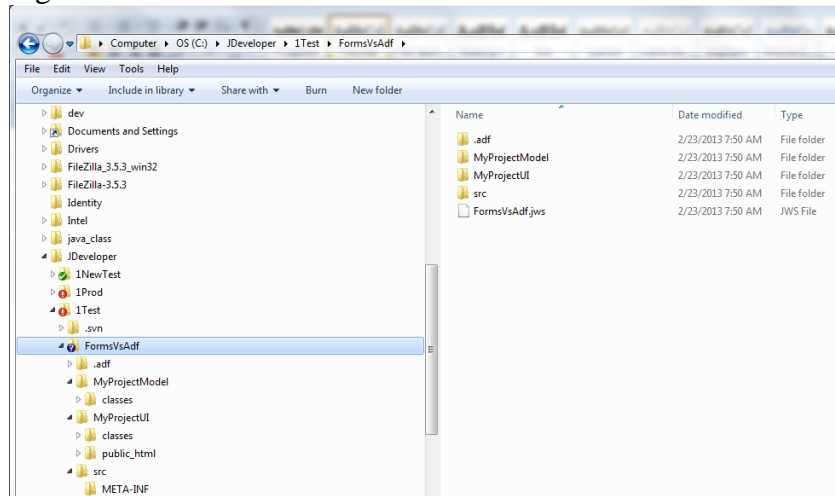


Once you click on finish, the application will set up the directory structure on your file system and open up the application with a Quick Start Guide. This is comparable to the Oracle Forms Wizard that helps you build your application. I would recommend using

# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

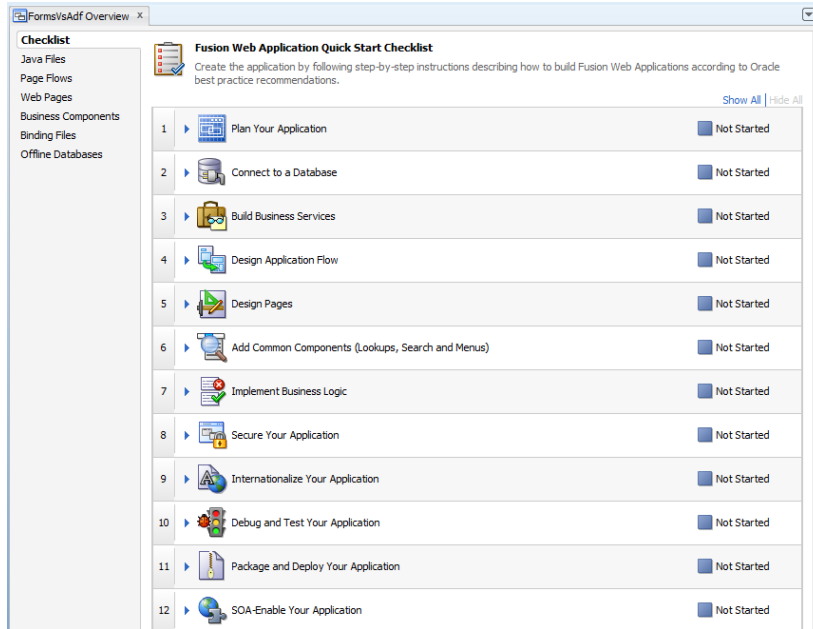
the ADF Quick Start Guide to build the first applications. As you learn the JDeveloper tool, you will probably build your application without the Wizard.

Figure B: File Structure



## Quick Start Guide

# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



Now it is time to back up your application. If you ever worked with Oracle Reports or Oracle Forms, the first thing you learn is to back-up your code often. This same holds true for Oracle ADF. There are so many additional options that you are bound to make a mistake that will require you to restore from a previous version. The simple way to back up your code is to copy your application directory structure (Figure B: File Structure) to a back-up directory. Another way is to use a Third-Party tool, Subversions, which is integrated into JDeveloper.

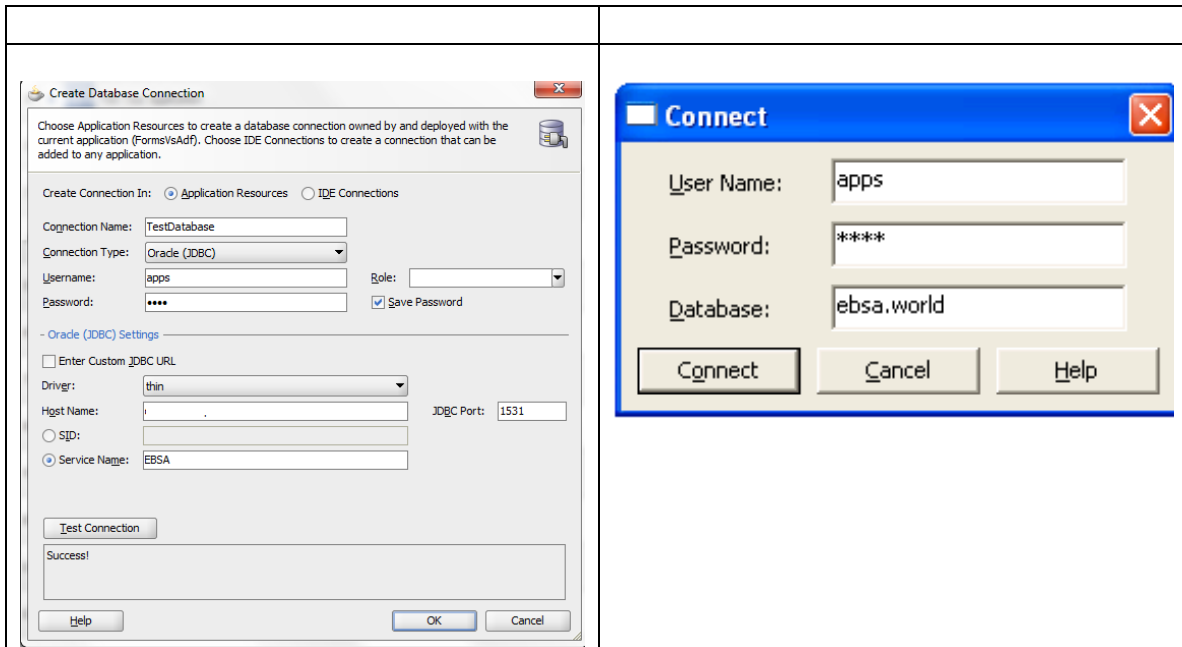
In this next section, I will attempt to draw some comparisons from ADF to Oracle Forms.

## Database Connection

JDeveloper and Oracle Developer Forms both require a database connection to access a database schema. ADF also has the ability to connect to multiple databases as well as to other sources of data (for example, BAM, BPM MDS, Content Repository, External Application, RIDC, SOA-MDS, URL, Worklist, WSIL). For this presentation, we are only concerned, with a database connection to a single database and schema.

JDeveloper	Oracle Forms Developer
------------	------------------------

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective



### Build Business Services (Entity Objects / Views / Links / Application Module)

ADF and Oracle Form applications are typically built to assist in managing data that resides in a database. Both ADF and Oracle Forms framework support, create, update, read and delete (CRUD) operations; however, there is one major difference. ADF automatically handles CRUD operations; whereas, Oracle Forms uses the Forms record manager to store records retrieved from the database but it is the responsibility of the Developer to manage, create, update and delete (CUD) operations. In Oracle Forms, CUD operations are executed using Data Block triggers and PL/SQL code.

In ADF, CRUD operations are performed on entity objects or database tables. This is the reason why ADF requires a separate updatable view objects to be used by the UI. Entity objects are used by the ADF framework to automatically handle CRUD operations and view objects are the mechanism to maintain the data in your user interface. There are two types of view objects: Updatable and Non Updatable. If the view object is updatable, then there will be an entity object definition defined in that view object. Any view object that needs to be consumed by UI (pages) will be packaged into an application module so it can be exposed to your UI (pages).

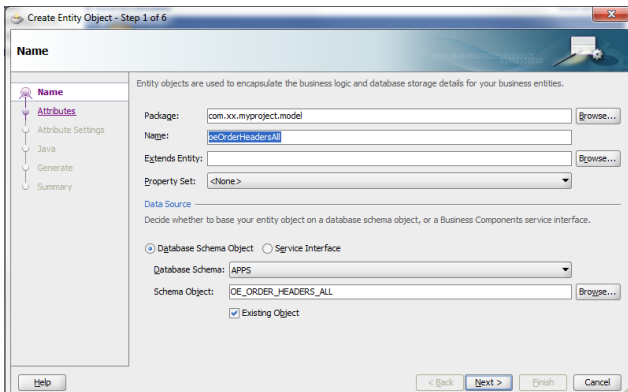
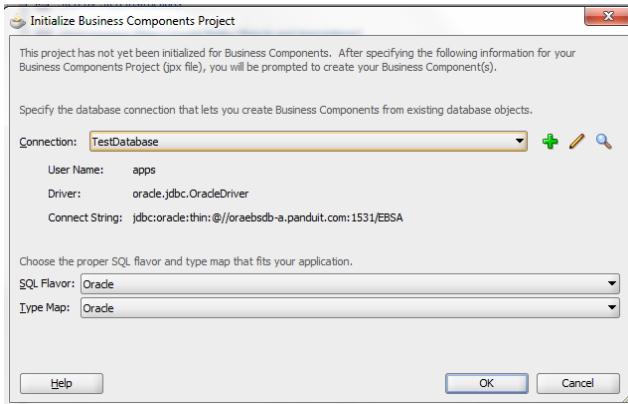
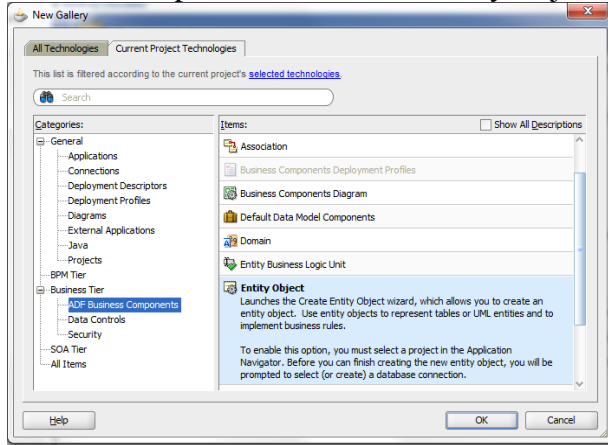
In terms of Oracle Forms, it can be argued that each Oracle Form application contains a virtual application module because it is a collection of view objects that are implicitly exposed to the user interface or canvases.

### Creating Business Service Objects

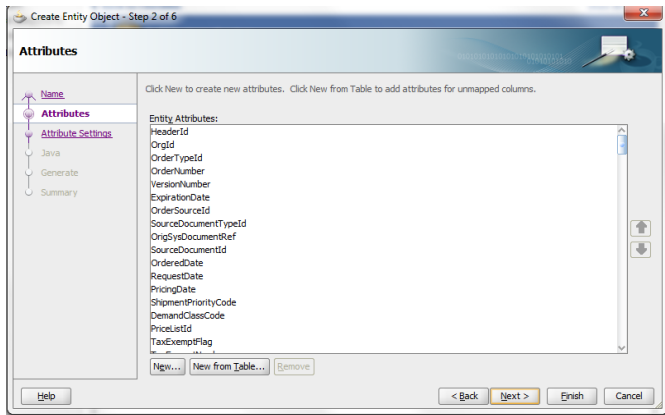
In order to assist you in creating Entity and View objects, JDeveloper has created various wizards to reduce the time to build your application. When you need to create an Entity object, place your cursor on the Model, or "MyProjectModel" and right click and select

# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

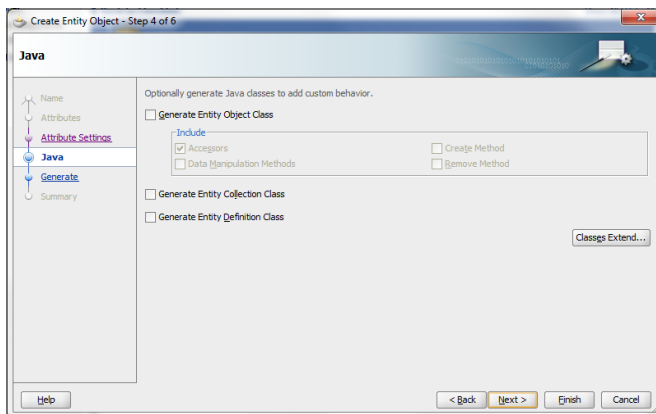
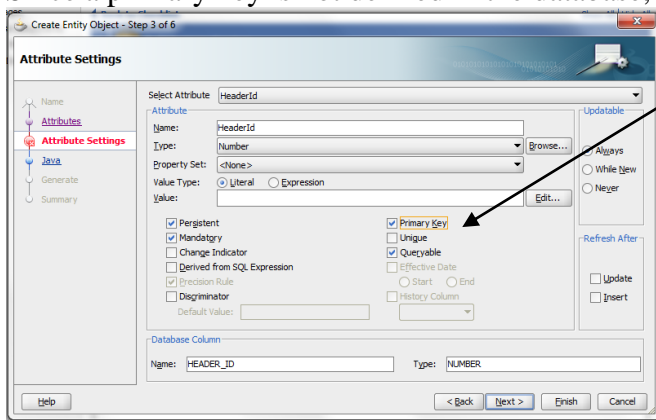
“New”. A pop-up gallery window will then appear. Select Business Tier → ADF Business Components → “Create Entity Object Wizard” to launch the entity wizard.



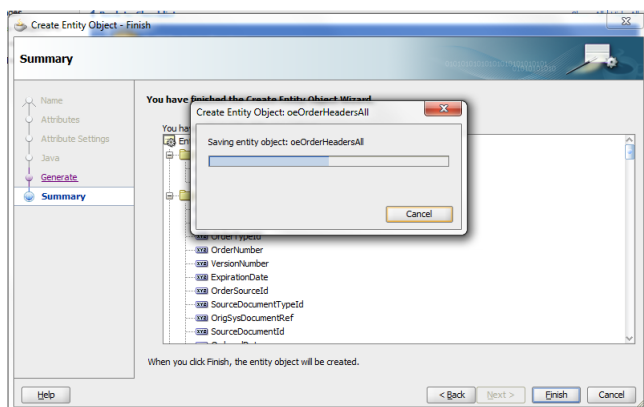
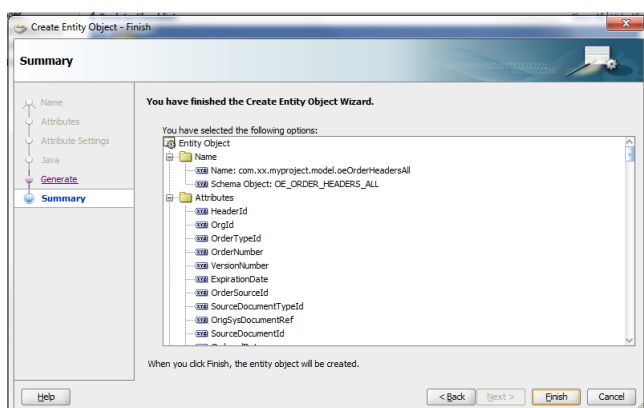
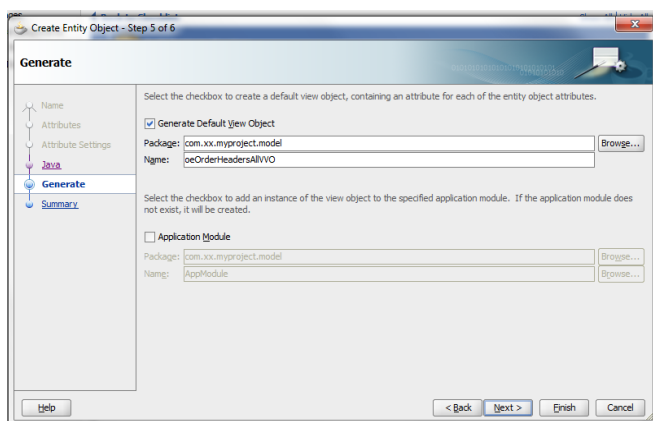
# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



Since a primary key is not defined in the database, select primary key.



# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

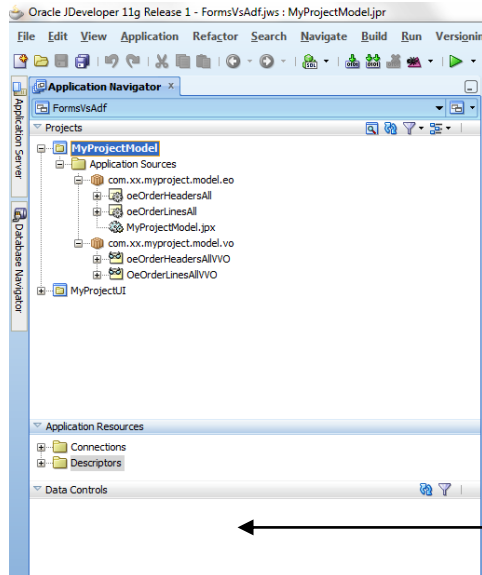


Perform the same steps for oe\_order\_lines\_all



# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

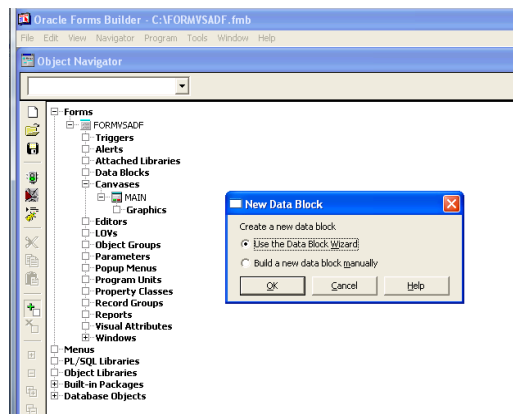
When you are done, your project should look like



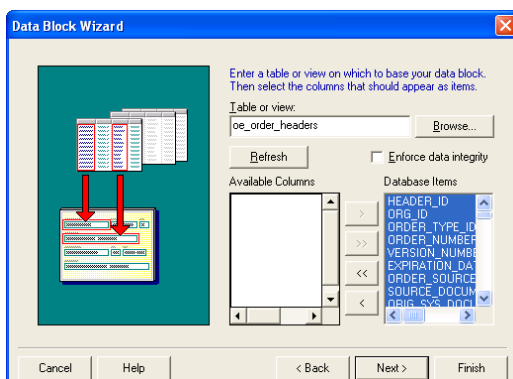
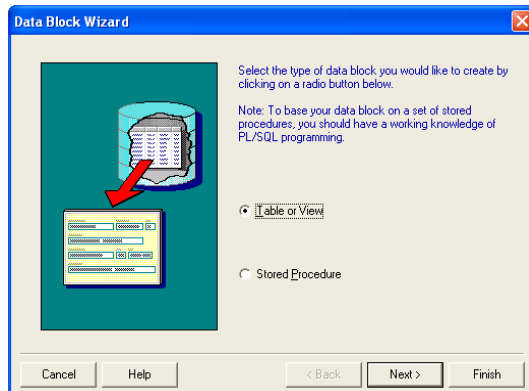
We have not exposed any view objects so nothing is showing under data controls

In Oracle Forms, database view objects perform roughly the same function as an ADF view object. It is the mechanism to maintain the data in your user interface. As with ADF framework, Oracle Forms framework has wizards to create your Data Block (view objects):

Create data blocks (entity & view objects).

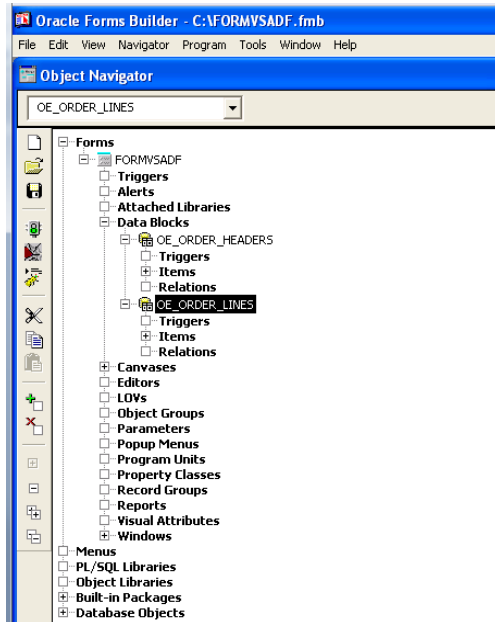


# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

Perform the same steps for oe\_order\_lines and when you are done the Forms navigator screen should look like



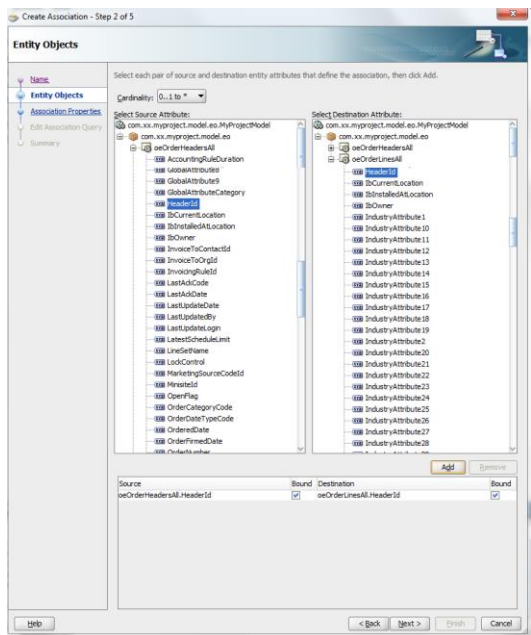
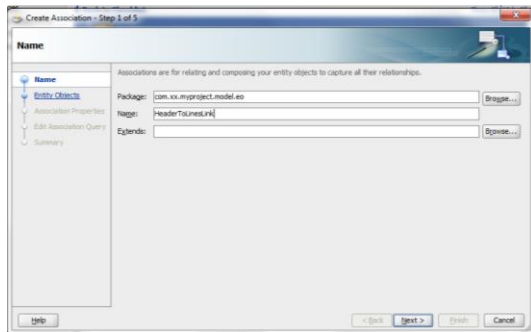
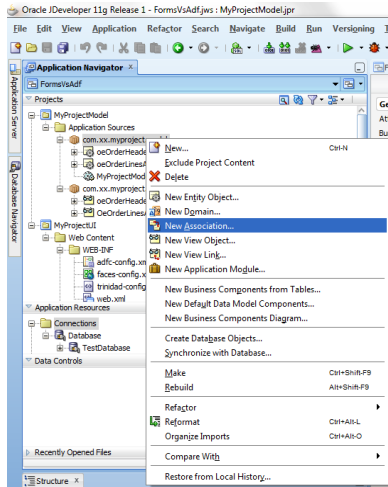
In ADF, the next step is to create a master-child relationship between the headers and lines table. The relationship is connected by a database column or header\_id. ADF supports two different styles of master-detail relationship:

**Association** This is a master-detail relationship defined at the entity level and since it is defined at the entity level, each entity object can reference each other in CRUD operations. As a general rule, create an association link when you need to enforce business rules, create or delete records in a master detail relationship. Note: If you have primary-foreign key relationship defined in your database, then an association link will automatically be created for you.

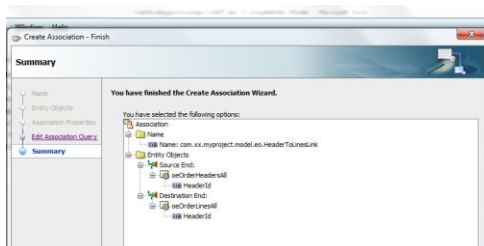
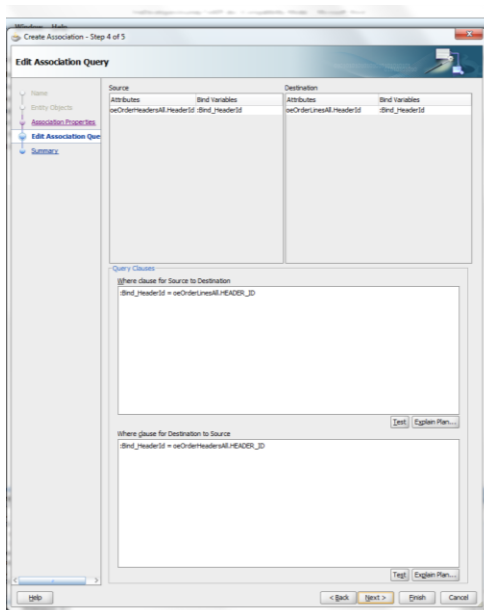
**View Link** This is a master-detail relationship defined at the view level and is mainly used to sync up blocks of data in your application. If you are creating a master-detail relationship screen in your application, then you should create a view link because it supports automatic master-detail synchronization. If you have association, then use the association to link the two objects in the UI.

# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

Here are the steps to create an association link

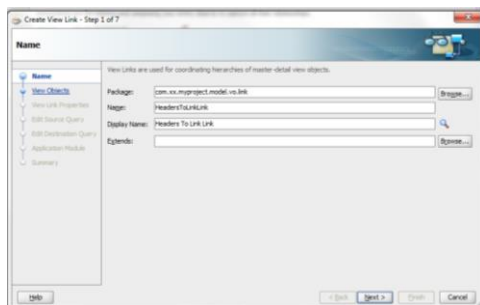
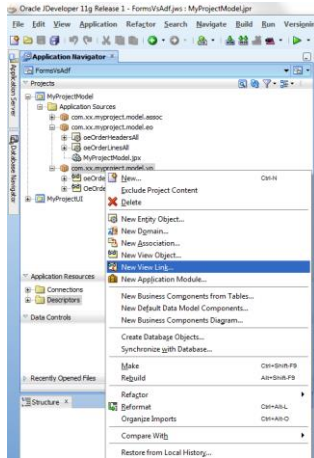


# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

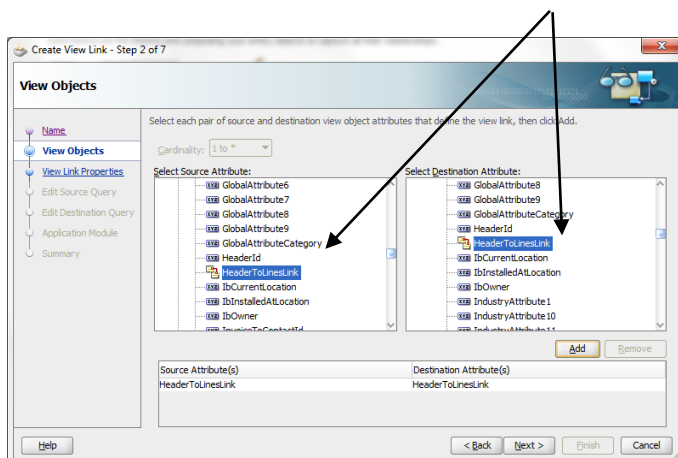


# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

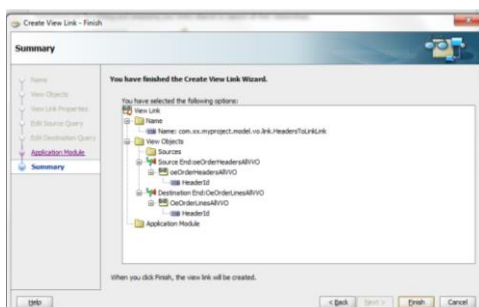
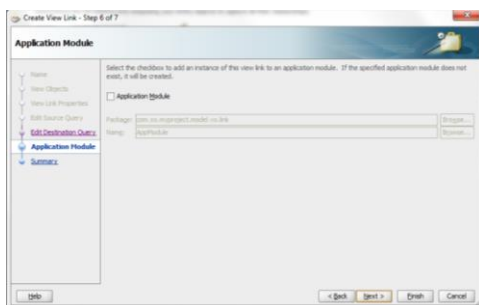
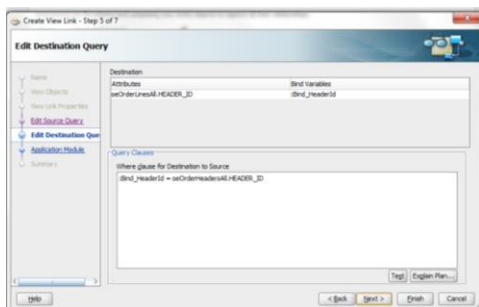
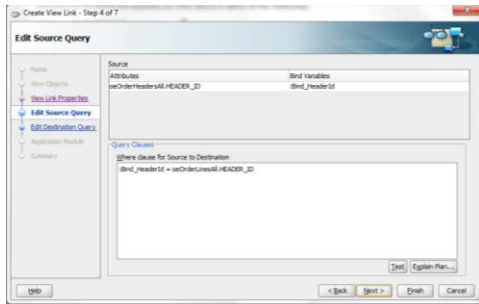
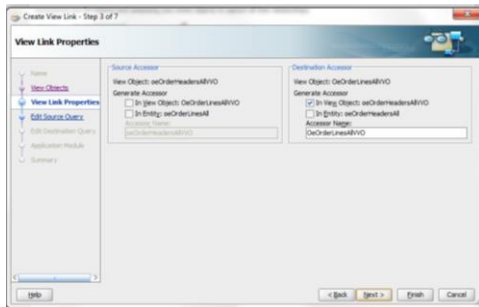
Here are the steps to create a view link.



Note: I am using the association in the view link

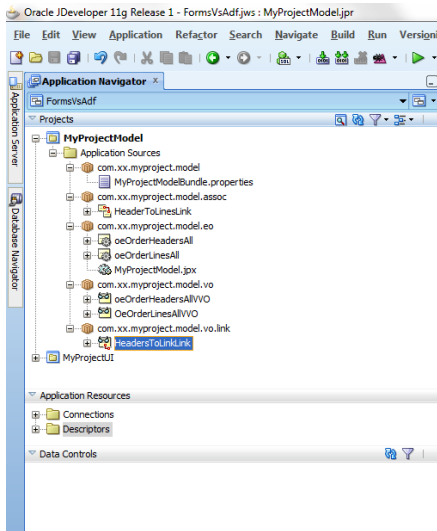


# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



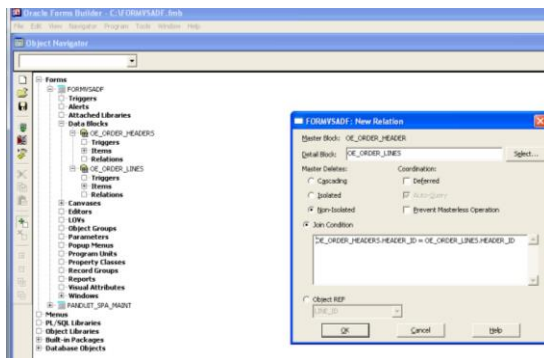
# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

When you are done defining the links, your project should look as follows:



In Oracle Forms, we do not have a distinction between table-based and view link master-detail relationship. We have a single relationship, which is a combination of an association and view link. It supports auto-query of the detail block and enforces cascade deletes.

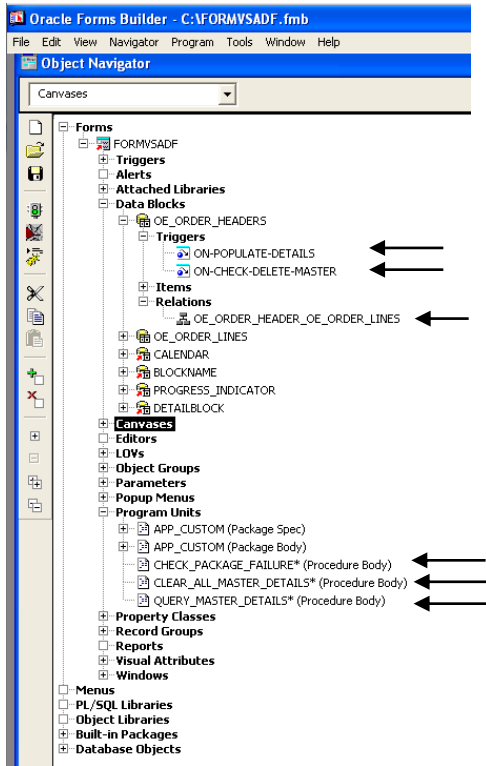
Create a relationship between headers and lines (association and view links).





## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

Once the relationship is created, Oracle Forms will automatically add code to enforce the relationship.

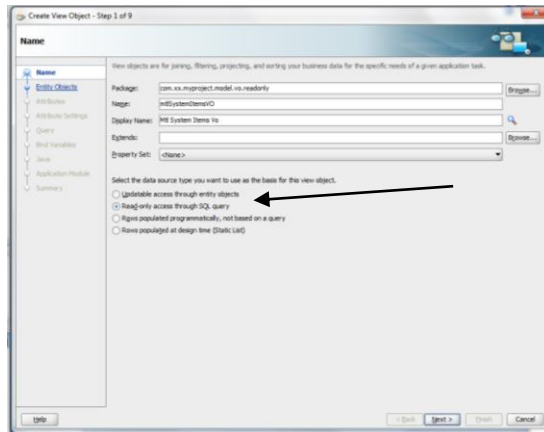


Now that we have our business relationships defined, we need to apply a common business functionality to help our Users understand the data. One such business component is a list of values (LOV) component, which translates database Ids to readable values.

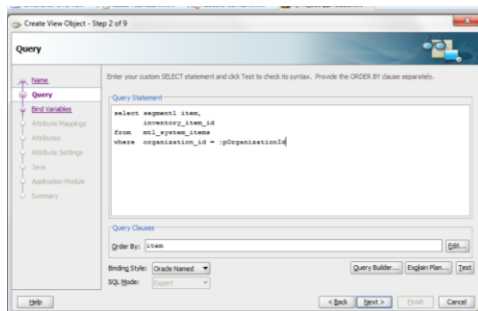
Here is an ADF example of creating a list of values related to the inventory\_item\_id on the sales order line table. The database id should be translated to a SKU# or product code. ADF provides a wizard to create a read-only view object. The wizard steps are displayed below:

1. Create a read-only view

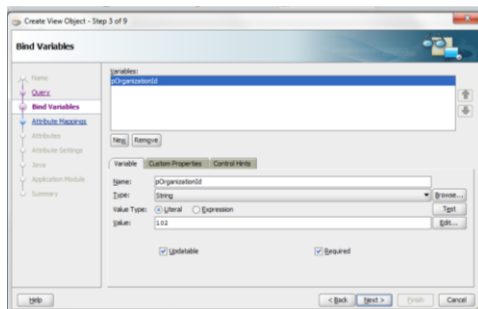
# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



2. Add your SQL statement with a parameter

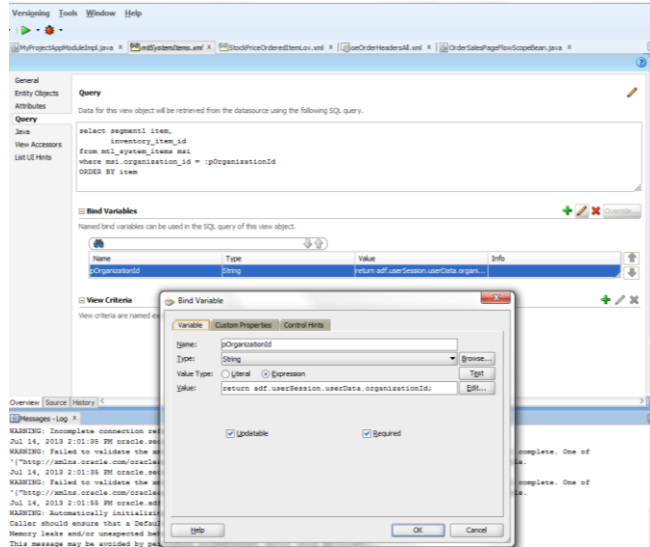


3. Define your parameter – Note: In this case, we hard coded the organization\_id but in a real application I would derive the value and set a session value.



Here is an example of passing a session value.

# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

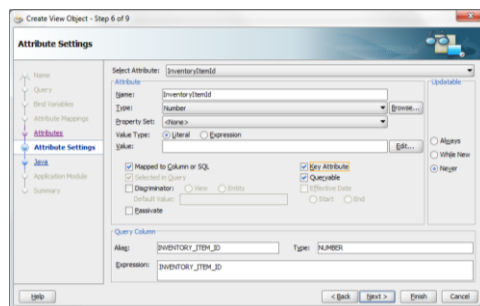


Value	return adf.userSession.userData.organizationId;
-------	---

This organization\_id session value is set in the AM Module. The organizationId is determined by another processes and in Oracle EBS it would be determine by calling a database function “fnd\_global.apps\_initialize”

```
private String getOrganizationId() {  
    return (String)getSession().getUserData().get("organizationId");  
}  
  
private void setOrganizationId(String organizationId) {  
    getSession().getUserData().put("organizationId",organizationId);  
}
```

4. Identify what column in your view is the primary key.

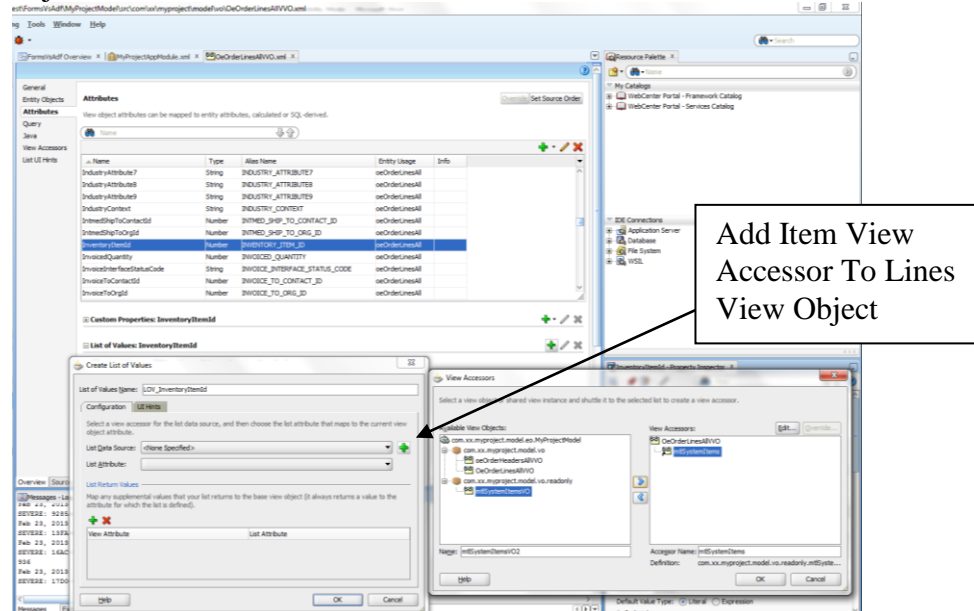


Click Finish

5. This next step needs to associate the list-of-values view to the attribute or database column defined on the sales order line view object or database table. In terms of ADF, this is called creating a view accessor to your sales order line or view object. The definition of a view accessor is the mechanism that lets you

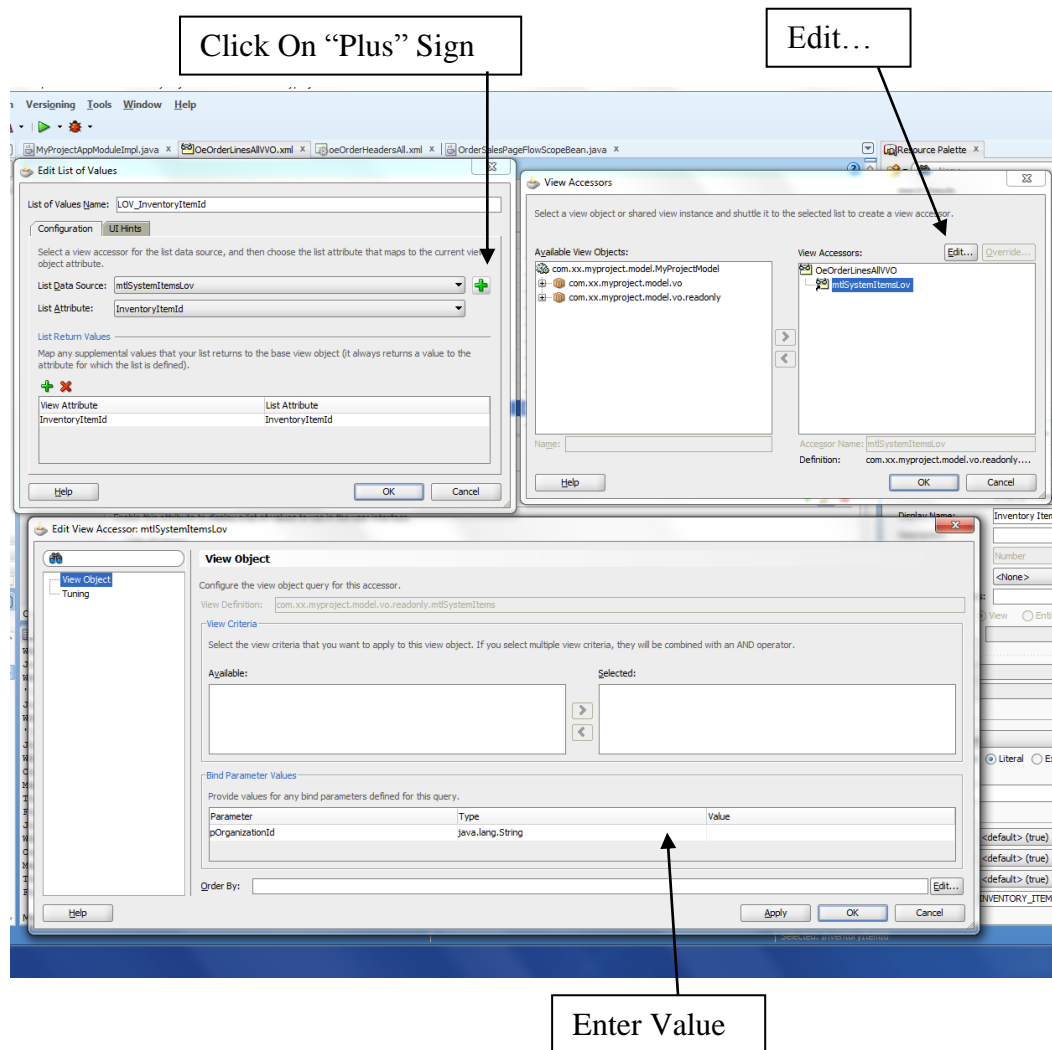
## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

obtain the full list of possible values from the row-set of the data source view object.

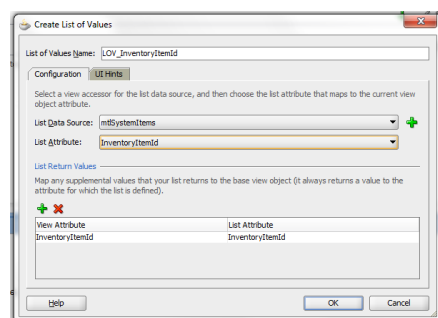


Note: You could also set the Vo parameter via the edit.... Option

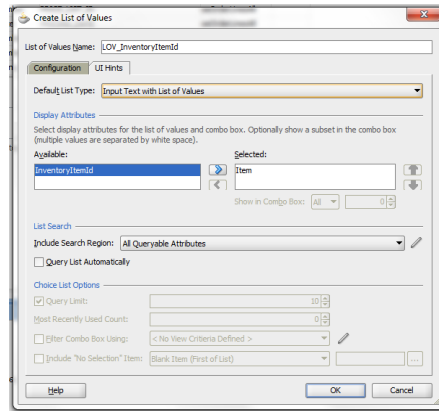
## The Basics of ADF and JDeveloper From an Oracle Forms Perspective



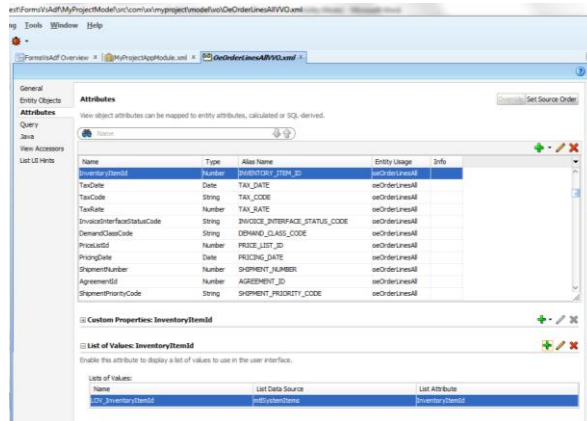
Once the view accessor is added to the line view object, it will be available in the "List Data Source" drop down list. Select the list data source and complete the mapping.



# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



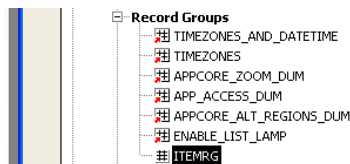
After the list of value is attached, you will see the list-of-values in the attribute window.



This is very similar to creating a list of values in Oracle Forms. First, a record group (view object) needs to be created. Then we need to create a list of values object, which is an explicit mapping of the record group result set to a data block definition. Lastly, we need to link the list of values definition to a specific field in the data block.

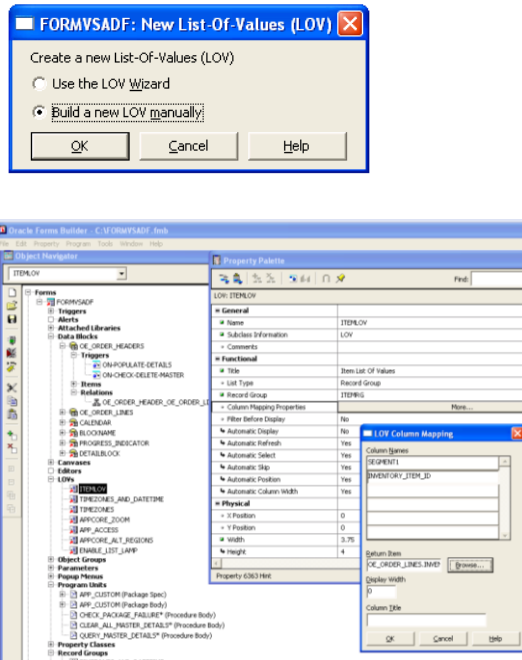
For my Oracle Forms example, I did not use the list-of-values wizard because I want to show the same type of steps as in ADF.

1. Create a record group which is equivalent to a read-only view object in ADF.

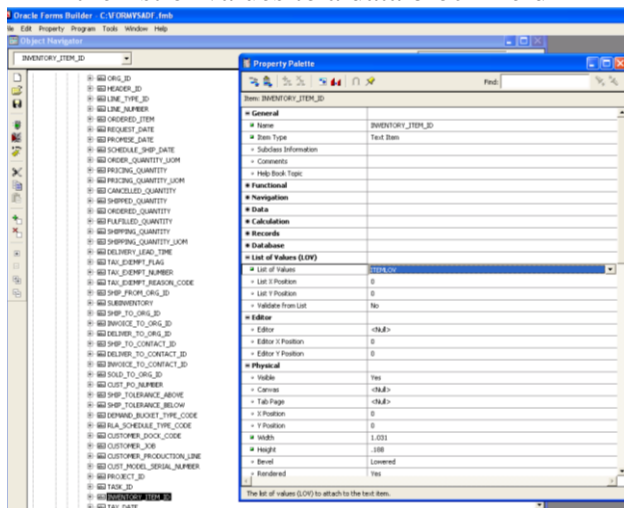


# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

2. Create a list of values (LOV) mapping between the definition of the result set and the Data Block field.

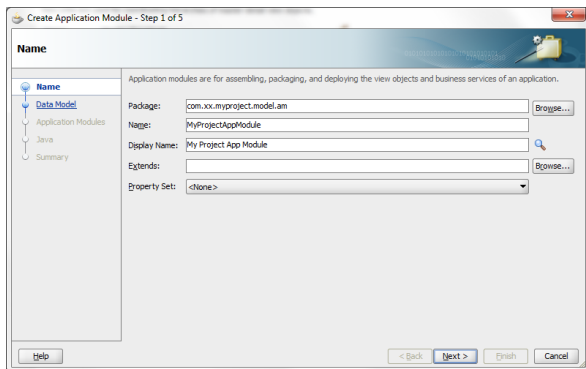
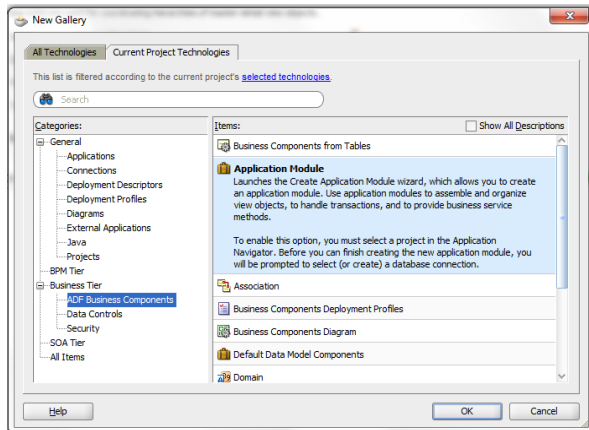


- ### 3. Link the list of values to a data block field

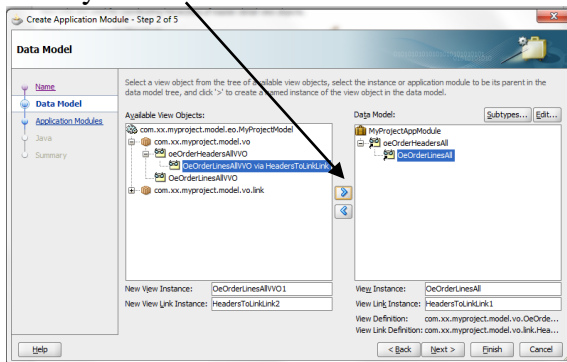


Once we have all our business services (entity objects, view objects, links and list of values objects) completed, our next step is to group the view objects together into application module so they can be exposed to the View layer. Here are the steps to create an application module in ADF:

# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



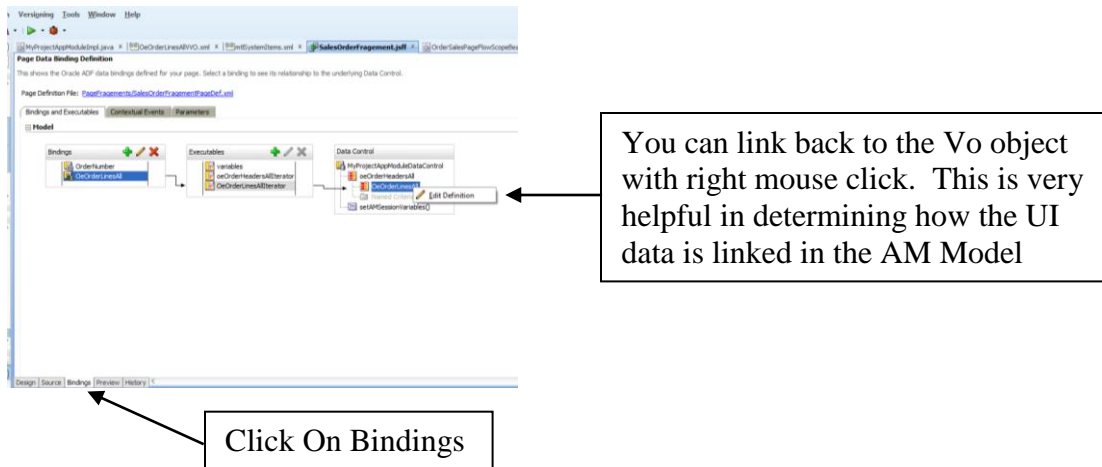
Once you shuffle “click Arrow” the Header and Line, click finish.



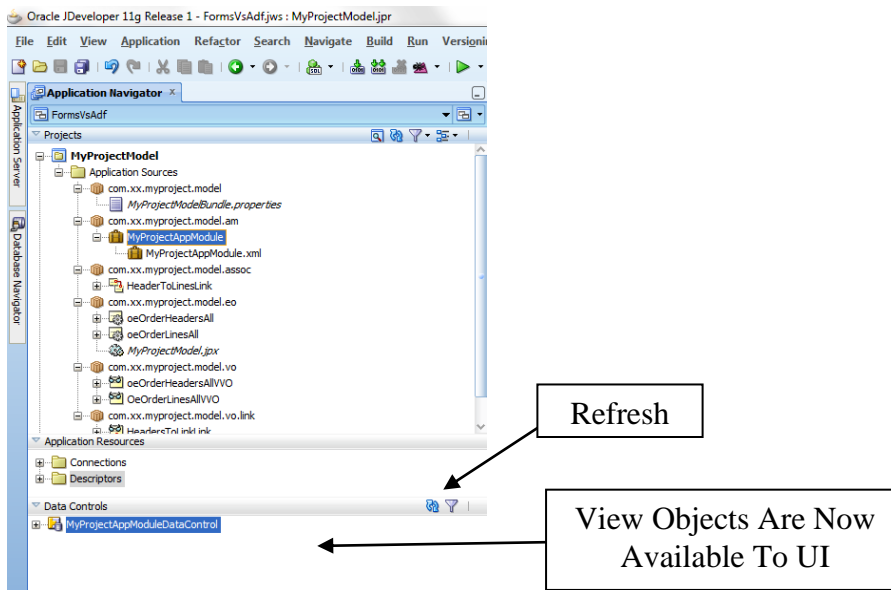
Note: Once you bind the objects in your UI Layer, the application module representation is visual in the View Layer Bindings (see below).



# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



When you are done defining the application module, refresh the “Data Controls” window and your project should look as follows:



For Oracle Forms, there is no action required to expose the Data Blocks to the user interface because the application module is implicitly exposed to the user interface or canvas.

## Initialize Application Session Values

For either Oracle Forms or ADF applications, we need to initialize the session when the application is launched. The purpose of the initializing is to prepare the application for the User who is about to enter into the application. In Oracle Forms, this is performed by the “When-New-Form” trigger which sets session specify values like inventory warehouse, operating unit, the correct operating unit, initialize descriptive flexfields or set the initial visual canvas. In ADF, the same actions can be performed by calling a

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

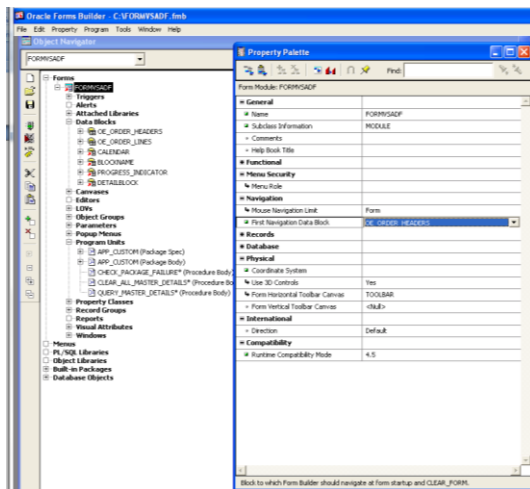
managed bean method or an application module method. As a refresher, a method is a collection of Java code with a sole purpose to carry out a task.

In a real-life ADF application, we would set session specific variables within the initialization method. For example, based on a given users responsibility, we could set visible and edit privileges within the application, specify an operating unit, determine what UI page should be rendered and place scope specific variables into Java Map so they can be referenced in other parts of the application.

Initializing your application is tightly coupled with controlling the flow of your application, so I will explain how you initialize your application in the following section (Design Application Flow).

### Design Application Flow

This next section compares how you control the flow of your application in Oracle Forms versus ADF. In Oracle Forms, we initially control the flow of the application by setting the “First Navigation Data Block” in project properties pallet.



The initialization of the Oracle Forms session and the option to set the initial canvas is set by the “Pre-Form” or the “When-New-Form” trigger. Here is an example of some PL/SQL code that is called by the form level trigger WHEN-NEW-FORM-INSTANCE:

```
PROCEDURE when_new_form_instance
IS
    win_msg          VARCHAR2(2000);
    l_org_id          NUMBER := TO_NUMBER(FND_PROFILE.VALUE('ORG_ID'));
    l_user_id         NUMBER := TO_NUMBER(FND_PROFILE.VALUE('USER_ID'));
    l_percision       NUMBER := NULL;
    l_name            hr_operating_units.name%TYPE;

BEGIN
    /*
    || Set First Navigation Block
    */
    FIRST_NAVIGATION_BLOCK ('OE_ORDER_HEADERS');
```

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

```
/*
|| Assign Valued To Control Block
*/
:CONTROL.USER_ID := l_user_id;

/*
|| Set Operating Unit Name
*/
SELECT hou.name
INTO   l_name
FROM   hr_operating_units hou
WHERE  hou.organization_id = l_org_id;

/*
|| Set Window Titles
*/
win_msg := 'Order Maintenance' || ' (' || l_name || ')';
set_window_property('MAIN_WIN', TITLE, win_msg);

/*
|| Get and Set Percision
*/
SELECT fc.precision
INTO   l_percision
FROM   hr_operating_units   hou,
       gl_sets_of_books     gsob,
       fnd_currencies        fc
WHERE  hou.organization_id = l_org_id
AND    hou.set_of_books_id = gsob.set_of_books_id
AND    gsob.currency_code = fc.currency_code;

:PARAMETER.SOB_PERCISION := NVL(l_percision, 2);

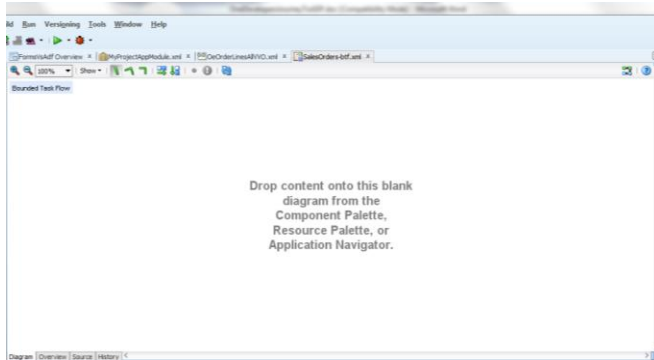
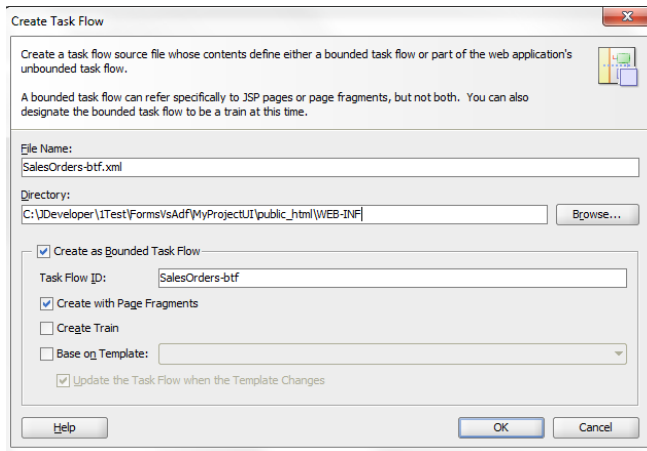
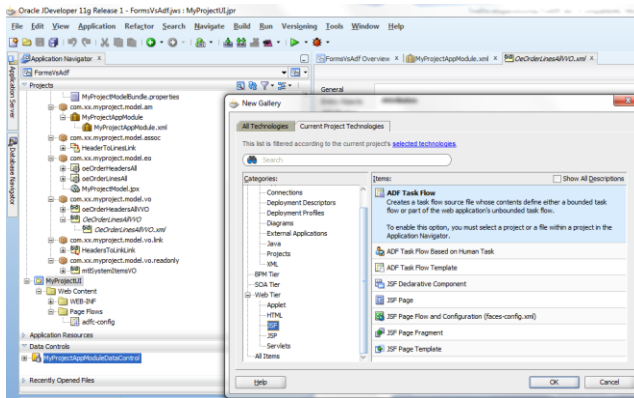
/*
|| Set DFFs
*/
FND_DESCR_FLEX.DEFINE
(
    block           => 'OE_ORDERS_HEADERS',
    field           => 'DF',
    appl_short_name => 'ONT',
    desc_flex_name  => '<some value>',
    title           => '<some value>'
);

END when_new_form_instance;
```

For Oracle Forms, the Developer can use hot-keys, GO-BLOCK and CALL-FORM options to control the flow of your application. The navigation coding is buried within forms triggers and PL/SQL code and sometimes can be difficult to debug as compared to Oracle ADF, which has a tool to visually control your flow of the application.

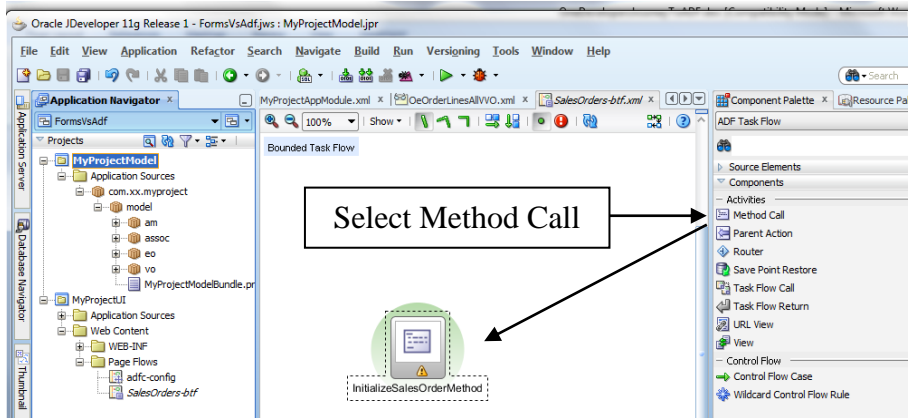
In ADF, you control the flow of your application by using a task flow. A task flow is a visual way to control the flow of your application. When you initially create your bounded task flow object, the task flow will be completely empty and it is your responsibility to visually define your flow. Once you launch the task flow wizard, you will see the following:

# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

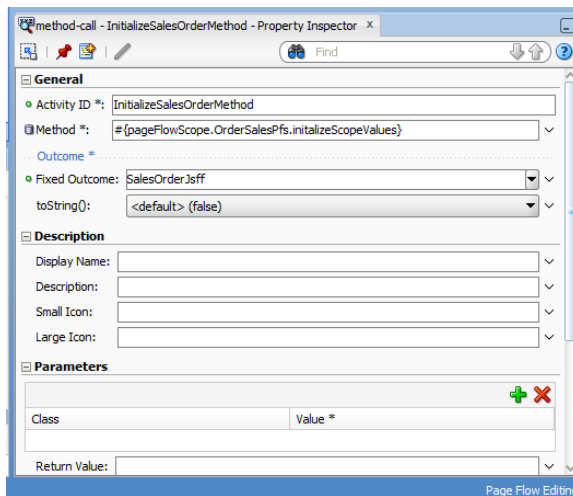


In ADF, we initialize our application by adding a “Method Call” activity in the task flow and we set this activity as the default. The Method Call activity will reference a managed bean method or application module method. In this example, I am referencing a managed bean method.

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective



In this example, I named the “Method Call” activity InitializeSalesOrderMethod and it references a managed bean called initializeScopeValues.



Here is a sample code that would be contained in the managed bean method. In this case, we are referencing an application parameter, which is defined in your task flow. In addition, it is calling an Application Module (AM) method that will initialize the EBS database session variables.

```
public void initializeScopeValues() {
    System.out.println("Start - initializeScopeValues");

    Map<String, Object> params = new HashMap<String, Object>();
    Map pfMap = AdfFacesContext.getCurrentInstance().getPageFlowScope();

    // These are task flow parameters which are similar to Oracle Form Parameters
    params.put("Warehouse", pfMap.get("pWarehouse"));

    System.out.println("Parameter - Warehouse: " + params.get("Warehouse"));

    // Get Session Values Like User Name
    SecurityContext context = ADFContext.getCurrent().getSecurityContext();
    String username = context.getUserName().toLowerCase();

    System.out.println("Variable - username: " + username);

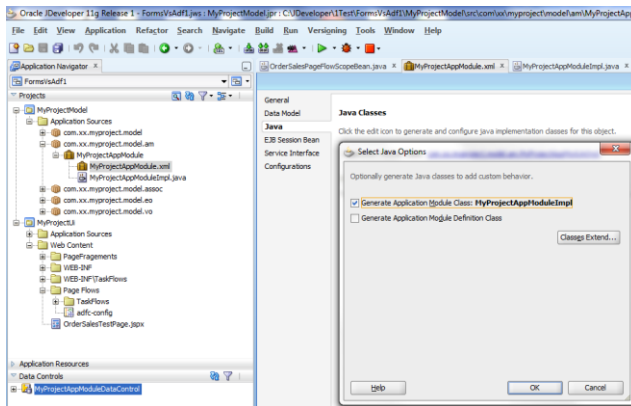
    //Call AM Method To Initialize Session Variables In The Model
}
```

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

```
BindingContainer bc = BindingContext.getCurrent().getCurrentBindingsEntry();
bc.getOperationBinding("setAMSessionVariables").execute();

System.out.println("End - initializeScopeValues");
}
```

In order to reference an AM method, you first need to create an Application Module Java class, add that method to the Application Module so it can be exposed via a Data Controller, which in turn allows you to consume and bind that AM method to Method Call Activity or UI Page. Here are the steps to create an AM Java Class that will hold your setAMSessionVariables method:



Open the newly created Java Class and write your new method.

```
public void setAMSessionVariables(){
    System.out.println("Start - setAMSessionVariables");

    String SessionWarehouseId = null;
    CallableStatement st =null;

    // Get the Page Flow Scope Variable From the Current Session And Assign It To A Session Variable
    SessionWarehouseId = ADFContext.getCurrent().getPageFlowScope().get("pfsWarehouseId").toString();

    // Set Warehouse Session Variable Which Can Be Use By Application Module / View Objects
    this.getSession().getUserData().put("pWarehouseId", SessionWarehouseId);

    System.out.println("End - setAMSessionVariables");
}
```

Note: When using session variables, you should test your application for activation and passivation because the userData HashMap is not persisted by default when activation / passivation occurs for application modules that have Application Module pooling enabled, which means that custom session data may be lost between requests. What we need to do is to passivate the session user data together with the other state data stored by the framework and load it back when the AM is requested the next time (when it gets activated again). To do this we have to overwrite two methods in the ApplicationModuleImpl class.

```
@Override
protected void activateState(Element aElement) {
    super.activateState(aElement);
    mLogger.info("+++++++ activateState");
    Hashtable lData = getSession().getUserData();
    if (aElement != null) {
        // 1. Search the element for any <PrivData> elements
        NodeList nl =
            aElement.getElementsByTagName(PRIVATEDATA);
    }
}
```

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

```
        if (nl != null) {
            // 2. If any found, loop over the nodes found
            for (int i = 0, length = nl.getLength(); i <
length; i++) {
                // 3. Get first child node of the <PrivData>
                element
                    Node child = nl.item(i).getFirstChild();
                    if (child != null) {
                        // 4. Set the data value to the user data
                        hashmap
                            String lDataString =
child.getNodeValue();
                            String[] lSplitkeyval =
lDataString.split(";");
                            for (int ii = 0; ii <
lSplitkeyval.length; ii++) {
                                mLogger.fine("..." + lSplitkeyval[ii]);
                                String[] lSplit =
lSplitkeyval[ii].split("=");
                                lData.put(lSplit[0], lSplit[1]);
                            }
                            break;
                        }
                    }
            }
        }

@Override
protected void passivateState(Document aDocument, Element
aElement) {
    super.passivateState(aDocument, aElement);
    mLogger.info("----- passivateState");

    // 1. Retrieve the value of the user data to save and
build a string representation
    Session lSession = getSession();
    Hashtable lData = lSession.getUserData();
    String lDataString = "";
    Set<String> keyset = lData.keySet();
    if (!keyset.isEmpty()) {
        Iterator<String> keys = keyset.iterator();
        while (keys.hasNext()) {
            String key = keys.next();
            mLogger.fine("..." + key + "=" + lData.get(key));
            lDataString += key + "=" + lData.get(key) + ";";
        }
    }

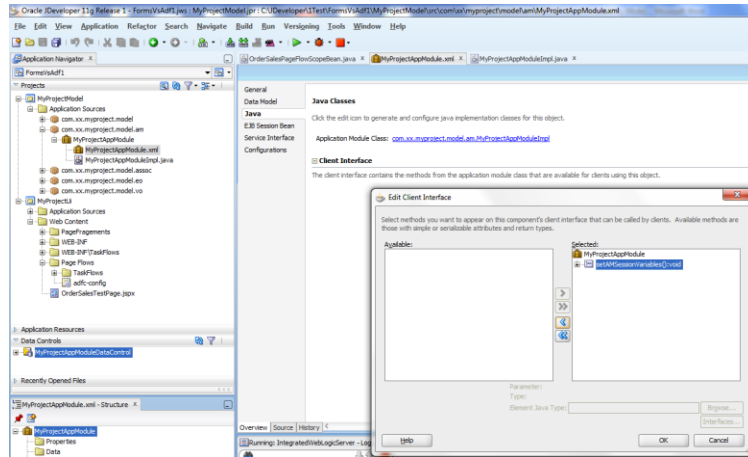
    // 2. Create an XML element to contain the value
    Node node = aDocument.createElement(PRIVATEDATA);
    // 3. Create an XML text node to represent the value
    Node cNode = aDocument.createTextNode(lDataString);
    // 4. Append the text node as a child of the element
    node.appendChild(cNode);
    // 5. Append the element to the parent element passed in
    aElement.appendChild(node);
}
```

OTN Harvest - April 2012

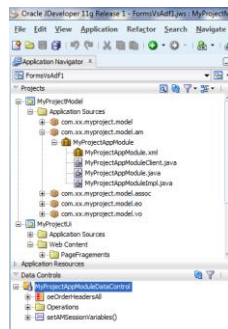
<http://www.oracle.com/technetwork/developer-tools/adf/learnmore/april2012-otn-harvest-1609383.pdf>

# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

Add the new method to the Client Interface.



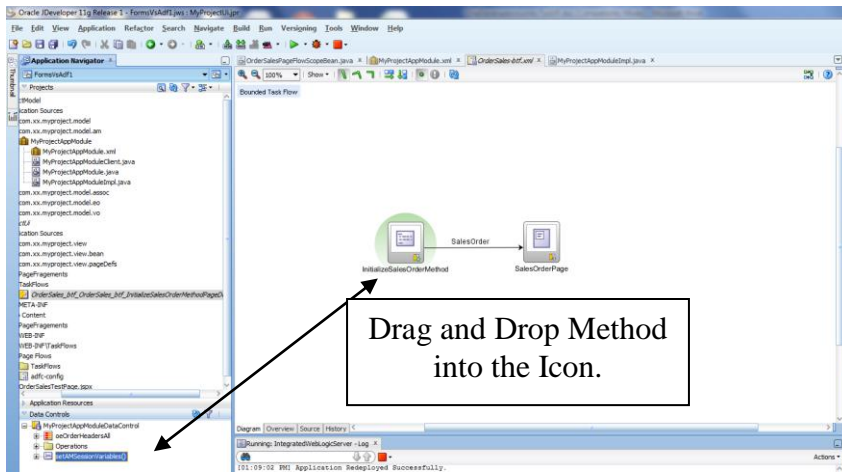
Refresh your Data Control Panel and you will see that the method is now exposed and can be consumed by the Method Call Activity in the View Layer.



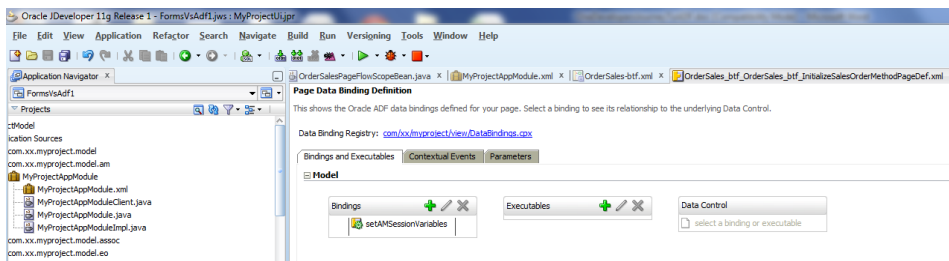
The last step is to bind the exposed AM method or “setAMSessionVariables” to your Method Call Activity or InitializeSalesOrderMethod.



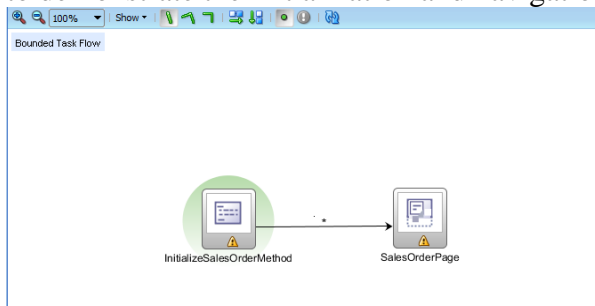
# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



Note: Once you drag and drop the exposed method onto the Method Call Activity, the ADF framework will automatically create the binding for you.

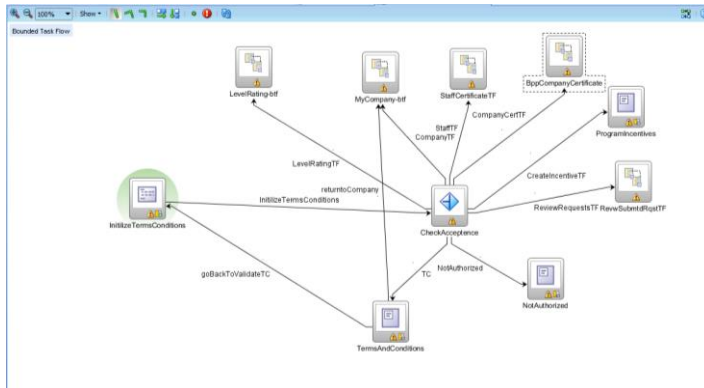


Once you initialize your ADF application, the next step is to navigate to your first page. In this example, I only have one method and one UI page so this is a very simplistic way to demonstrate the initialization and navigation of an ADF application.



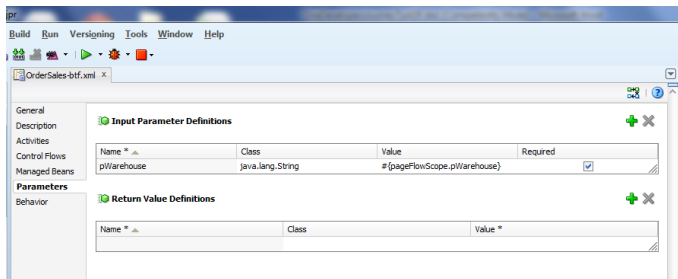
In a real life ADF application, the task flow design can quite complex.

# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

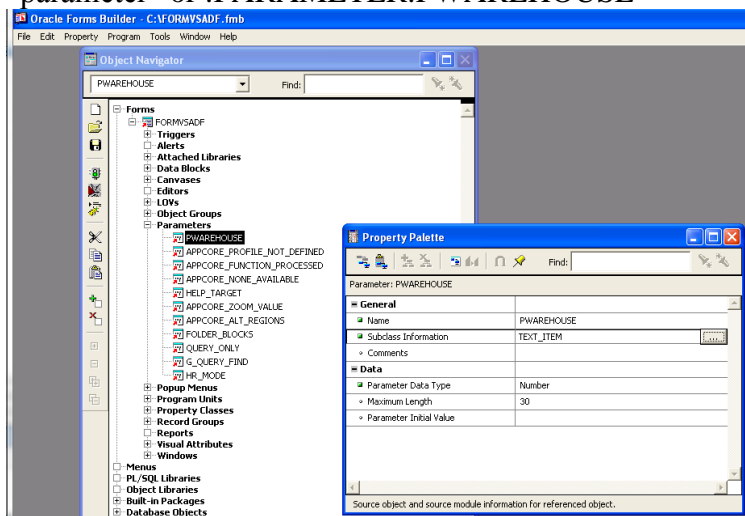


## Application Parameters

In the previous section or Design Application Flow, I mentioned that ADF applications can accept parameters. Parameters are defined as part of a task flow. In this example, I created one required parameter that accepts the warehouse Id. The parameter value can be retrieved by getting the value from the pageflowscope hash map object as demonstrated in the previous session.



In Oracle Form, the application parameter is defined under the application and the parameter value can be referenced by prefixing the variable with the key word “parameter” or :PARAMETER.PWAREHOUSE



## Designing Your UI Application

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

Translating most of the commonly used Oracle Forms UI components, such as modal and non modal windows, canvas, stacked canvases, frames, input text items, labels, buttons, tabs, current record indicator and multi-row block “Table” is straight forward. In Oracle Forms, we have a very limited set of components as compared to ADF, so the challenge is trying to understand all the new components and features available with ADF. In order to help the Developer understand each component, Oracle provides a tool called “ADF Faces Rich Client Demos”, or <http://jdevadf.oracle.com/adf-richclient-demo/faces/index.jspx>.

In addition to hundreds of new UI components, ADF introduces the concept of reusable pages or page fragments. Page fragments allow you to create parts of a page and a page can be made up of one or more page fragments. When building your application, determine if you have any common tasks that could take advantage of page fragments. Another reason to use page fragments is to break down very large and complex pages so it is easier to maintain.

In order to transition your current understanding of Oracle Form components, build a simple master-detail project and practice using these ADF components and discovering any new features by reviewing “ADF Faces Rich Client Demos”.

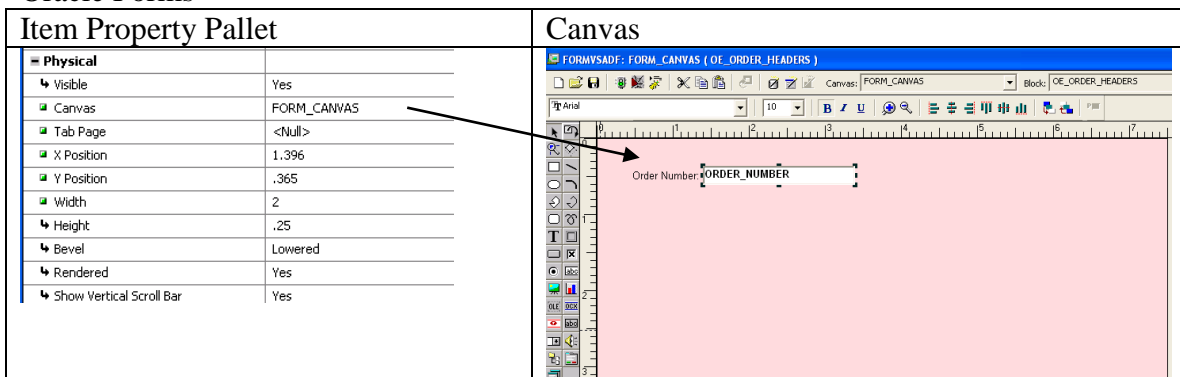
For additional reference, review an Oracle video that compares the differences between Oracle Forms and ADF. This video is called “Redeveloping Oracle Forms in ADF” ADF Insider Essentials.

Redeveloping Oracle Forms in ADF	<a href="http://www.youtube.com/watch?v=TiqbW1CAMMc">http://www.youtube.com/watch?v=TiqbW1CAMMc</a>
----------------------------------	---

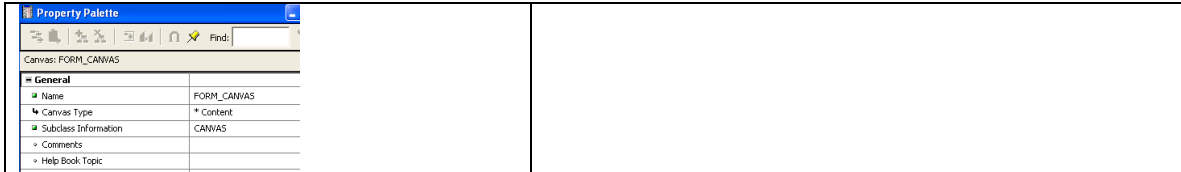
### Common Layout Components

ADF and Oracle Forms are similar in that they provide a declarative way of adding components to pages. With Oracle Forms, the Developer will declaratively add field components as well as any layout features to the page using the property pallet and or canvas layout components. The Developer is also responsible for the exact position of each item on the page.

#### Oracle Forms

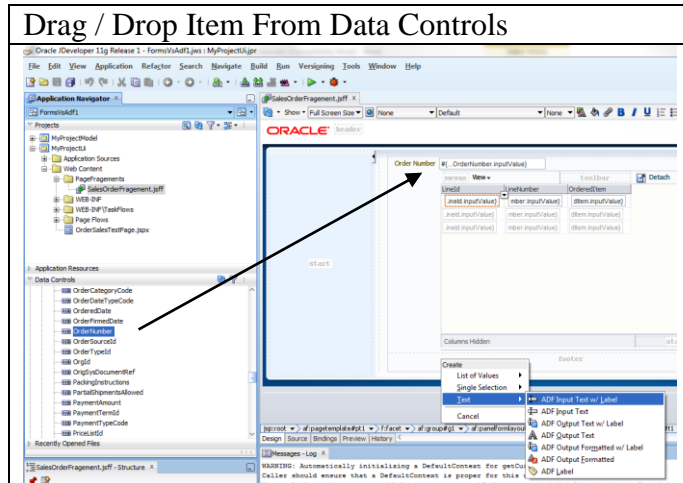


# The Basics of ADF and JDeveloper From an Oracle Forms Perspective



One of major differences between Oracle Forms and ADF is the Developers ability to control the exact location of the components on a page. Oracle Forms requires the Developer to specify the exact location of the component on the page; whereas, ADF can only control the relative position of a component on the page. The exact position of the components will be determined when the page is rendered to the User.

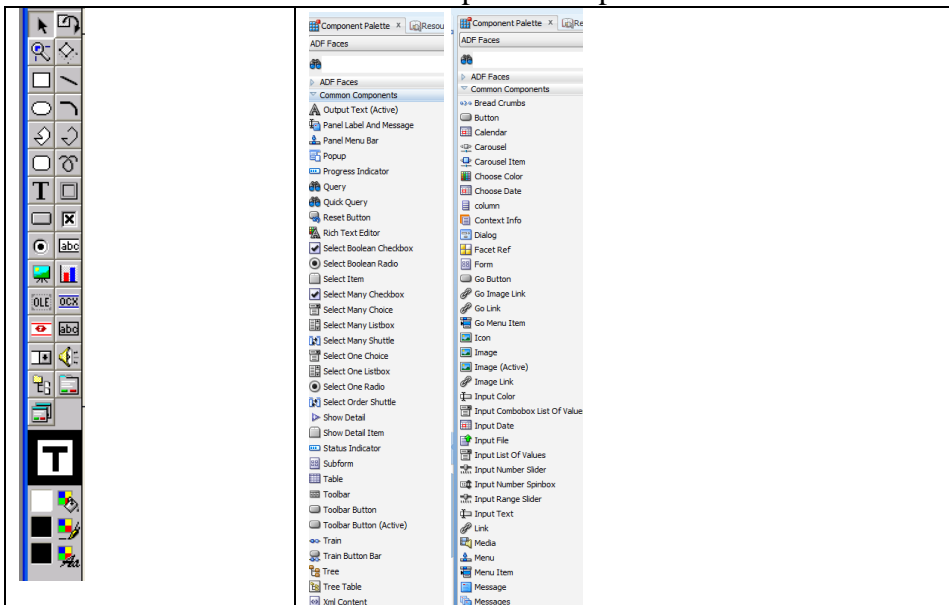
## ADF



### Comparison of the property palette components between Oracle Forms and ADF:

## Oralce Forms

### ADF – Sample of components



## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

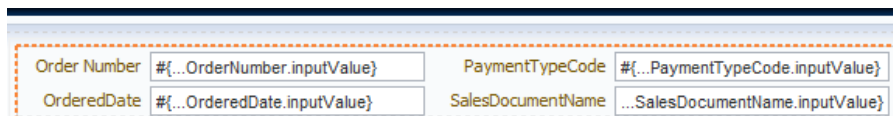
Another major difference between Oracle Forms and ADF is that ADF has layout features that surround the components on the page. This is very similar to Oracle Reports whereby we have frames and anchors to control the flow of the page. The ADF layout features control the flow of the page as well as how the components are displayed inside the page. As part of these new layout features, there are two new concepts to understand.

One new concept is called a facet. A facet is a layout component that contains other layout components and it is responsible for maintaining the relative positioning of all of its contents. For example, a `panelBorderLayout` contains a top, bottom, start, end and center facet and any items placed into each area or facet will retain their relative position on the page; therefore, a company logo will be placed in the top facet and any copyrights info will be placed in the bottom facet. This will ensure the “picture frame” will remain consistent regardless of how the center of the page is displayed.

The second new concept is called geometry management and component stretching. Geometry management is the process by which the parent components, and child components negotiate the actual sizes and locations of the components in an application or page. At the core of geometry management, solution is a resize notification mechanism that allows components that support geometry management to be notified of browser resize activity. In other words, geometry management will make sure the application is displayed correctly based on the size of the screen.

In this next section, I will explain some of the common layout components and their purpose.

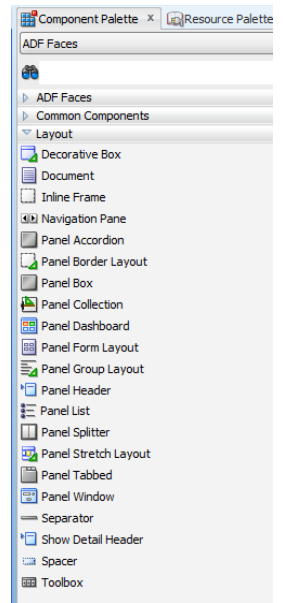
- `af:panelForm` – Mostly used for input fields on a page. The layout will automatically right justify the prompts and left justify the fields.



The screenshot shows a UI design within a JDeveloper window. It features a `panelForm` layout containing four input fields arranged in a 2x2 grid. The fields are labeled 'Order Number', 'PaymentTypeCode', 'OrderedDate', and 'SalesDocumentName'. Each label is right-aligned, and each input field is left-aligned. The labels and fields are enclosed in a dashed orange border.

Note: Some UI design call for left justification of labels and this can be accomplished by modifying the Cascading Style Sheet (CSS), which is called a Skinning in ADF.

- `af:panelHeaderLayout` – This component provides a title for the group of items contained within the layout. In Oracle Forms, this would be comparable to a Frame.
- `af:panelGroupLayout` – The layout allows you to layout components in a row or column. For simple ADF pages without `panelsplitter` or `panelStretchLayouts`, the



## The Basics of ADF and JDeveloper From an Oracle Forms Perspective

outermost layout will be a panelGroupLayout with a setting of vertical which will cause all remaining components/layouts to flow vertically on the page.

- Vertical

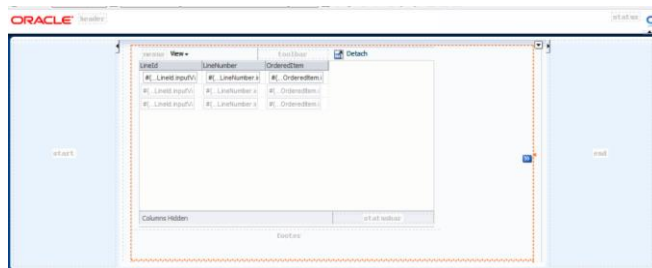


- Horizontal



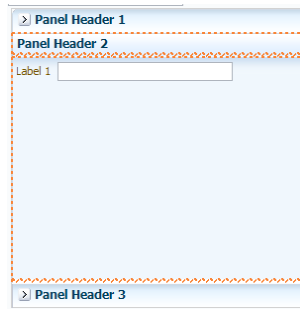
Note: You can nest another layout within the outermost layout and that nested layout's components will retain its individual layout properties such as horizontal positioning but the layout itself or container will be arranged vertically within the page.

- af:panelBorderLayout – The components dropped into each facet will maintain their relative position at all times. This layout is used when you need a header section for a logo and information in footer section of the page.



- af:panelStretchLayout – Expands contained components to fill the width of the page. This is mostly used if another layout feature cuts off the display of a layout component.
- af:panelSplitter – This component will divide an area into two horizontal or vertical panels. The Developer also has the option to allow the User to adjust the size of each panel. I use this feature with a nested accordion layout to provide a quick reference menu or links on the left hand side of the page.
- af:panelAccordion – Looks and acts like several email software layouts.

## The Basics of ADF and JDeveloper From an Oracle Forms Perspective



For Oracle Forms Developers, the ability of not controlling the exact position of a component will be frustrating until you learn how to control the components position by using the relative positioning layout options. In addition to Developers, the Application Architects who design these new pages should have an understanding of effort level required to build pages using relative positioning versus the ability to specify the exact location of a component on a page. For example, a page layout developed in ADF might take twice as long as Oracle Forms because the Developer has to find a way to nest layout components to achieve the desired page layout. I have provided a couple of references for designing the perfect layout.

Layout Best Practices	<a href="http://www.oracle.com/technetwork/developer-tools/adf/layoutbestpractices-084287.html">http://www.oracle.com/technetwork/developer-tools/adf/layoutbestpractices-084287.html</a>
Achieving The Perfect Layout with ADF Faces RC	<a href="http://www.quovera.com/whitepapers/downloads/koletzke_adf_faces_layout_paper.pdf">http://www.quovera.com/whitepapers/downloads/koletzke_adf_faces_layout_paper.pdf</a>

As part of mastering the layout components, its also important to understand that each layout component has stretching properties. The stretching properties are important because most ADF pages will have nested layout components so the nested layout componets must be compatiable to achive the correct layout. For example, a stretchable child component needs to be nested inside a stretching parent object otherwise the child component might not render properly on all browsers. Therefore, the correct combination of layout components is necessary to developing desired layout.

For additonal reference, I have provided a URL which contains the geometry properties for each component. For exmple, the panelGroupLayout geometry can be found using ADF Demo.

panelGroupLayout geometry properties	<a href="http://jdevadf.oracle.com/adf-richclient-demo/faces/components/panelGroupLayout.jspx">http://jdevadf.oracle.com/adf-richclient-demo/faces/components/panelGroupLayout.jspx</a>
Table 8-1 ADF Faces Layout Components	<a href="http://docs.oracle.com/cd/E16764_01/web.1111/b31973/af_orgpage.htm">http://docs.oracle.com/cd/E16764_01/web.1111/b31973/af_orgpage.htm</a>

### Deployment

Neither ADF nor Oracle Forms create a stand-alone executable; therefore, the compiled source code needs to run on some type of server that can interpret the compile code. In

# The Basics of ADF and JDeveloper From an Oracle Forms Perspective

the case of Oracle Forms, we typically use the Oracle's Enterprise Business Suite technology stack to run our compiled application. As for ADF applications, we need a Java EE-compliant server, such as a WebLogic Server (WLS) to run our compiled ADF application.

Deploying your application to a Java EE server requires two steps:

1. Prepare the deployment file. This step consists of creating a deployment file, which consists of collecting all the supporting libraries into a Java archive (JAR) file.
2. Copy the deployment file to the application server. In this step, you can use either JDeveloper to install your application on the WLS server or you can upload the JAR file to the WLS server and then use a WLS deployment tool in the WLS console to install your application.

Here are a few deployment references:

Deploying Applications To WebLogic Server Using JDeveloper and WLS Console	<a href="http://www.quovera.com/whitepapers/downloads/rmoug_2012_deployment_doc.pdf">http://www.quovera.com/whitepapers/downloads/rmoug_2012_deployment_doc.pdf</a>
FAQ for Integration of Oracle E-Business Suite and Oracle Application Development Framework (ADF) Applications	<a href="https://metalink.oracle.com/metalink/plsql/show_doc?db=NOT&amp;id=1296491.1">https://metalink.oracle.com/metalink/plsql/show_doc?db=NOT&amp;id=1296491.1</a>

## Conclusion

In conclusion, transitioning from Oracle Forms to ADF will be overwhelming, time consuming, always challenging, sometimes frustrating; however, the most important thing to remember is that it is possible. This paper was designed to provide a high level how-to guide for Developers transferring their skill sets from Oracle Forms to ADF pages. There are multiple references in the paper that should be reviewed to gain a better understanding of the basic concepts of ADF. Your true understanding of ADF development will be achieved when you start building your first couple of applications. Lastly, it is important to realize that it takes time to understand ADF, so be patient and take that first step in to the world of ADF.