A Developers Journey From Oracle EBS Forms To Oracle EBS ADF Pages

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 Collaborate 2013
Denver, Colorado

Abstract

This presentation will explain my journey from an Oracle EBS Forms Developer to an Oracle ADF Pages Developer. My journey starts as an experienced Oracle Forms Developer that is starting a new Oracle ADF application with no knowledge of Oracle ADF "skill set" and ends with supporting an ADF application. Prior to starting this new Oracle ADF application, I had attended some formal Java education training classes, but that was ten years ago so I only remember concepts.

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 finally deploying that application in a WebLogic /Oracle EBS environment.

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. How to deploy an application in a Weblogic / Oracle EBS environment?
6. 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 finally 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 frame work of Model, View, Controller. The ADF framework separate 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 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 understand 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 JDevloper application; 2) Deploy using and Enterprise Application Archive (EAR File). In addition, basic navigation skills are required for the WebLogic console & enterprise manager screens.

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

The most difficult and frustrating part of my journey was 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	http://docs.oracle.com/javase/tutorial/
IBM Java Tutorial	http://www.ibm.com/developerworks/java/tutorials/j-introtojava1/j-introtojava1-pdf.pdf

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:

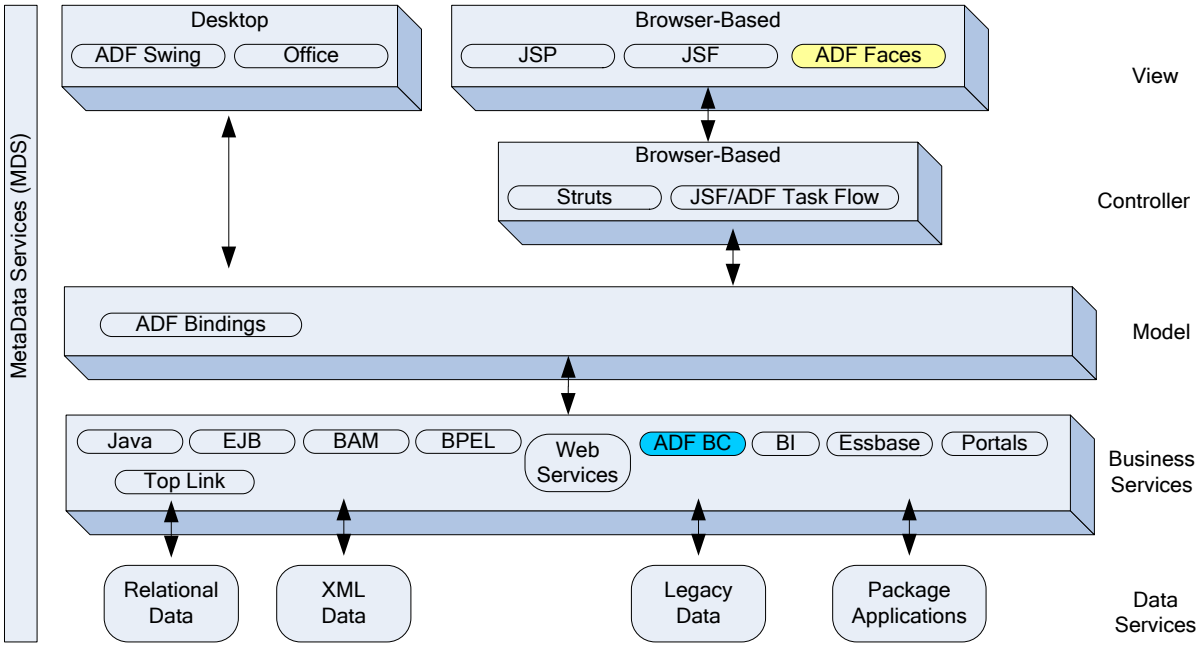
Java APIs	http://docs.oracle.com/javase/6/docs/api
Oracle APIs	`oracle.jbo.*` package

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 validation routines, setting a database sequence on a table, used to build error messages, referencing built-in calls like setting current date & 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	http://www.oracle.com/technetwork/developer-tools/jdev/introduction-to-groovy-128837.pdf

ADF Concepts

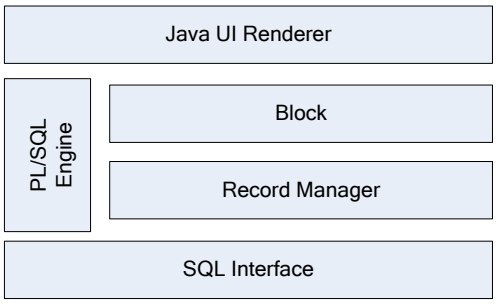
What is the ADF Framework?
Oracle Application Development Framework (ADF) is a Java-based development tool (much like Oracle Forms is a PL/SQL-based tool) designed to take full advantage of Java Enterprise Edition or Java EE. ADF Technology simplifies interaction with "Java" EE and Oracle's Fusion Middleware.



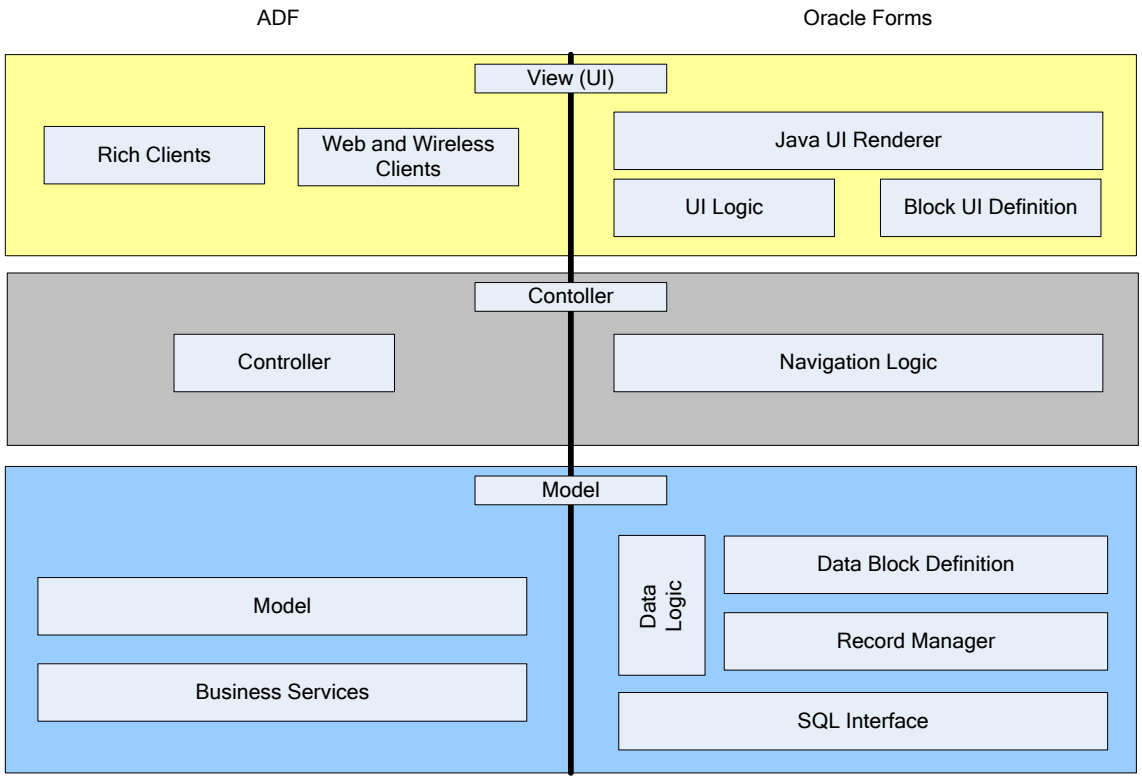
- The view layer contains the UI pages used to view or modify that data
- The controller layer processes user input and determines page navigation
- The model layer represents the data values related to the current page
- The business service layer handles data access and encapsulates business logic

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?



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 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 & 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 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.

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 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	http://jdevadf.oracle.com/adf-richclient-demo/faces/index.jspx

Controller

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)	http://www.oracle.com/technetwork/developer-tools/jdev/adf-task-flow-design-132904.pdf

Oracle® Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework 11g Release 1 (11.1.1.6.0)
- Section 14 Getting Started With Task Flows http://bit.ly/adfdevguide111160s14
- Section 16.4 Sharing Data Control Instances http://bit.ly/adfdevguide111160s164
- Section 18 Introduction to Complex Task Flows http://bit.ly/adfdevguide111160s18
- Section 18.3 Managing Transactions http://bit.ly/adfdevguide111160s183

Additional Concepts

Application Module (AM)

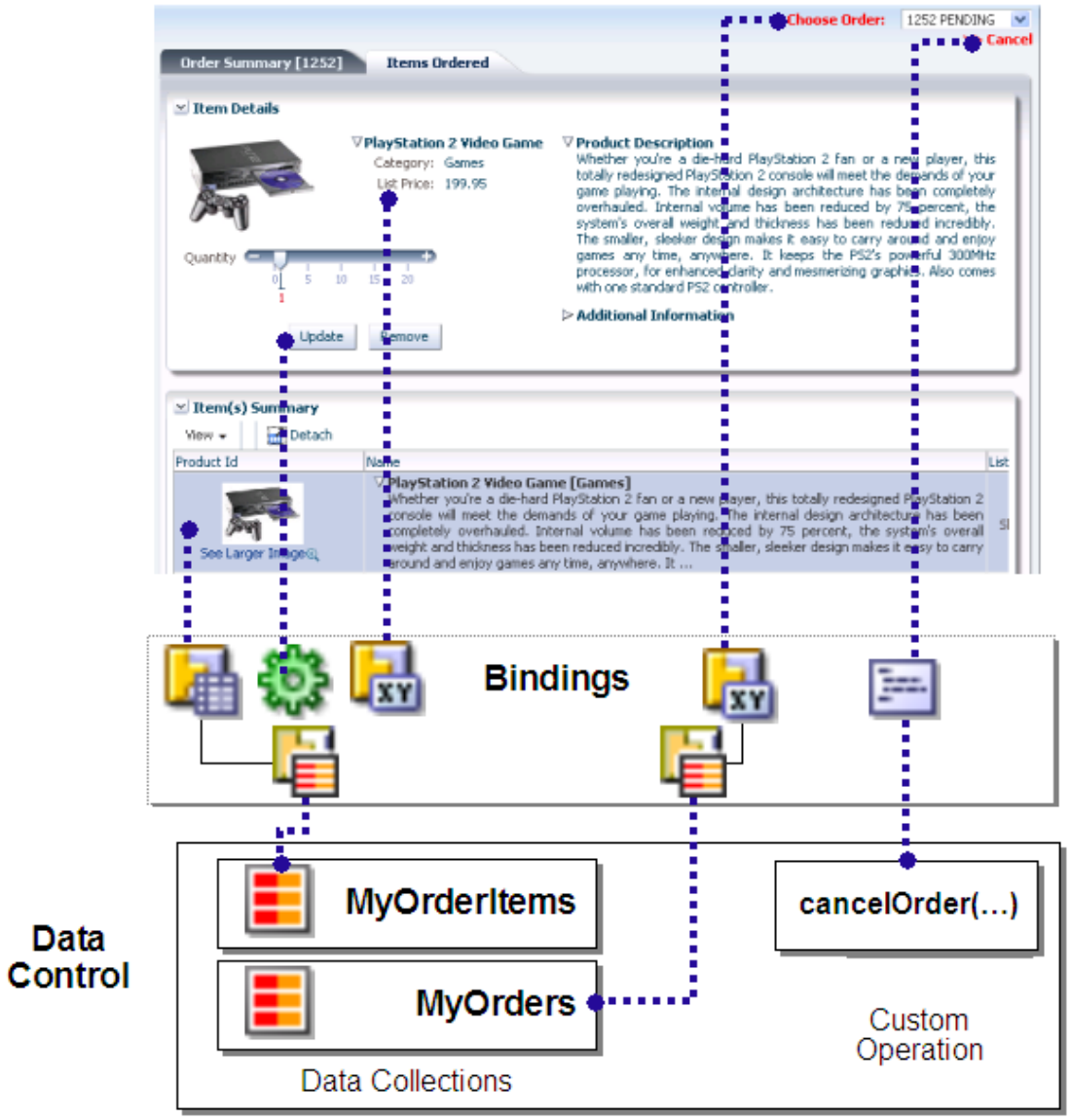
This is a collection of business rules and transactions (view objects and view links defined in the Model layer) that are related to a certain function or use case. When building your application, the view and view links need to be added to a container or application module (AM), so they can be exposed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

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. 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:

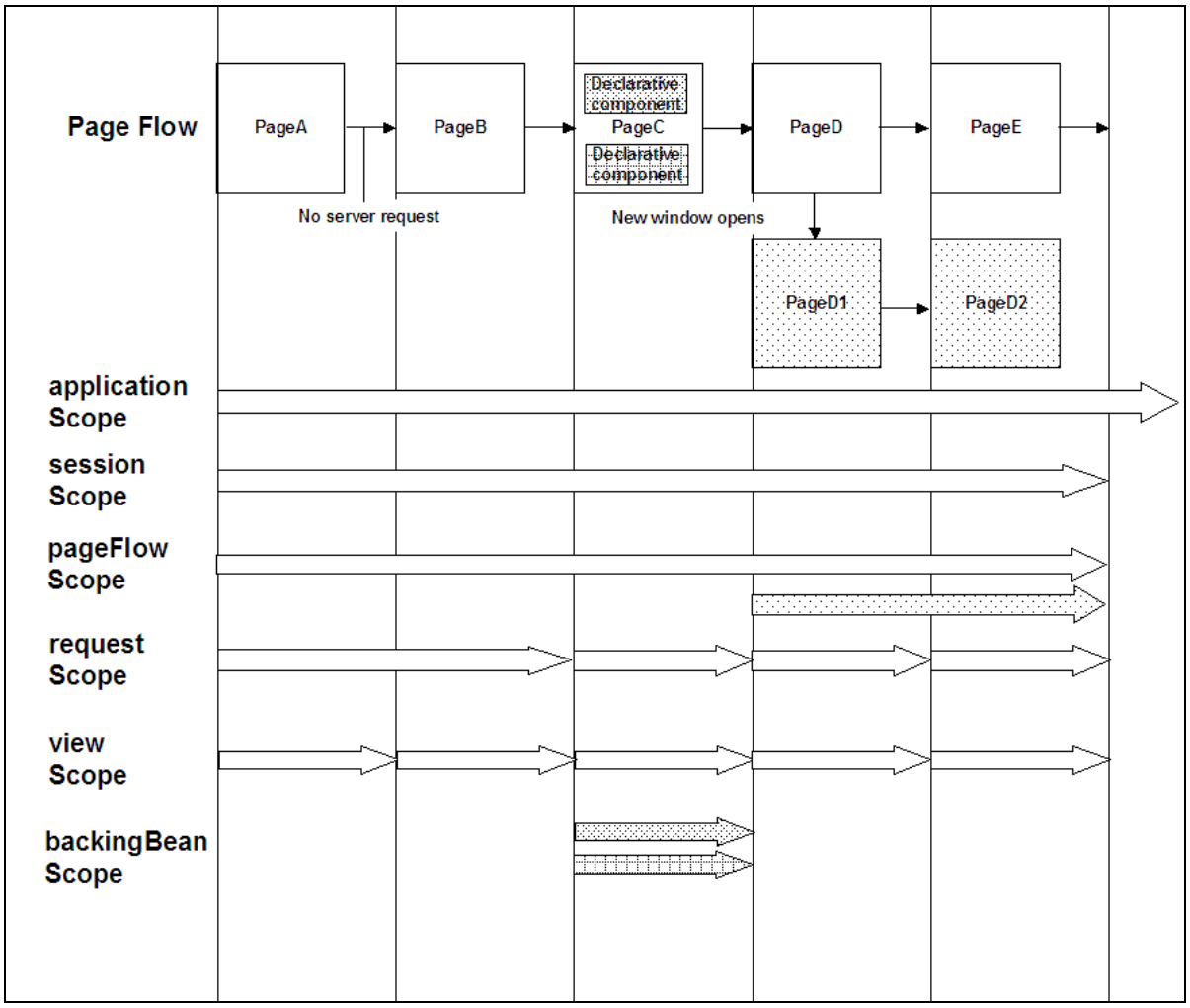
applicationScope	The object is available for the duration of the application.
sessionScope	The object is available for the duration of the session.
requestScope	The object is available for the duration between the time an HTTP request is sent until a response is sent back to the client.

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.
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.

Expression Language

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	http://docs.oracle.com/javaee/1.4/tutorial/doc/JSPIntro7.html

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.

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 PreferredPaymentMethodChangeListner(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 behavior for your business domain objects. • Get an attribute • Set an attribute

	Note: Add custom code to set a value after it is retrieved from the database and before it's passed to the view object or to override the view object value before storing the value into the database.
ViewObjectImpl Class	A view object is a base class for view objects • Set Where By Clause • Set Order By Clause For Form Developers, this is where your block triggers exists. Note: I have also seen Developers set the Where By and Order By in an Application Module method
ViewRowImpl Class	A view object is a base class for view row objects • Get Attribute • Set Attribute 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.
Managed Bean	This is where you store all your UI based code that controls the page as well as call methods in the application module. • Initialize code for your session • Call methods defined in the application module • Conditionally Rendering • Set variables For Form Developers, this is where all your UI events triggers

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)

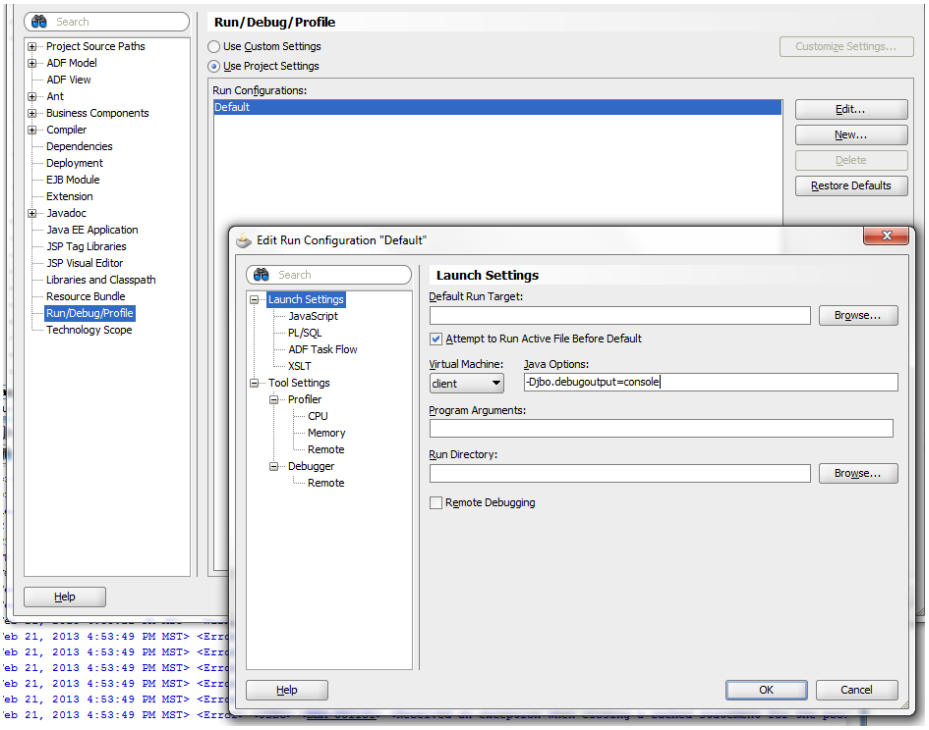
ADF and JDeveloper provide several ways to debug your code.

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).



    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.**model**
Entities	com.xx.myproject.**model entities**
Associations	com.xx.myproject.**model.assoc**
View Links	com.xx.myproject.**model.vo.link**
Updatable Views	com.xx.myproject.**model.vo**
Read Only Views	com.xx.myproject.**model.vo.readonly**
Application Module (AM)	com.xx.myproject.**model.am**
Diagram	com.xx.myproject.**model.diagram**

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	http://www.oracle.com/technetwork/developer-tools/jdev/overview/index.html

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.

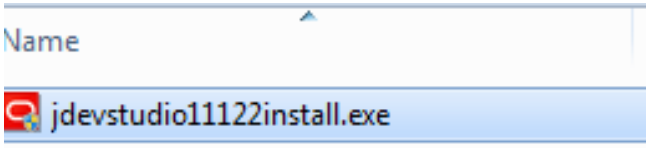
If you are NOT planning on deploying to a Weblogic Server (WLS), then download the latest JDeveloper version. In all other cases, reference the compatibility link below:

ADF Runtimes vs WLS versions as of JDeveloper 11.1.1.6.0	https://blogs.oracle.com/onesizedoesntfitall/entry/adf_runtimes_vs_wls_versions

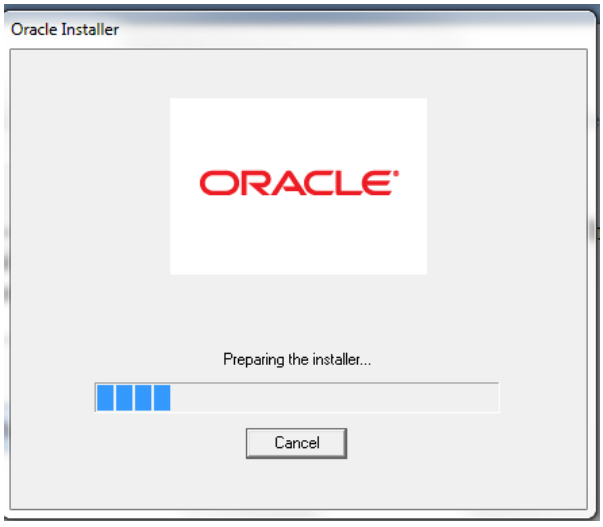
Installation

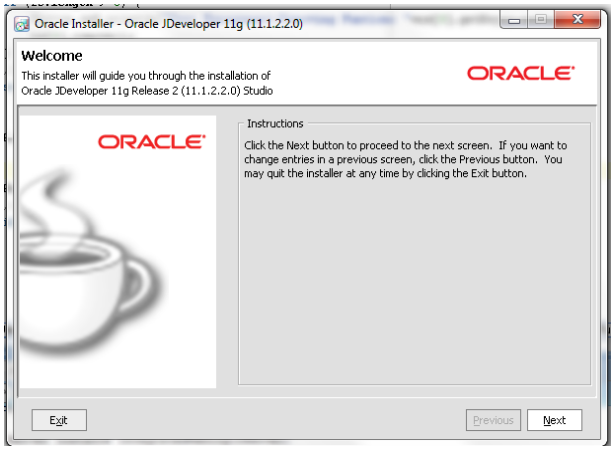
Oracle provides detailed instructions on how to install a local version of. I will just highlight a few steps:

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

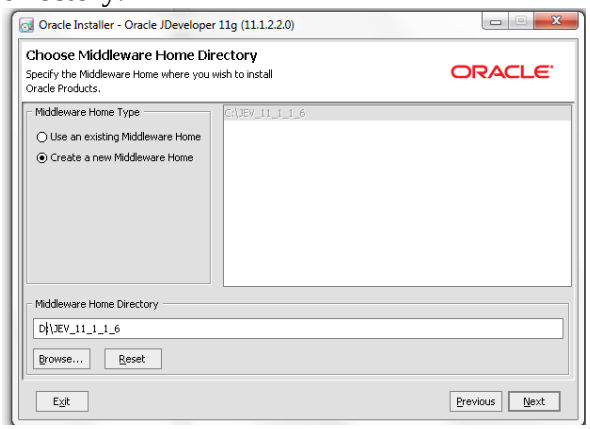


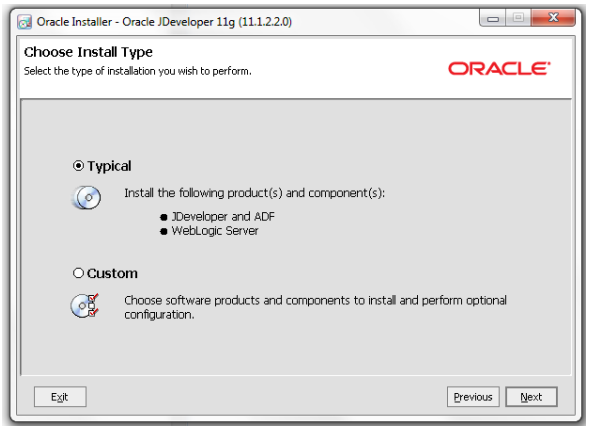
Click On jdevstudio11122install.exe



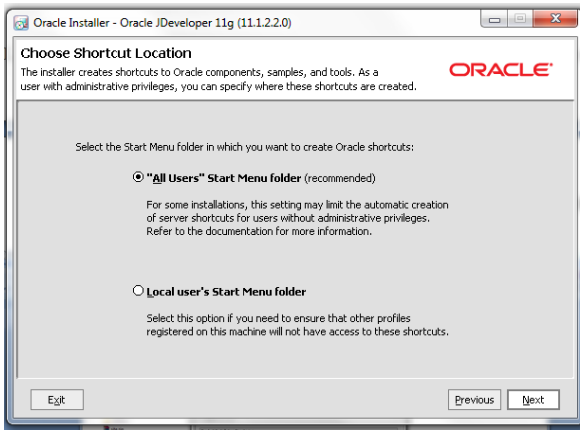


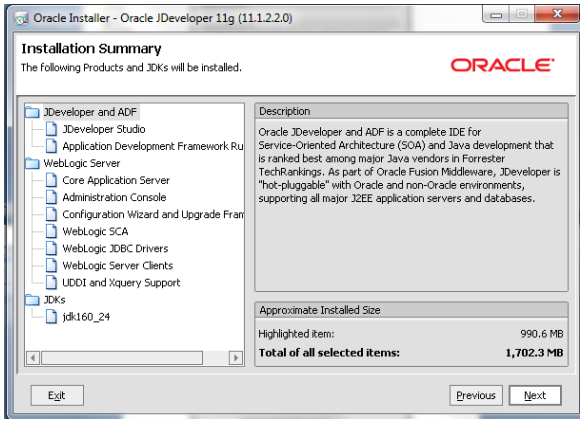
It is recommended you always install each JDeveloper installation in its own directory.

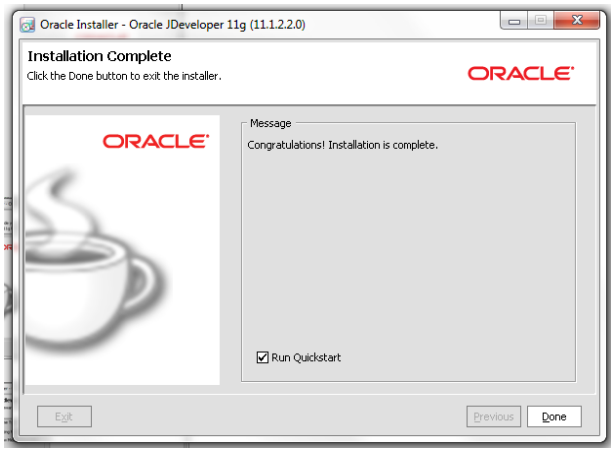





Oracle Installer - Oracle JDeveloper 11g (11.1.2.2.0)
Choose Product Installation Directories
Provide the directories where you wish to install Oracle Products.
ORACLE
Discard Changes
Middleware Home Directory
D:\JEV_11_1_1_6
Product Installation Directories
The Product Home might contain shared utilities and any products or components for which unique directories are not set.
JDeveloper and ADF:
D:\JEV_11_1_1_6\jdeveloper
WebLogic Server:
D:\JEV_11_1_1_6\wlserver_10.3
Exit
Previous
Next


Oracle Installer - Oracle JDeveloper 11g (11.1.2.2.0)
Choose Shortcut Location
The installer creates shortcuts to Oracle components, samples, and tools. As a user with administrative privileges, you can specify where these shortcuts are created.
ORACLE
Select the Start Menu folder in which you want to create Oracle shortcuts:
"All Users" Start Menu folder (recommended)
For some installations, this setting may limit the automatic creation of server shortcuts for users without administrative privileges. Refer to the documentation for more information.
Local user's Start Menu folder
Select this option if you need to ensure that other profiles registered on this machine will not have access to these shortcuts.
Exit
Previous
Next


Oracle Installer - Oracle JDeveloper 11g (11.1.2.2.0)
Installation Summary
The following Products and JDKs will be installed.
ORACLE
JDeveloper and ADF
JDeveloper Studio
Application Development Framework Ru
WebLogic Server
Core Application Server
Administration Console
Configuration Wizard and Upgrade Fram
WebLogic SCA
WebLogic JDBC Drivers
WebLogic Server Clients
UDDI and Xquery Support
JDKs
jdk160_24
Description
Oracle JDeveloper and ADF is a complete IDE for Service-Oriented Architecture (SOA) and Java development that is ranked best among major Java vendors in Forrester TechRankings. As part of Oracle Fusion Middleware, JDeveloper is "hot-pluggable" with Oracle and non-Oracle environments, supporting all major J2EE application servers and databases.
Approximate Installed Size
Highlighted item: 990.6 MB
Total of all selected items: 1,702.3 MB
Exit
Previous
Next



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

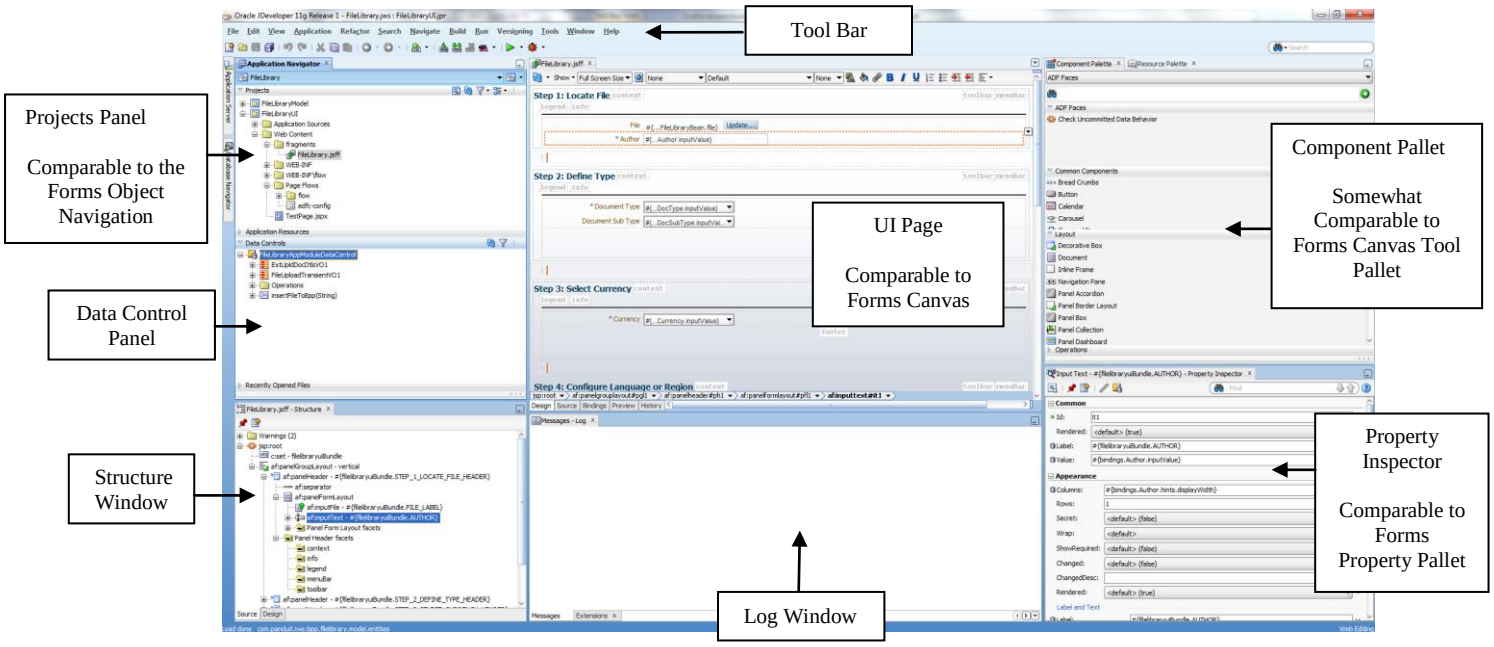
Oracle Install Guide – Screen Shots	http://docs.oracle.com/cd/E35521_01/install.111230/e17074/toc.htm
Oracle Install Guide – Video	https://blogs.oracle.com/workingwithadf/entry/installing_and_configuring_jdeveloper_with_adf

Building Your First Applications

Now that you have installed JDeveloper, I would 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.

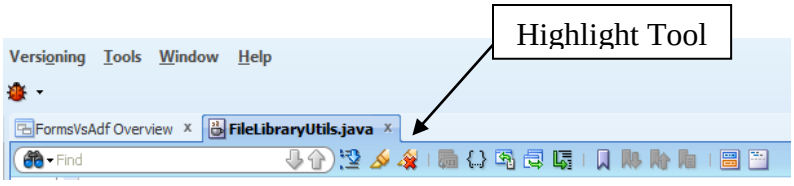
   Figure A: JDeveloper Navigation



Here are some navigation short-cuts:

Ctrl + Mouse Click	Go To Declaration
Ctrl + -	Go To Class
Ctrl + Alt + -	Go To File
Alt + Home	Locate file in app navigator
App Navigator	Find as you type (Comparable to Oracle Forms)
Ctrl + F	Find
Ctrl + Shift + F	Find In Files
Ctrl + Alt + H	Highlight Fields

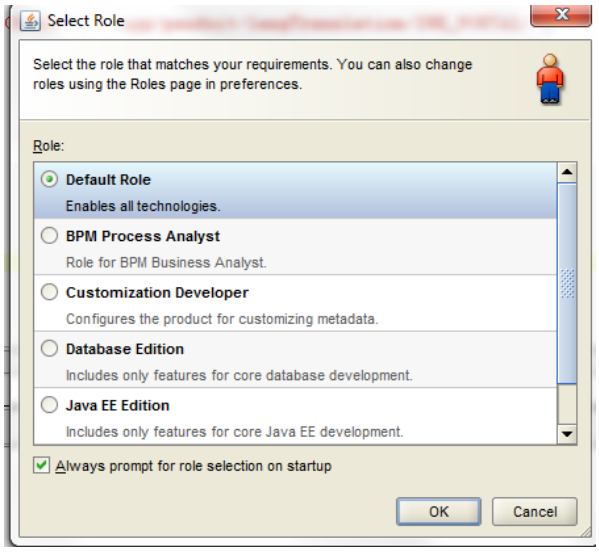
Highlight is a great way to find text within in your custom code and I use this feature all the time.



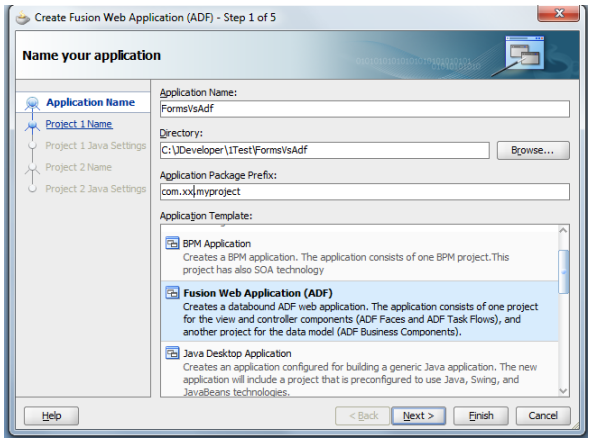
2. Review an Oracle tutorial on how to build your first application. I referenced this URL multiple times when I was building the first applications.

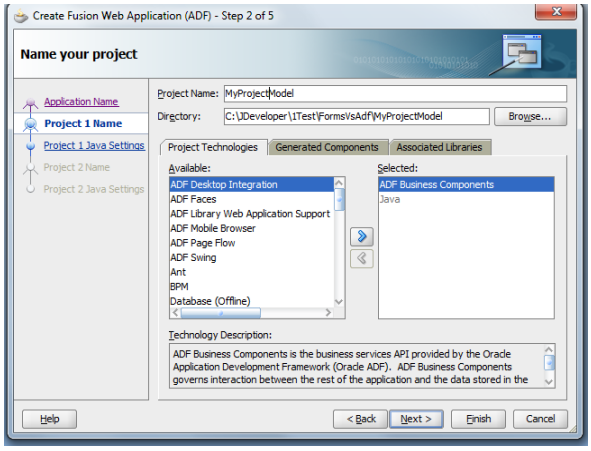
Developing Rich Web Applications With Oracle ADF	http://docs.oracle.com/cd/E18941_01/tutorials/jdtut_11r2_55/jdtut_11r2_55_1.html

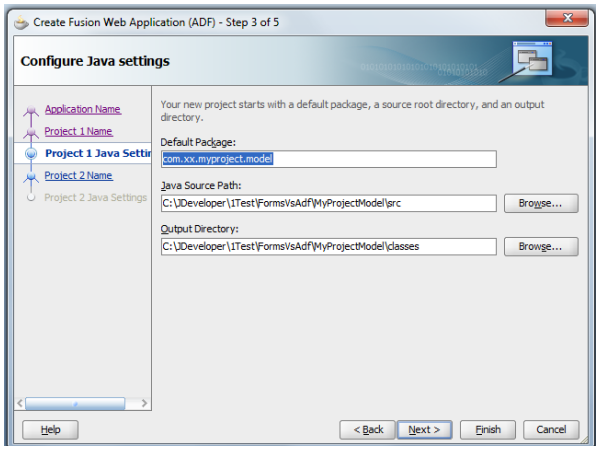
Now it is time to start building your first application. Once you start JDeveloper, it will prompt you to select a role. A role is a way to only display tools and components related to your project. Select the default role until you have researched how this option works.

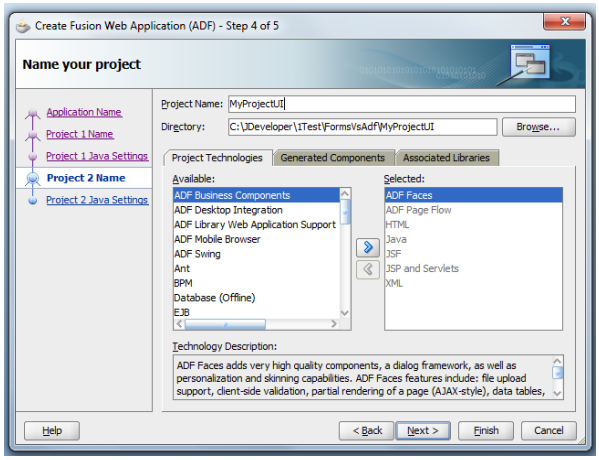


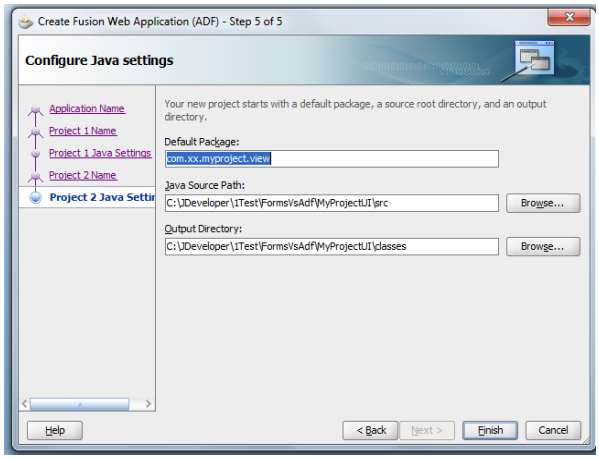
Create a new application:




Create Fusion Web Application (ADF) - Step 2 of 5
Name your project
Application Name
Project 1 Name
Project 1 Java Settings
Project 2 Name
Project 2 Java Settings
Project Name: MyProjectModel
Directory: C:\JDeveloper\1Test\FormsVsAdf\MyProjectModel
Browse...
Project Technologies
Generated Components
Associated Libraries
Available:
ADF Desktop Integration
ADF Faces
ADF Library Web Application Support
ADF Mobile Browser
ADF Page Flow
ADF Swing
Ant
BPM
Database (Offline)
Selected:
ADF Business Components
Java
Technology Description:
ADF Business Components is the business services API provided by the Oracle Application Development Framework (Oracle ADF). ADF Business Components governs interaction between the rest of the application and the data stored in the
Help
< Back
Next >
Finish
Cancel

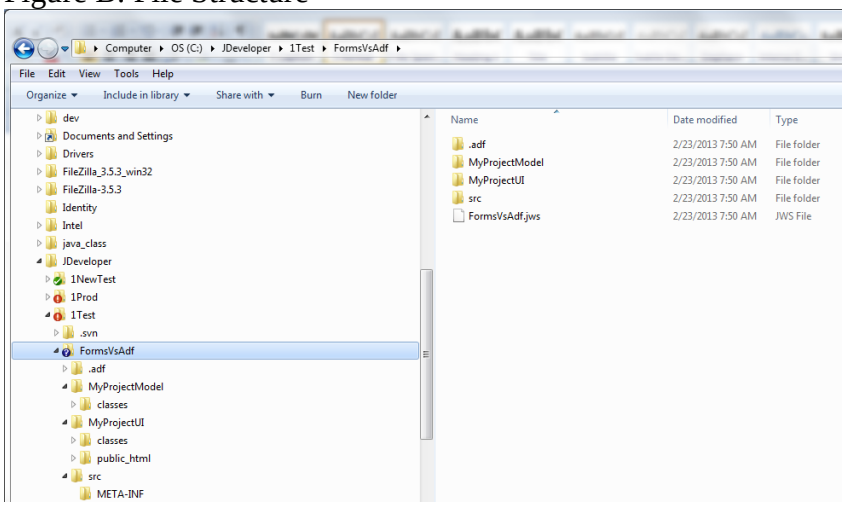

Create Fusion Web Application (ADF) - Step 3 of 5
Configure Java settings
Application Name
Project 1 Name
Project 1 Java Settir
Project 2 Name
Project 2 Java Settings
Your new project starts with a default package, a source root directory, and an output directory.
Default Package:
com.xx.myproject.model
Java Source Path:
C:\JDeveloper\1Test\FormsVsAdf\MyProjectModel\src
Browse...
Output Directory:
C:\JDeveloper\1Test\FormsVsAdf\MyProjectModel\classes
Browse...
Help
< Back
Next >
Finish
Cancel


Create Fusion Web Application (ADF) - Step 4 of 5
Name your project
Application Name
Project 1 Name
Project 1 Java Settings
Project 2 Name
Project 2 Java Settings
Project Name: MyProjectUI
Directory: C:\JDeveloper\1Test\FormsVsAdf\MyProjectUI
Browse...
Project Technologies
Generated Components
Associated Libraries
Available:
ADF Business Components
ADF Desktop Integration
ADF Library Web Application Support
ADF Mobile Browser
ADF Swing
Ant
BPM
Database (Offline)
EJB
Selected:
ADF Faces
ADF Page Flow
HTML
Java
JSF
JSP and Servlets
XML
Technology Description:
ADF Faces adds very high quality components, a dialog framework, as well as personalization and skinning capabilities. ADF Faces features include: file upload support, client-side validation, partial rendering of a page (AJAX-style), data tables,
Help
< Back
Next >
Finish
Cancel

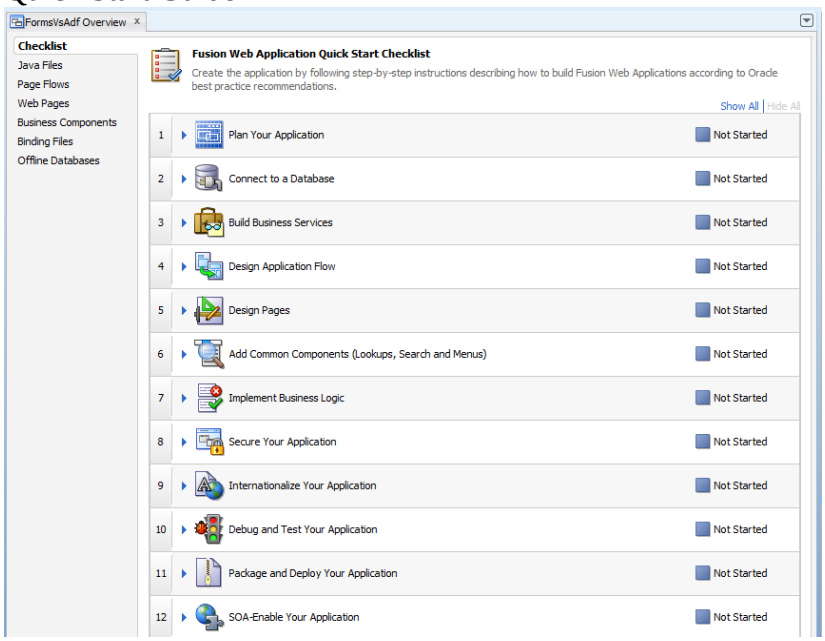


Once you click on finish, the application will set up the directory structure on your file system and open up the application with a Quick Start Guide. This is comparable to the Oracle Forms Wizard that helps you build your application. I would recommend using the ADF Quick Start Guide to build the first applications. As you learn the JDeveloper tool, you will probably build your application without the Wizard.

Figure B: File Structure



Quick Start Guide

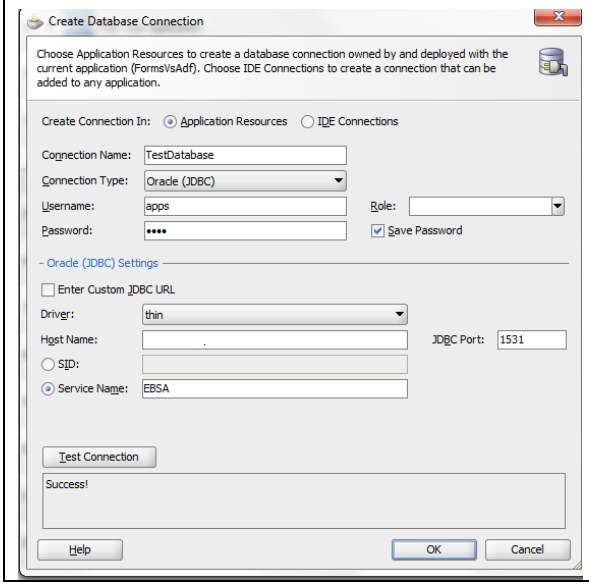
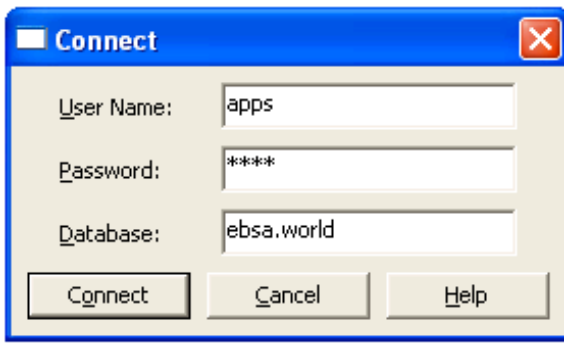


Now it is time to back up your application. If you ever worked with Oracle Reports or Oracle Forms, the first thing you learn is to back-up your code often. This same holds true for Oralce ADF. There are so many additional options that you are bound to make a mistake that will require you to restore 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 concern, with a database connection to a single database and schema.

JDeveloper	Oracle Forms Developer
	

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.

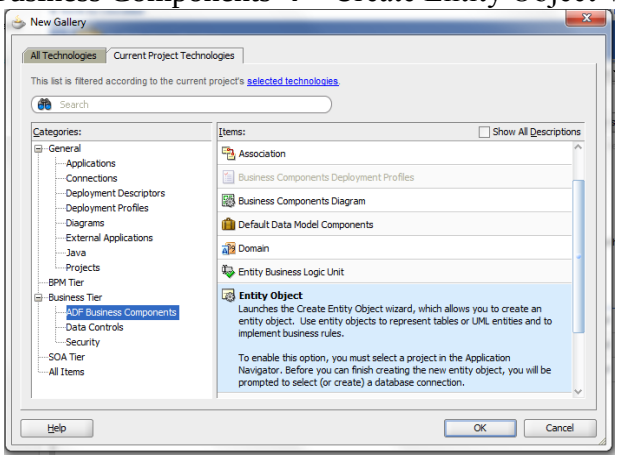
In ADF, CRUD operations are performed on entity objects or database tables. This is the reason why ADF requires a separate updatable view objects to be used by the UI. Entity objects are used by the ADF framework to automatically handle CRUD operations and view objects are the mechanism to maintain the 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).

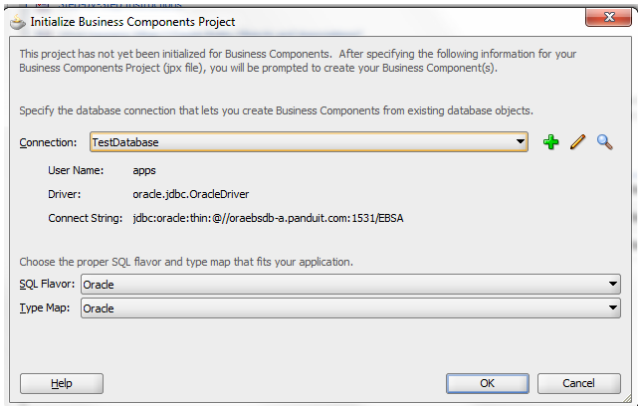
In terms of Oracle Forms, it can be argued that each Oracle Form application contains a virtual application module because it is a collection of view objects that are implicitly exposed to the user interface or canvases.

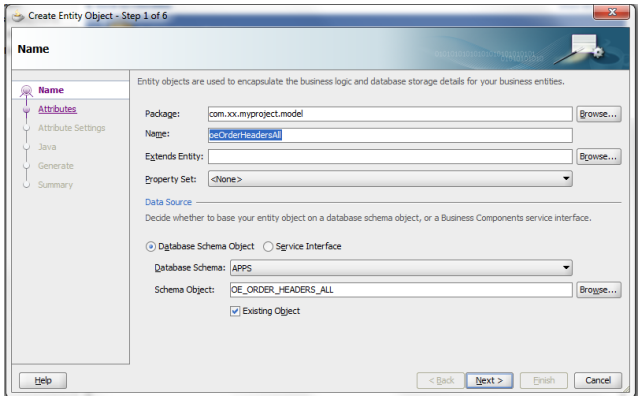
Creating Business Service Objects

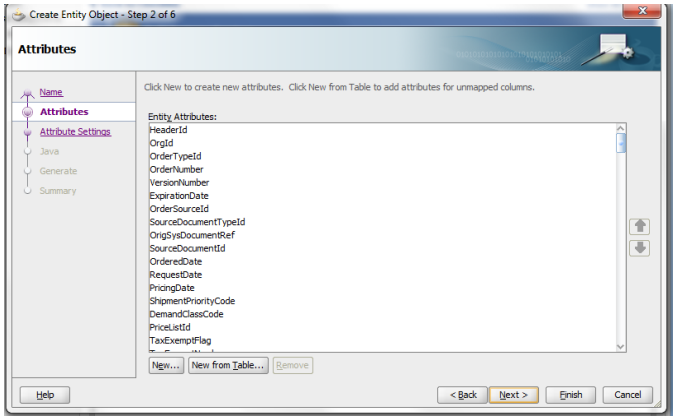
In order to assist you in creating Entity and View objects, JDeveloper has created various wizards to reduce the time to build your application. When you need to create an Entity

object, place your cursor on the Model, or "MyProjectModel" and right click and select "New". A pop-up gallery window will then appear. Select Business Tier → ADF Business Components → "Create Entity Object Wizard" to launch the entity wizard.

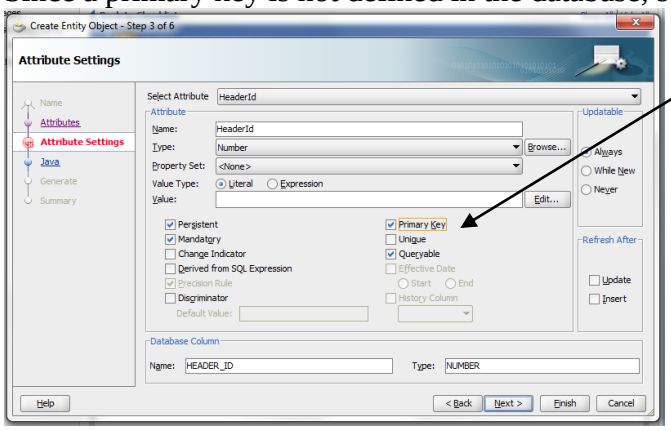


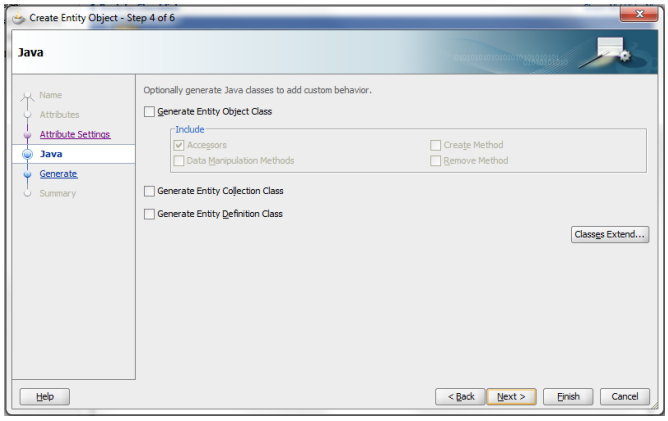


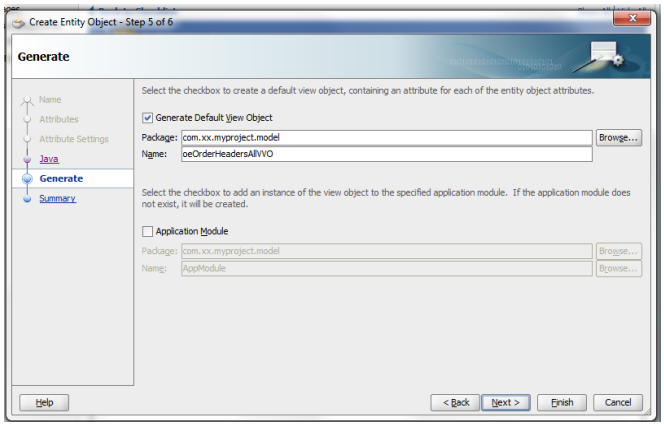


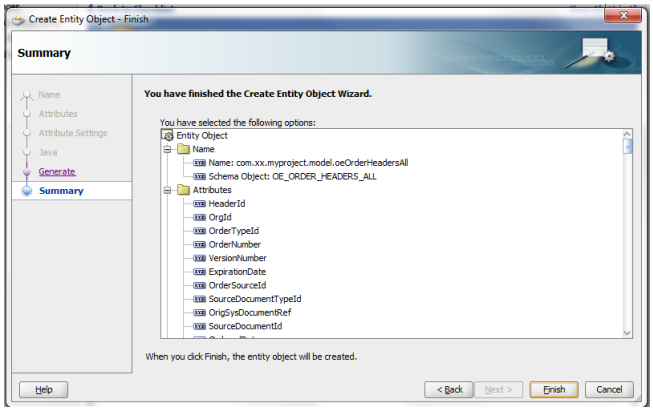


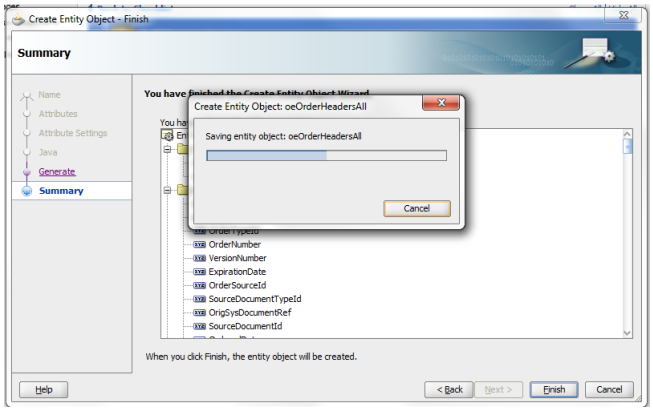
Since a primary key is not defined in the database, select primary key.





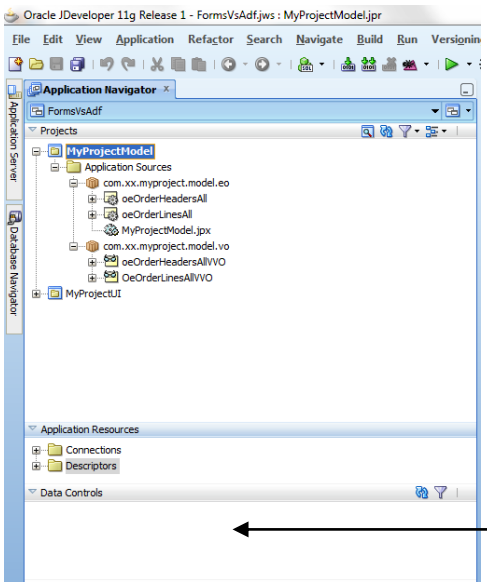






Perform the same steps for oe_order_lines_all

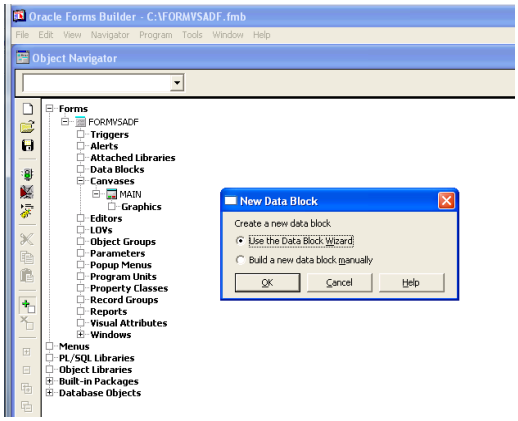
When you are done, your project should look like

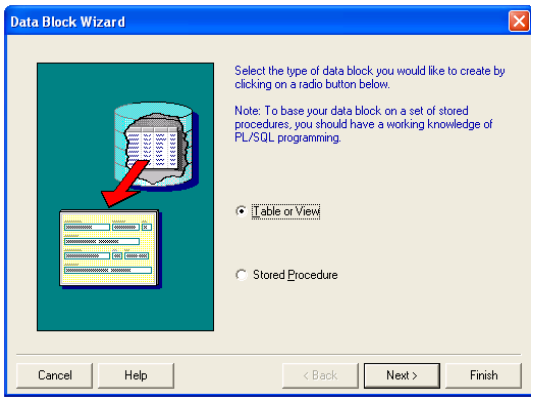


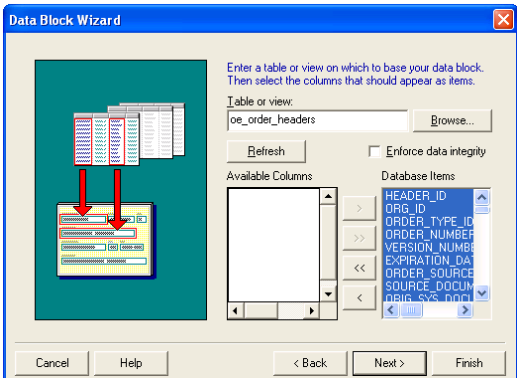
We have not exposed any view objects so nothing is showing under data controls

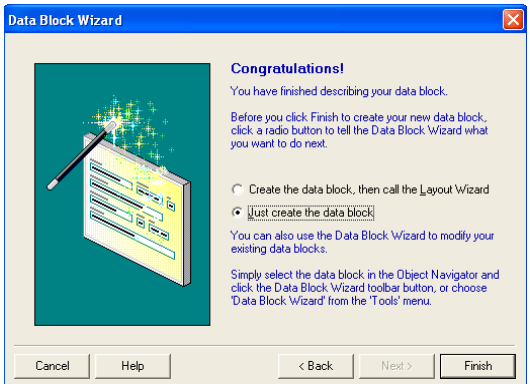
In Oracle Forms, database view objects perform roughly the same function as an ADF view object. It is the mechanism to maintain the data in your user interface. As with ADF framework, Oracle Forms framework has wizards to create your Data Block (view objects):

Create data blocks (entity & view objects).

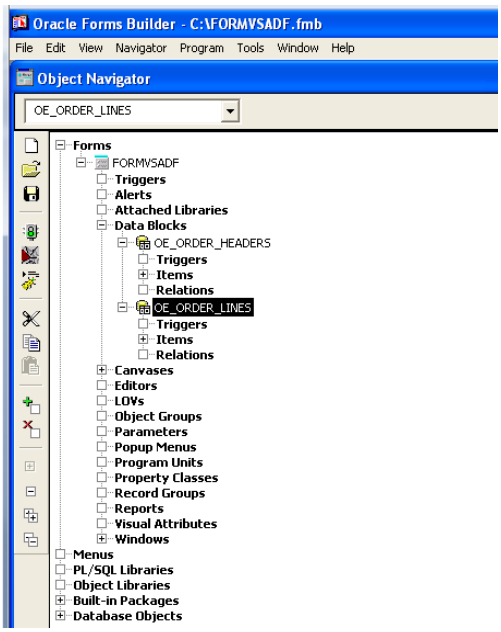



Data Block Wizard
Select the type of data block you would like to create by clicking on a radio button below.
Note: To base your data block on a set of stored procedures, you should have a working knowledge of PL/SQL programming.
Table or View
Stored Procedure
Cancel
Help
< Back
Next >
Finish


Data Block Wizard
Enter a table or view on which to base your data block.
Then select the columns that should appear as items.
Table or view:
oe_order_headers
Browse...
Refresh
Enforce data integrity
Available Columns
Database Items
HEADER_ID
ORG_ID
ORDER_TYPE_ID
ORDER_NUMBER
VERSION_NUMBE
EXPIRATION_DAT
ORDER_SOURCE
SOURCE_DOCUM
ORIG_SYS_DOCU
Cancel
Help
< Back
Next >
Finish


Data Block Wizard
Congratulations!
You have finished describing your data block.
Before you click Finish to create your new data block, click a radio button to tell the Data Block Wizard what you want to do next.
Create the data block, then call the Layout Wizard
Just create the data block
You can also use the Data Block Wizard to modify your existing data blocks.
Simply select the data block in the Object Navigator and click the Data Block Wizard toolbar button, or choose 'Data Block Wizard' from the 'Tools' menu.
Cancel
Help
< Back
Next >
Finish

Perform the same steps for oe_order_lines and when you are done the Forms navigator screen should look like

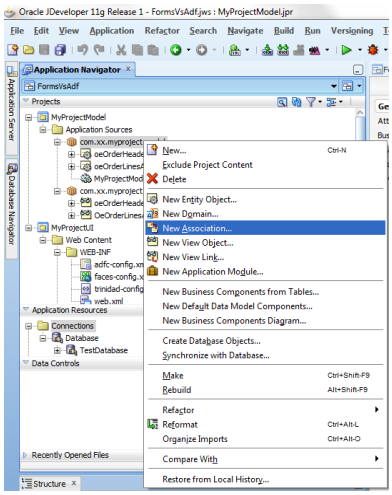


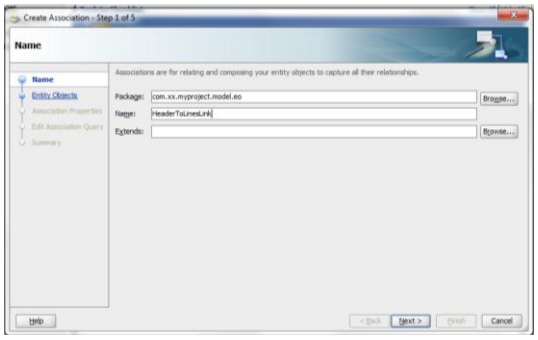
In ADF, the next step is to create a master-child relationship between the headers and lines table. The relationship is connected by a database column or header_id. ADF supports two different styles of master-detail relationship:

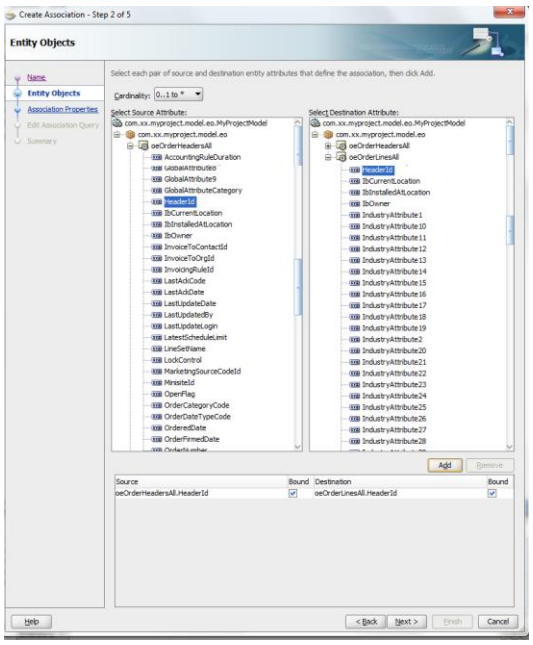
Association This is a master-detail relationship defined at the entity level and since it is defined at the entity level, each entity object can reference each other in CRUD operations. As a general rule, create an association link when you need to enforce business rules, create or delete records in 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.

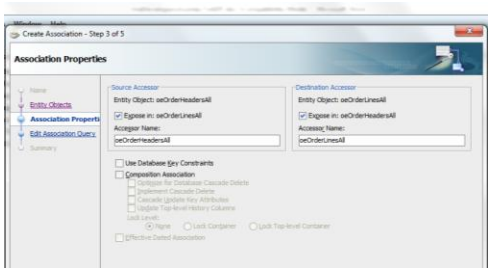
View Link This is a master-detail relationship defined at the view level and is mainly used to sync up blocks of data in your application. If you are creating a master-detail relationship screen in your application, then you should create a view link because it supports automatic master-detail synchronization.

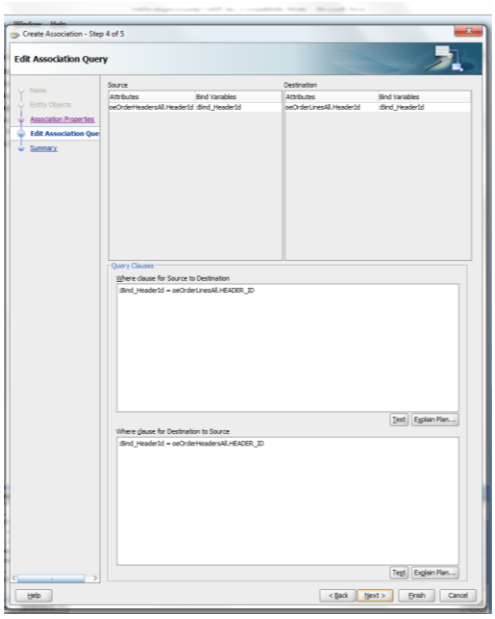
Here are the steps to create an association link

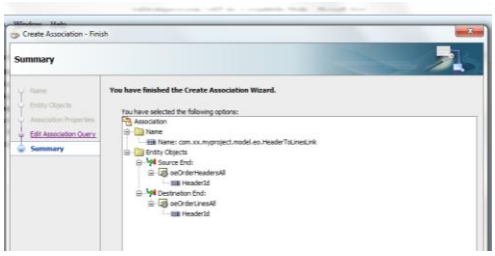




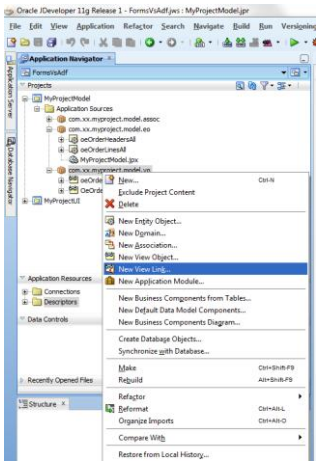


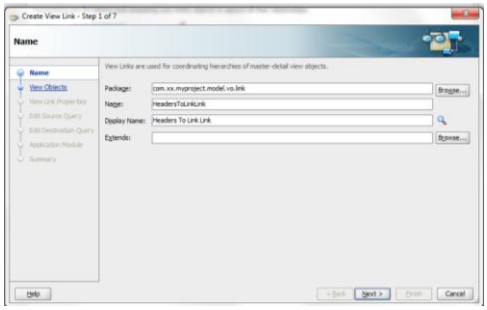

Create Association - Step 3 of 5
Association Properties
Entity Objects
Association Properti
Edit Association Query
Summary
Source Accessor
Entity Object: oeOrderHeadersAll
Expose in: oeOrderLinesAll
Accessor Name:
oeOrderHeadersAll
Destination Accessor
Entity Object: oeOrderLinesAll
Expose in: oeOrderHeadersAll
Accessor Name:
oeOrderLinesAll
Use Database Key Constraints
Composition Association

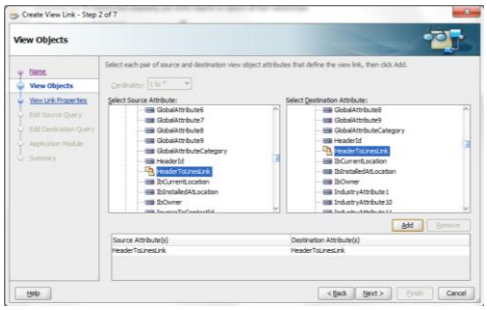

Create Association - Step 4 of 5
Edit Association Query
Name
Entity Objects
Association Properties
Edit Association Que
Summary
Source
Destination
Attributes
Bind Variables
oeOrderHeadersAll.HeaderId
:Bind_HeaderId
oeOrderLinesAll.HeaderId
:Bind_HeaderId
Query Clauses
Where clause for Source to Destination
:Bind_HeaderId = oeOrderLinesAll.HEADER_ID
Test
Explain Plan...
Where clause for Destination to Source
:Bind_HeaderId = oeOrderHeadersAll.HEADER_ID
Test
Explain Plan...
Help
< Back
Next >
Finish
Cancel

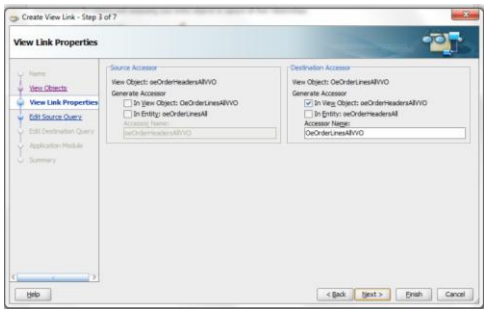

Create Association - Finish
Summary
Name
Entity Objects
Association Properties
Edit Association Query
Summary
You have finished the Create Association Wizard.
You have selected the following options:
Association
Name
Entity Objects
Source End:
oeOrderHeadersAll
HeaderId
Destination End:
oeOrderLinesAll
HeaderId

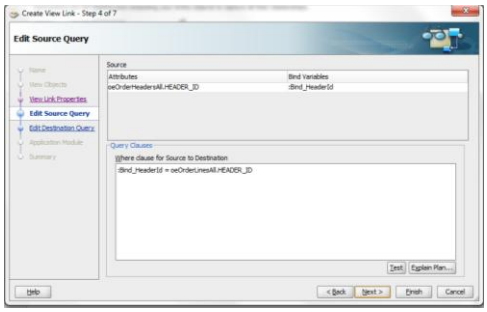
Here are the steps to create a view link:

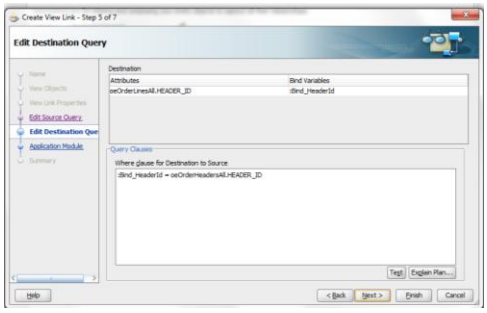


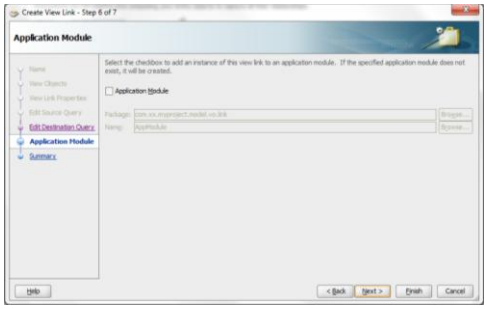


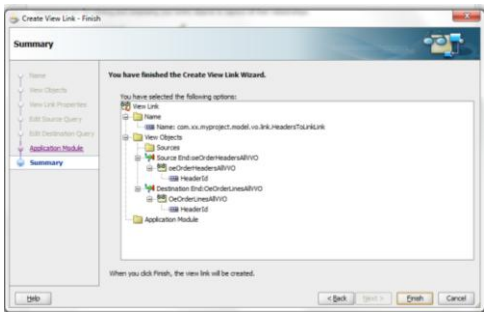




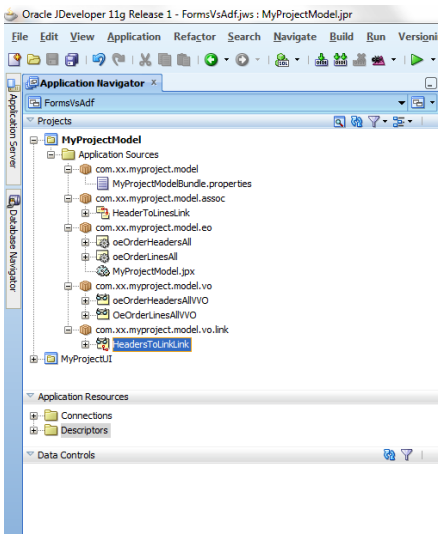

Create View Link - Step 4 of 7
Edit Source Query
Source
Attributes
Bind Variables
oeOrderHeadersAll.HEADER_ID
:Bind_HeaderId
Query Clauses
Where clause for Source to Destination
:Bind_HeaderId = oeOrderLinesAll.HEADER_ID
Test
Explain Plan...
Help
< Back
Next >
Finish
Cancel


Create View Link - Step 5 of 7
Edit Destination Query
Destination
Attributes
Bind Variables
oeOrderLinesAll.HEADER_ID
:Bind_HeaderId
Query Clauses
Where clause for Destination to Source
:Bind_HeaderId = oeOrderHeadersAll.HEADER_ID
Test
Explain Plan...
Help
< Back
Next >
Finish
Cancel


Create View Link - Step 6 of 7
Application Module
Application Module
Help
< Back
Next >
Finish
Cancel

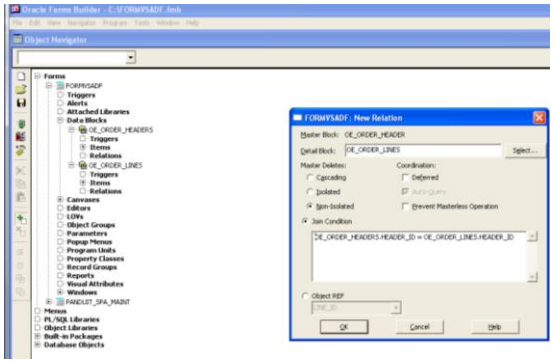

Create View Link - Finish
Summary
You have finished the Create View Link Wizard.
You have selected the following options:
View Link
Name
View Objects
Sources
Application Module
When you click Finish, the view link will be created.
Help
< Back
Finish
Cancel

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

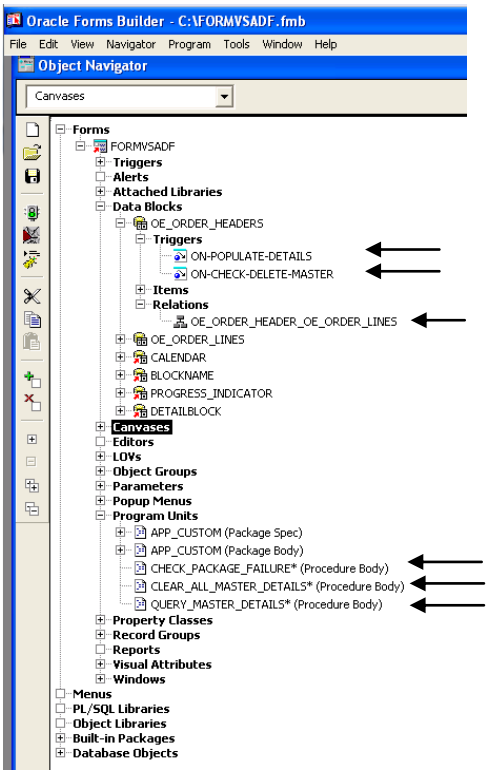


In Oracle Forms, we do not have a distinction between table based and view link master-detail relationship. We have a single relationship, which is a combination of an association and view link. It supports auto-query of the detail block and enforces cascade deletes.

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



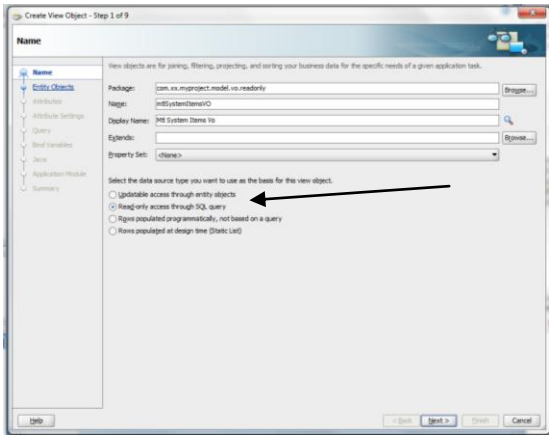
Once the relationship is created, Oracle Forms will automatically add code to enforce the relationship.



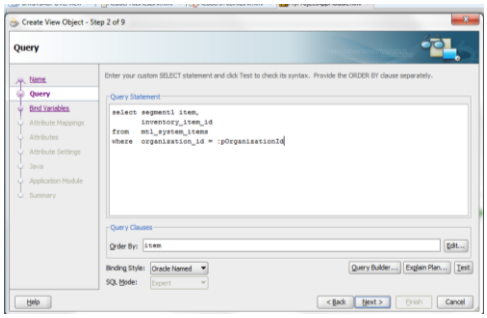
Now that we have our business relationships defined, we need to apply a common business functionality to help our Users understand the data. One such business component is a list of values (LOV) component, which translates database Ids to readable values.

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 our displayed below:

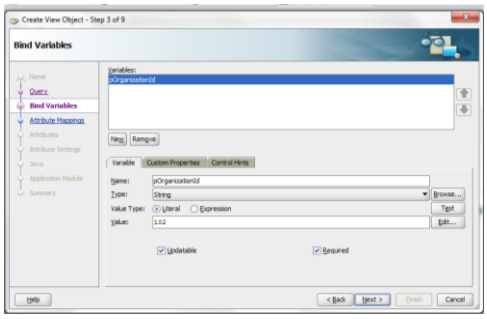
1. Create a read-only view



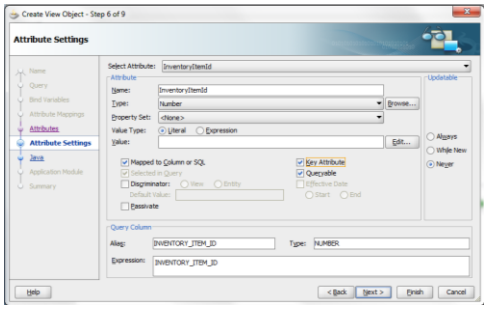
2. Add your SQL statement with a parameter



3. Define your parameter

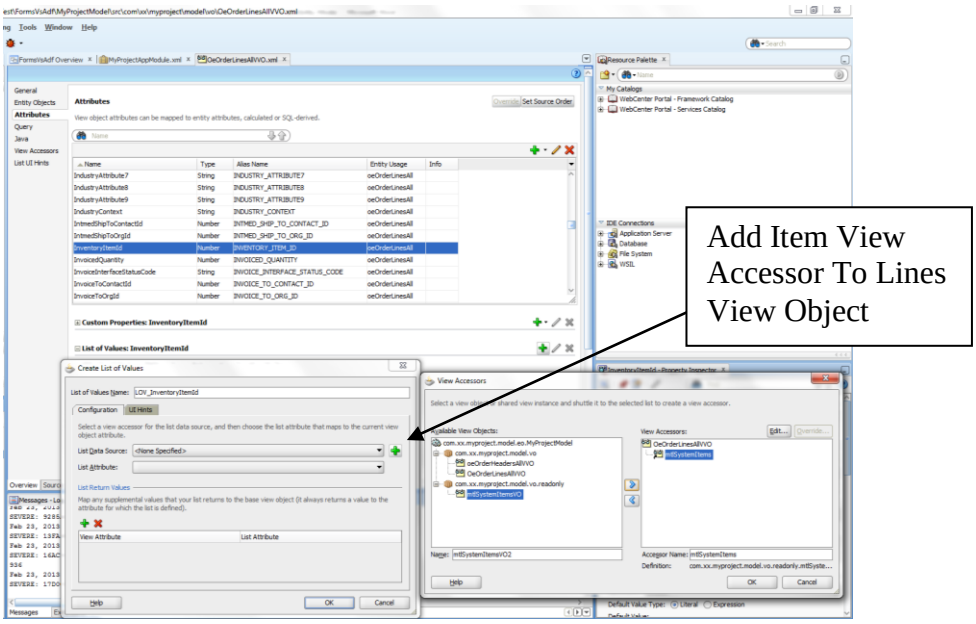


4. Identify what column in your view is the primary key

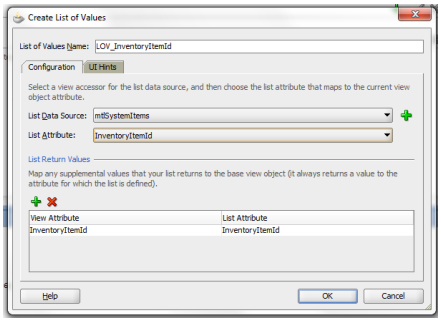


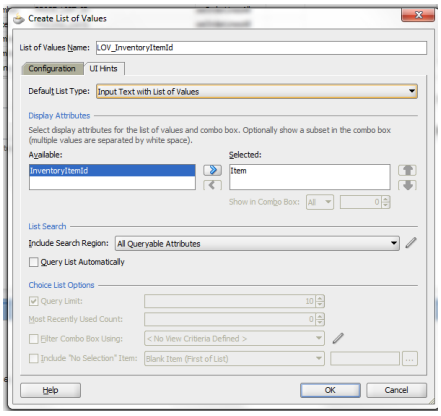
Click Finish

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

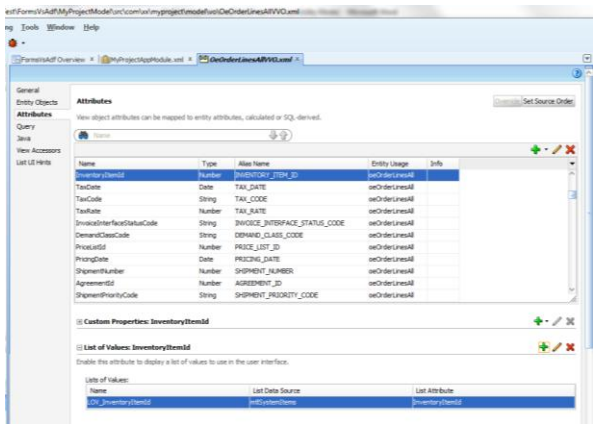


Once the view accessor is added to the line view object, it will be available in the "List Data Source" drop down list. Select the list data source and complete the mapping.





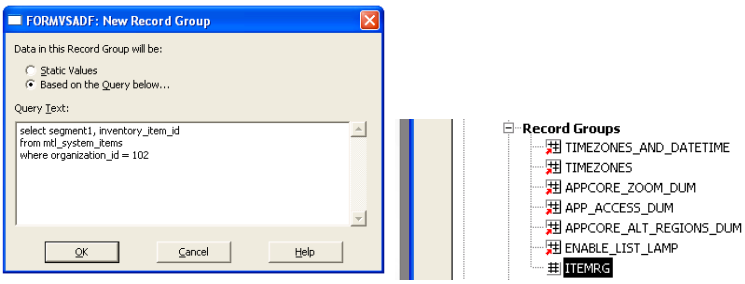
After the list of value is attached, you will see the list-of-values in the attribute window.



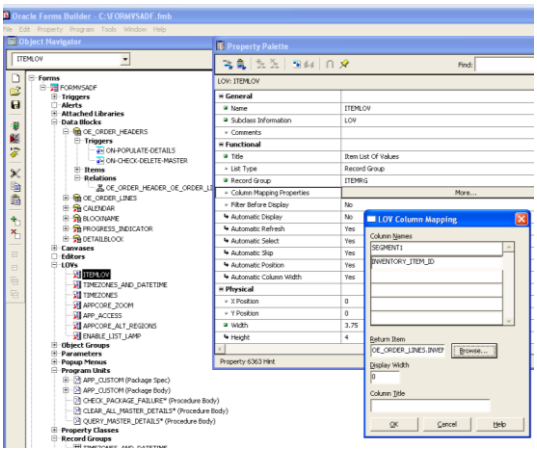
This is very similar to creating a list of values in Oracle Forms. First, a record group (view object) needs to be created. Then we need to create a list of values object, which is an explicit mapping of the record group result set to a data block definition. Lastly, we need to link the list of values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

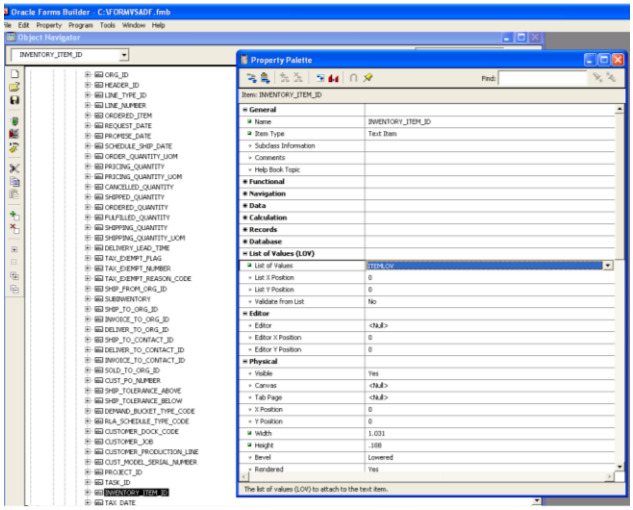
1. Create a record group which is equivalent to a read-only view object in ADF



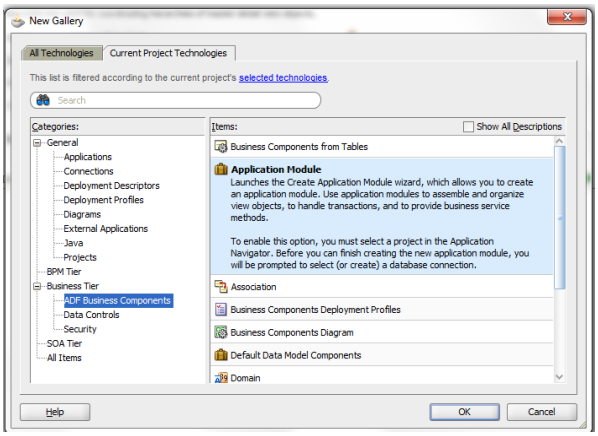
2. Create a list of values (LOV) mapping between the definition of the result set and the Data Block field.

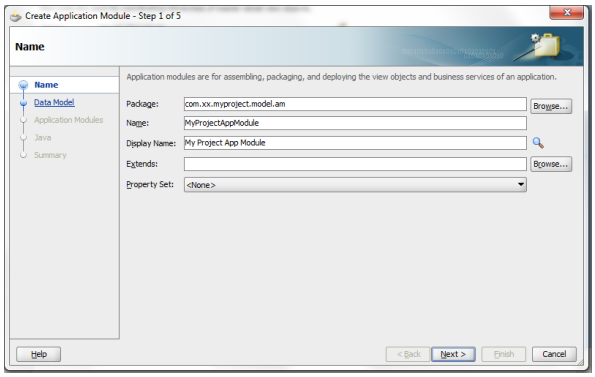


3. Link the list of values to a data block field

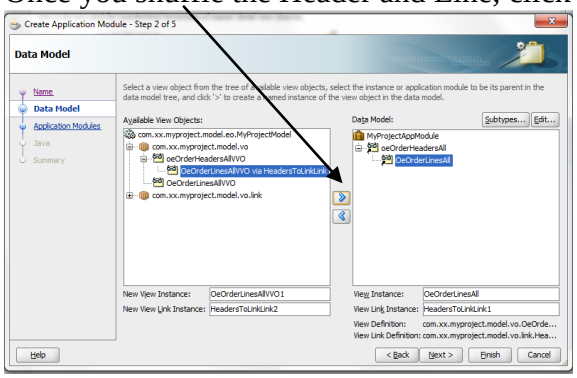


Once we have all our business services (entity objects, view objects, links and list of values objects) completed, our next step is to group the view objects together into application module so they can be exposed to the View layer. Here are the steps to create an application module in ADF:

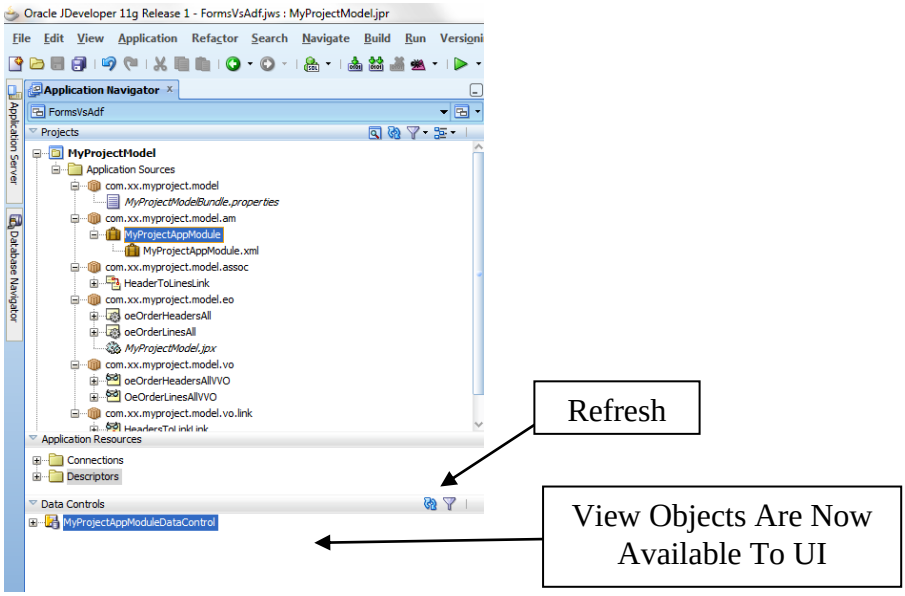




Once you shuffle the Header and Line, click finish.



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

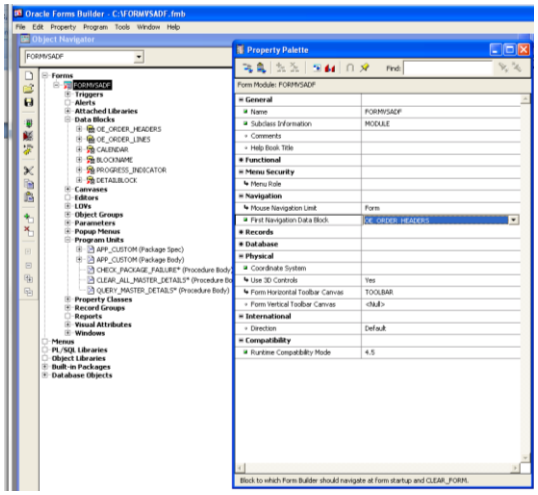
For either Oracle Forms or ADF applications, we need to initialize the session when the application is launched. The purpose of the initializing is to prepare the application for the User who is about to enter into the application. In Oracle Forms, this is performed by the "When-New-Form" trigger which 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 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.



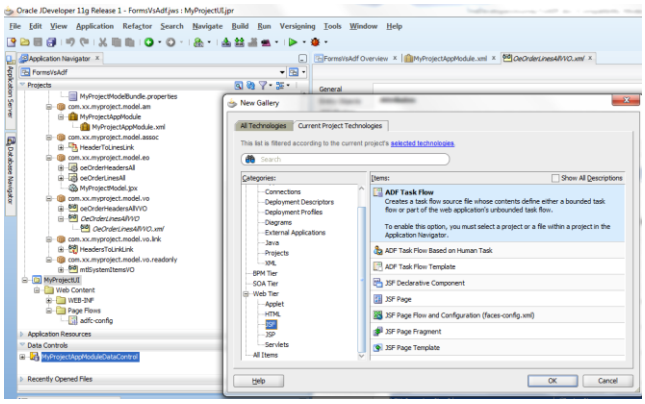
The initialization of the Oracle Forms session and the option to set the initial canvas is set by the "Pre-Form" or the "When-New-Form" trigger. Here is an example of some PL/SQL code that is called by the form level trigger WHEN-NEW-FORM-INSTANCE:

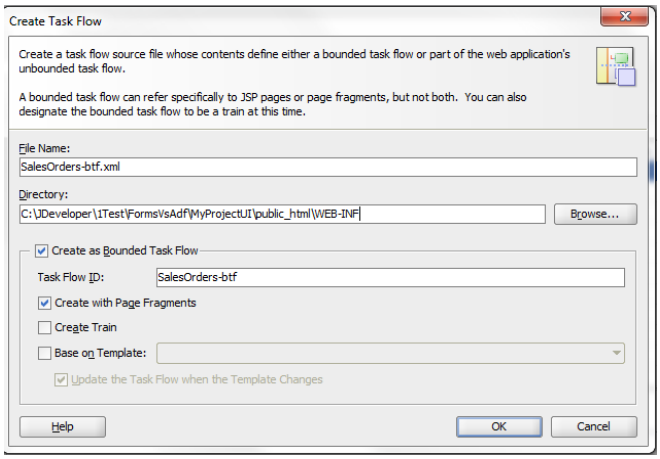
```
PROCEDURE when_new_form_instance
```

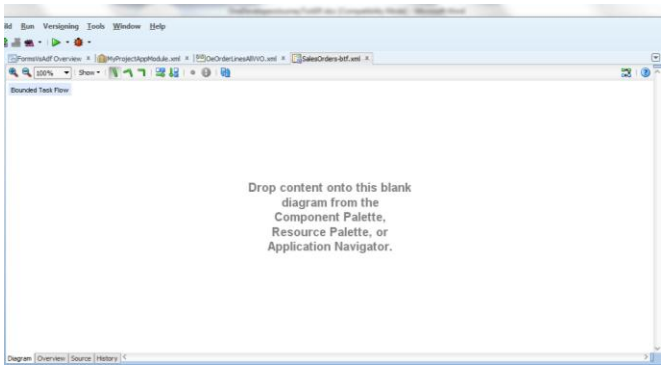
```
IS
     win_msg           VARCHAR2(2000);
     l_org_id          NUMBER := TO_NUMBER(FND_PROFILE.VALUE('ORG_ID'));
     l_user_id         NUMBER := TO_NUMBER(FND_PROFILE.VALUE('USER_ID'));
     l_percision       NUMBER := NULL;
     l_name            hr_operating_units.name%TYPE;

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

/*
|| 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.

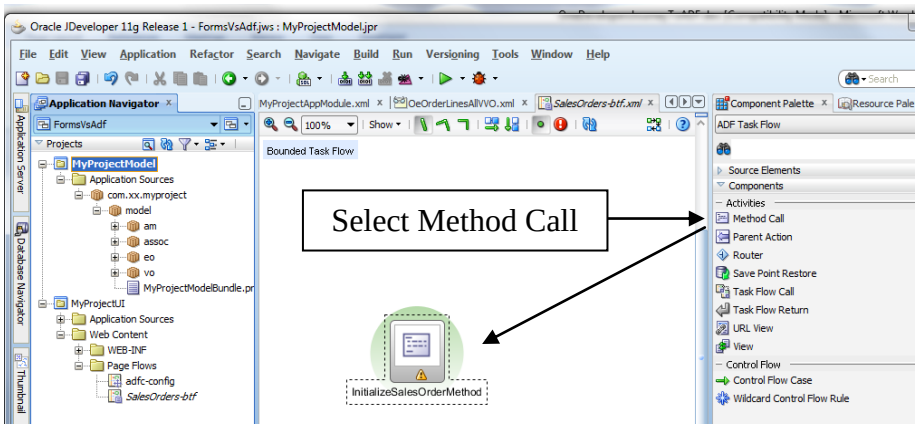
In ADF, you control the flow of your application by using a task flow. A task flow is a visual way to control the flow of your application. When you initially create your bounded task flow object, the task flow will be completely empty and it is your responsibility to visually define your flow. Once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



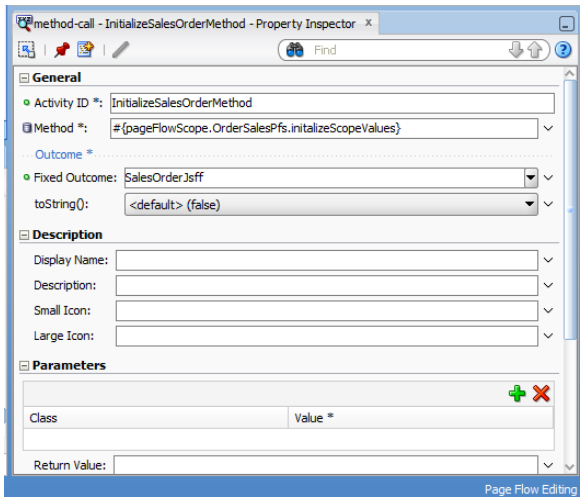




In ADF, we initialize our application by adding a “Method Call” activity in the task flow and we set this activity as the default. The Method Call activity will reference a managed bean method or application module method. In this example, I am referencing a managed bean method.



In this example, I named the "Method Call" activity InitializeSalesOrderMethod and it references a managed bean called initalizeScopeValues.

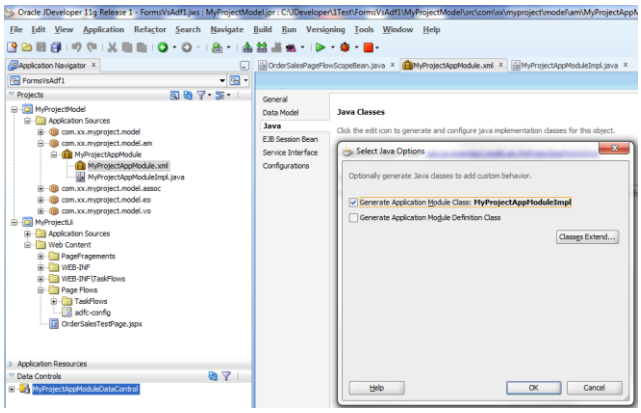


Here is a sample code that would be contained in the managed bean method. In this case, we are referencing an application parameter, which is defined in your task flow. In addition, it is calling an Application Module (AM) method that will initialize the EBS database session variables.

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

        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 Initalize Session Variables In The Model
          BindingContainer bc = BindingContext.getCurrent().getCurrentBindingsEntry();
          bc.getOperationBinding("setAMSessionVariables").execute();


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

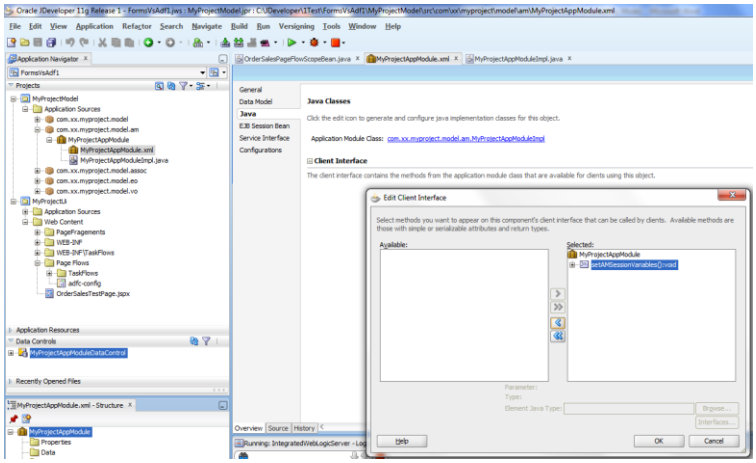
In order to reference an AM method, you first need to create an Application Module Java class, add that method to the Application Module so it can be exposed via a Data Controller, which in turn allows you to consume and bind that AM method to Method Call Activity or UI Page. Here are the steps to create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



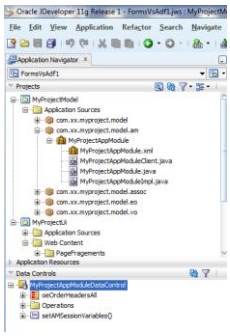
Open the newly created Java Class and write your new method.

```
    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");
    }
```

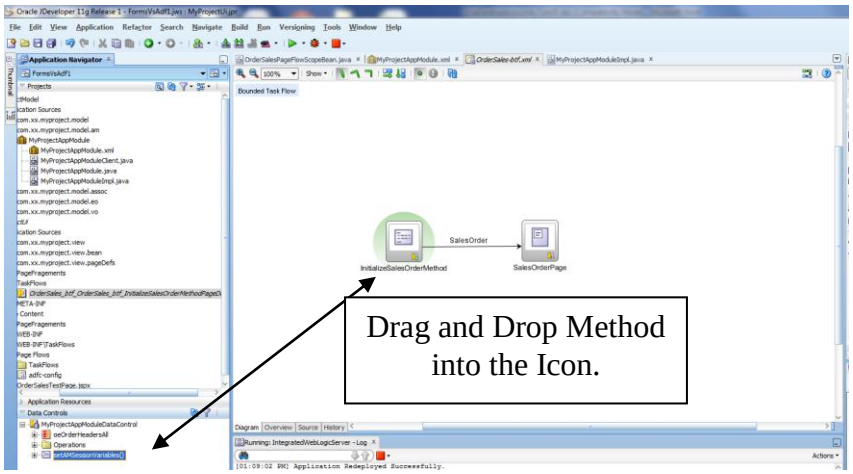
Add the new method to the Client Interface.



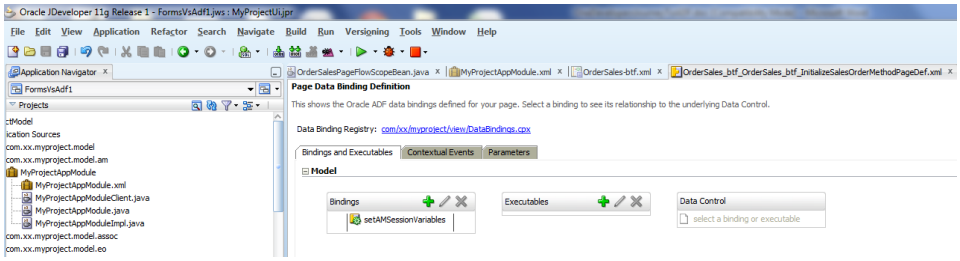
Refresh your Data Control Panel and you will see that the method is now exposed and can be consumed by the Method Call Activity in the View Layer.



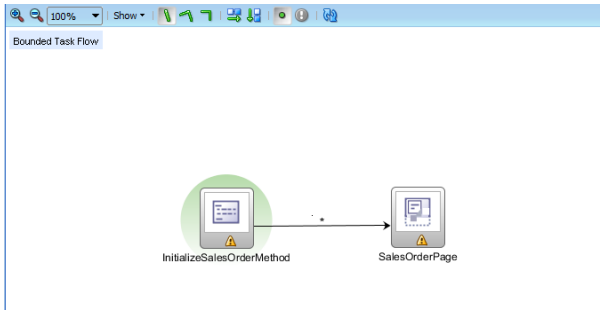
The last set is to bind the exposed AM method, or "setAMSessionVariables" to your Method Call Activity, or InitializeSalesOrderMethod.



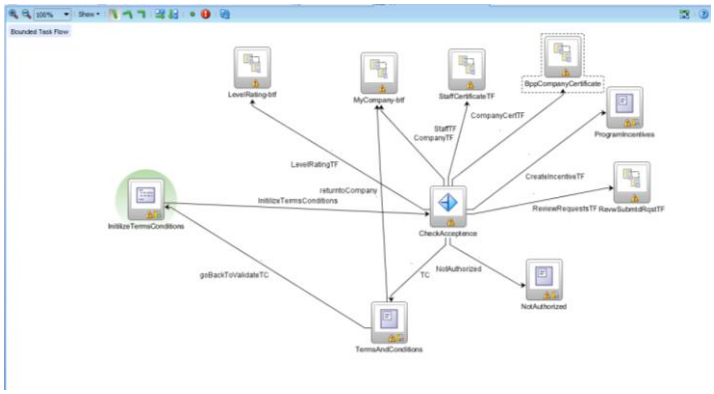
Note: Once you drag and drop the exposed method onto the Method Call Activity, the ADF framework will automatically create the binding for you.



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.

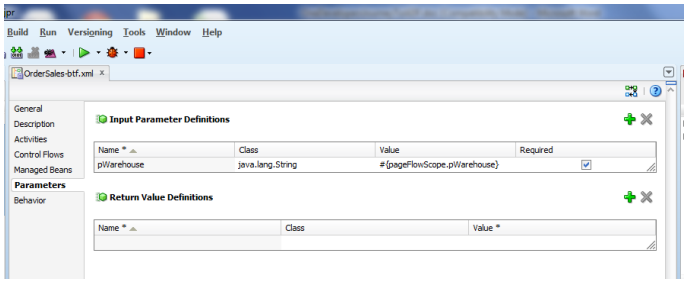


In a real life ADF application, the task flow design can quite complex.

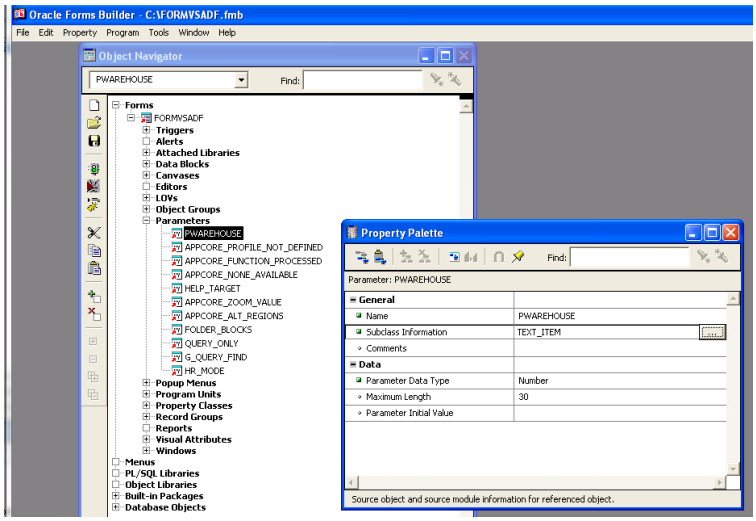


Application Parameters

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

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	http://www.youtube.com/watch?v=TiqbW1CAMMc

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 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	http://www.quovera.com/whitepapers/downloads/rmoug_2012_deployment_doc.pdf
FAQ for Integration of Oracle E-Business Suite and Oracle Application Development Framework (ADF) Applications	https://metalink.oracle.com/metalink/plsql/showdoc?db=NOT&id=1296491.1

Conclusion

In conclusion, transitioning from Oracle Forms to ADF was overwhelming, time consuming, always challenging, sometimes frustrating; however, the most important thing to remember is that it is possible. My journey started with no knowledge of ADF. It continues with spending hours exploring the internet to find documentation and videos to assist in my understanding of ADF, countless attempts to build a single application that would work from end-to-end, to attending ADF design meetings where I would write down five to ten ADF concepts or terms that I need to research after each meeting, to building my first application where I need ask ten "how to" questions of day, to building an application on my own and concluding with supporting that application in a production environment.

I wrote write this paper because I know there are other Oracle Forms Developers that still need to take on this journey and I want to provide a high level how-to guide. Secondly, I wanted to share some of the great references that helped me gain my understanding of ADF. My understanding of ADF grows daily and it has been nine months since I started

my journey and I have to say that I just scratched the surface of ADF's capabilities, so be patient and take that first step into the world of ADF.