



# ILP PROGRAM - ORACLE APPLICATIONS

## Tata Consultancy Services OA Framework Study Guide

Author: [MD. Nazmul Hoda \(TCS\)](#)  
Creation Date: Jan 7, 2015  
Last Updated: Jan 7, 2015  
Document Ref: [ILP/ORACLEAPPS/OAF/01](#)  
Version: DRAFT 1A

### Approvals:

<Approver 1>

Santanu Sarkar (TCS)

---

<Approver 2>

Shubho Chakraborty(TCS)

---

---

## Document Control

---

### Change Record

Date	Author	Version	Change Reference
07-May-15	<a href="#">MD. Nazmul Hoda</a>	Draft 1a	No Previous Document

---

### Reviewers

Name	Position

---

### Distribution

Copy No.	Name	Location
1	Library Master	Project Library
2		Project Manager
3		
4		

#### Note To Holders:

If you receive an electronic copy of this document and print it out, please write your name on the equivalent of the cover page, for document control purposes.

If you receive a hard copy of this document, please write your name on the front cover, for document control purposes.

## Contents

### Table of Contents

<b>Document Control.....</b>	<b>2</b>
How to use this manual .....	5
Introduction to OA Framework:.....	6
Setting Up your Development Environment:.....	7
OA Framework Development Runtime Configuration:.....	14
OA Framework Essentials.....	16
Anatomy of an OA Framework Page:.....	16
Page Basics: .....	16
The Model: .....	17
The View:.....	18
Types of Regions in a Page:.....	19
Types of Items in a Page:.....	29
The Controller: .....	33
Web Bean Architecture:.....	34
Internal Bean Structure:.....	35
Application Module Retention Level:.....	39
Building an OA Framework Applications (Basics): .....	43
Implementing the Model: .....	43
Designing Model Objects: .....	43
Business Components Packages: .....	45
Designing the User Interface:.....	46
Implementing the Controller: .....	47
Designing an OA Controller:.....	47
Handling an HTTP Get: .....	49

---

Modifying Bean Properties: .....	50
Handling an HTTP POST (Form Submit): .....	52
Error Handling: .....	54
Exception Classes: .....	56
OAException:.....	56
OAttrValException: .....	58
ORowValException:.....	59
Hands – On Exercises: .....	61

## How to use this manual



Video1: Script: Vid1-Introduction to the chapter and its content – Face recording.

---

This video will introduce the material covered in this pdf, the goals,

1. How this document is organized
  2. What is the purpose of this document
  3. What will you achieve after going through the document and related videos
  4. How to read this document
  5. How does it relate to the work you will be doing on real project
  6. Reference to other reading materials for further references
- 

This manual has been organized as a step by step guide to teach how to create reports using Oracle Developer Suite 10G. The target audience is new comes to Oracle Developer suite. It assumes that the reader has basic knowledge of Oracle concepts and PL/SQL. After completing this course, you will be able to create variety of reports using Oracle Developer Suite 10G.

This manual is organized to be read in a serial fashion and follow the instructions given in the document as it is. Practical examples are given in each section to guide you through every step. The tables referred here are common (shared) tables used by different batches, so care should be taken not to delete or update the rows which does not belong to you, this may create problem for the other batches. At the end of the course, you should delete the data you have created.

There are several symbols used to designate particular sections, which are described below:



- Describes the purpose of the section.



- Notes relevant to the section above



- This denotes the task to be completed by the audience on his own PC. The layout of the output has to be followed as it is. For any confusion, the faculty should be contacted.

## Introduction to OA Framework:

OA Framework is a J2EE Framework developed by Oracle for extending Oracle e-Business Suite. OA Framework follows the MVC architecture which is a widely accepted framework for web based applications. In short a Framework is nothing but a partially built application which can be extended to suite the organization's business needs.

When we build a new application we have to take care of at least below things into consideration:

- Authentication System (Login and User Management)
- Security
- Accessibility
- User Personalization and Extensible Layer

OA Framework has all the above features. This allows the developer to concentrate on business logic and let the framework handle the technology.

By using OA Framework a developer can:

- Build new Self Service Pages for Oracle e-Business Suite
- Personalize the Self Service Pages
- Extend the Self Service (seeded) pages

### **Parts of OA Framework:**

The three parts of OA Framework are:

- BC4J
- UIX
- OA Extension

**BC4J** – BC4J stands for Business Components for Java. The business components represent the business logic. The different BC4J objects are Application Module, View Object, Entity Object, Association Object, View Link and Validation View Object, Validation Application Module.

**UIX** – UIX stands for User Interface XML which represents the java components for representing the User Interface. UIX represents the OA page in OA Framework.

**OA Extension** – OA Extension represents the declarative data for UIX extension. It resides in the MDS (Meta Data Repository)

## Setting Up your Development Environment:

To kick start the development activity in OA Framework we need to do some basic and high level setups for ease of development and to avoid complexities in the on – going development processes.

It is highly important that an OA Framework Developer obtains the specific version of Oracle JDeveloper 10g with OA Extension for their version of Oracle e-Business Suite. JDeveloper 10g with OA Extension is used with R12 e-Business Suite instances.

Please refer to the **MetaLink note 416708.1** which contains all the latest information on the released patches. Kindly take the assistance of a Database Administrator on this task.

When we download the proper patch for Oracle e-Business Suite instance the patch will be named similar to the following:

**p5856648\_R12\_GENERIC.zip**

JDeveloper 10g with OA extension does not contain an installation routine. The installation process consists of unzipping the patch to a directory. For example

On Linux - /Jdev

On Windows – D:/Jdev

When we unzip the patch it will create three (3) subdirectories as follows:

- Jdevbin – This subdirectory contains all the executables, utilities and supporting files for JDeveloper 10g with OA extension.
- Jdevdoc – This subdirectory contains the OA framework Developer's guide along with javadoc files for the APIs
- Jdevhome – This subdirectory is empty. But can be used as the base directory for our OA Framework files.

Once the relevant patches have been applied for the initial setups required for JDeveloper with OA Framework, we will have to follow the steps mentioned below:

1. Refer to MetaLink note 416708.1 to obtain the proper patch for your Oracle e-Business Suite instance.
2. Download the proper JDeveloper patch.
3. Install JDeveloper by unzipping the patch provided by the DBA to a proper directory.
4. Configure the environment variables
5. Obtain the database connection file (.dbc) for Oracle e-Business Suite
6. Create a shortcut to the JDeveloper executable
7. Create a Framework Development user and responsibility for OA Framework testing purposes.
8. Un compress Tutorial.zip into the JDEVHOME directory
9. Launch JDeveloper 10g application
10. Configure the connections
11. Test the installation and configuration

Following steps would help us in configuring the environment variables:

- JDEV\_USER\_HOME is a mandatory environment variable that points to the base directory where the development files are stored.
- JDEV\_JAVA\_HOME is an optional environment variable that points to a specific Java SDK.



Setting the JDEV\_USER\_HOME environment variable on Windows:

- Right – Click My Computer on your Desktop, select Properties
- In the System Properties dialog, select the Advanced Tab
- On the Advanced Tab, select the Environment Variables button
- Select the New Button at the User Variables for <username> box
- In the New User Variable dialog, enter JDEV\_USER\_HOME in the variable name field
- Set the Variable Value field to the location of your JDEV\_USER\_HOME subdirectory (for example D:\Jdev\jdevhome\jdev)
- Select OK in each of the dialogs you opened to save the new user environment variable.

Setting the JDEV\_USER\_HOME environment variable in Linux:

- Open a terminal window on your Linux machine and type the commands listed  
set JDEV\_USER\_HOME = /Jdev/jdevhome/jdev  
export JDEV\_USER\_HOME

Locate the .dbc file from \$INST\_TOP/appl/fnd/12.0.0/secure and

Put this file into the JDEV\_USER\_HOME/dbc\_files/secure

Create the shortcut – The executable for JDeveloper is located in Windows –  
D:\JDev\jdevbin\jdev\bin\jdevW.exe

Linux - \JDev\jdevbin\jdev\bin\jdev

Linux does not run the .exe (Windows) executable

Creating a shortcut varies by operating system as follows:

Windows – Create a Desktop shortcut to the JDeveloper executable

Linux – Create an alias or shell script which points to the JDeveloper executable.

Assign the Oracle e-Business Suite User

By default in the examples use the following connection information:

Application User: FWKTESTER

Application Password: FWKDEV

Application Short Name: AK

Responsibility Key Name: FWK\_TBX\_TUTORIAL

Uncompress Tutorial.zip –

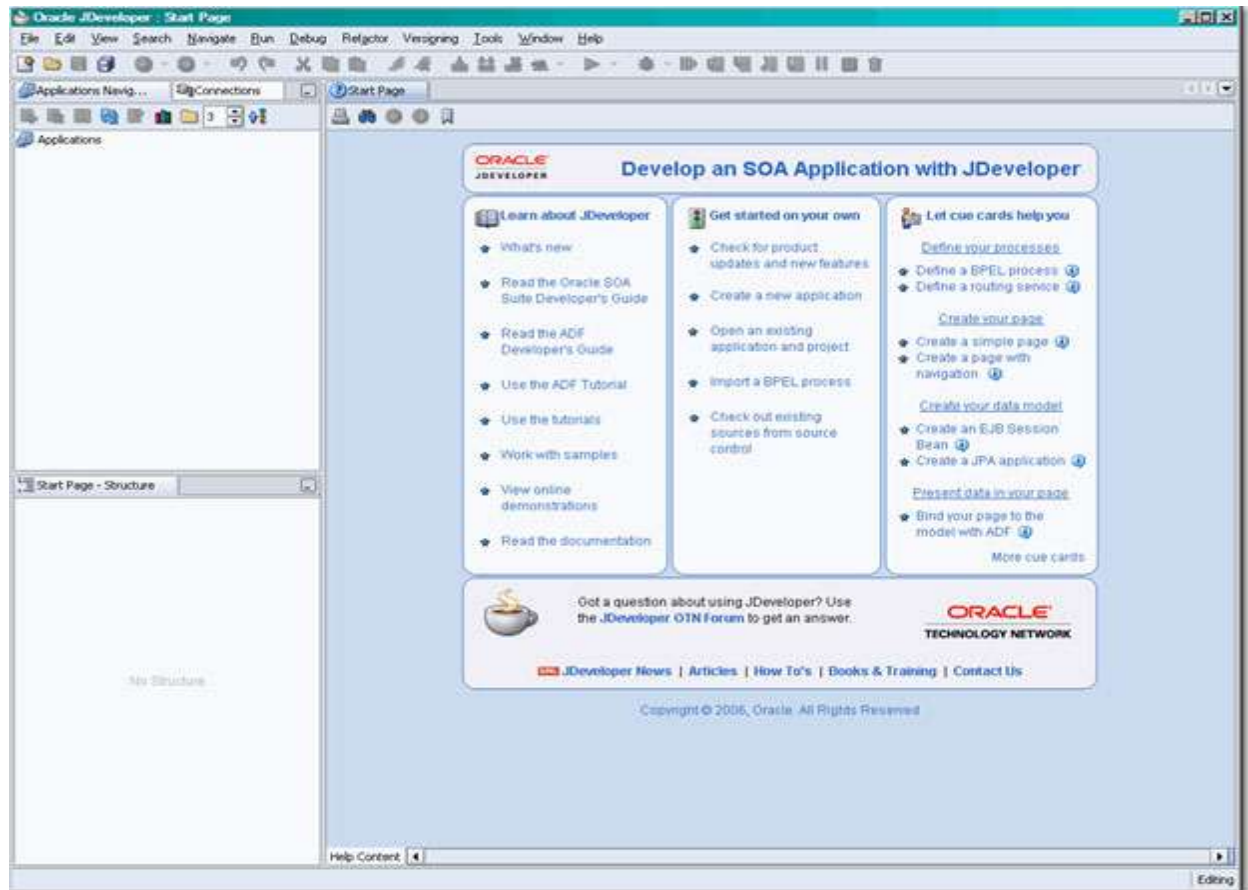
In the JDEVBIN directory there is a file named Tutorial.zip. Un compress this file into JDEVHOME directory.

Because the .zip file contains the. /jdev/... and other directories it will create the proper structure as it is uncompressed.

### Launch JDeveloper 10g –

The method to launch JDeveloper 10g will vary depending on operating system, operating system version and the shell.

The first time JDeveloper 10g is launched we see the Start Page. It is something we should close.



### Configure the connections and test –

The steps to configure the connections and test are as follows:

#### 1. Configure the Database Connection

Connections Tab on Navigator Panel.

Right – Click Database, select New Database Connection

Name the Connection

Username = apps (This is the database user to which we are going to connect. Within Oracle e-Business Suite the master database user that has access to all the Oracle e-Business Suite database object is APPS)

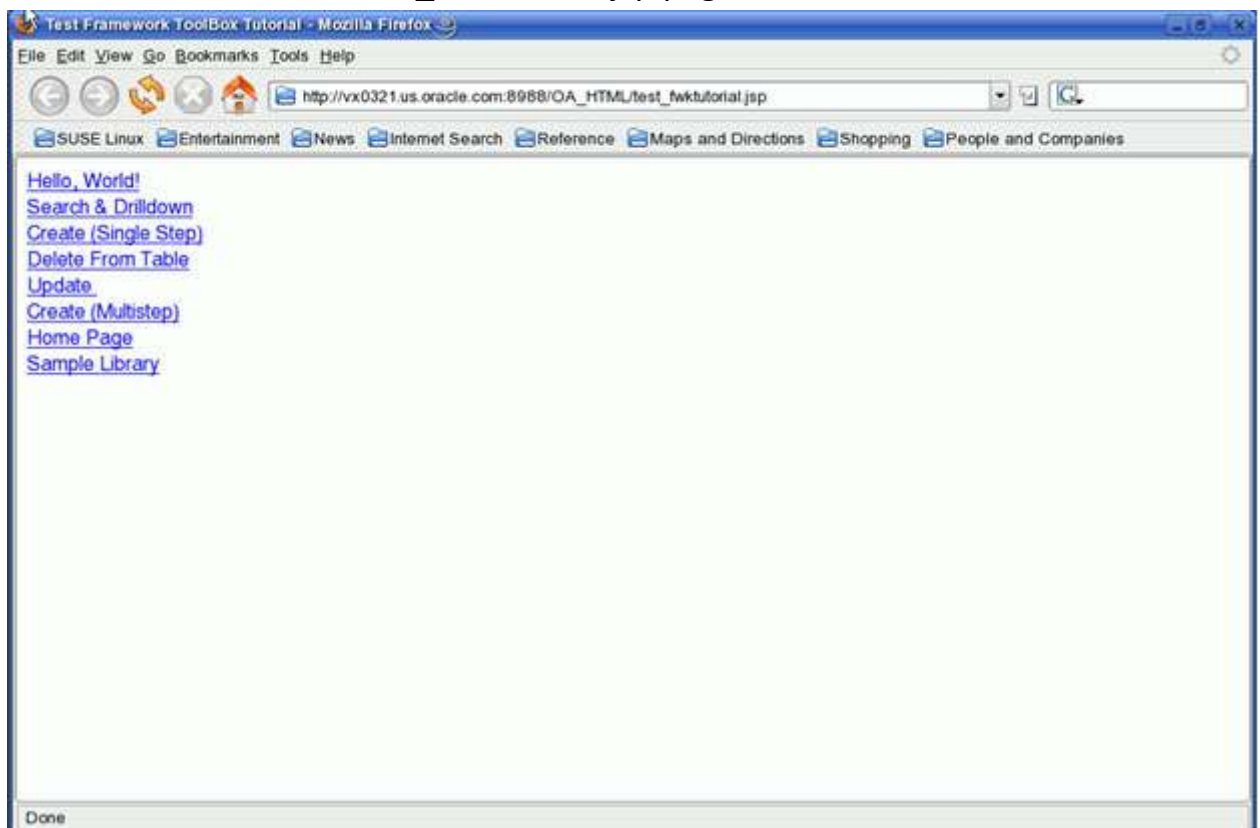
Password = apps (The default password for APPS username)

Set the host name and SID. Refer to .dbc file.

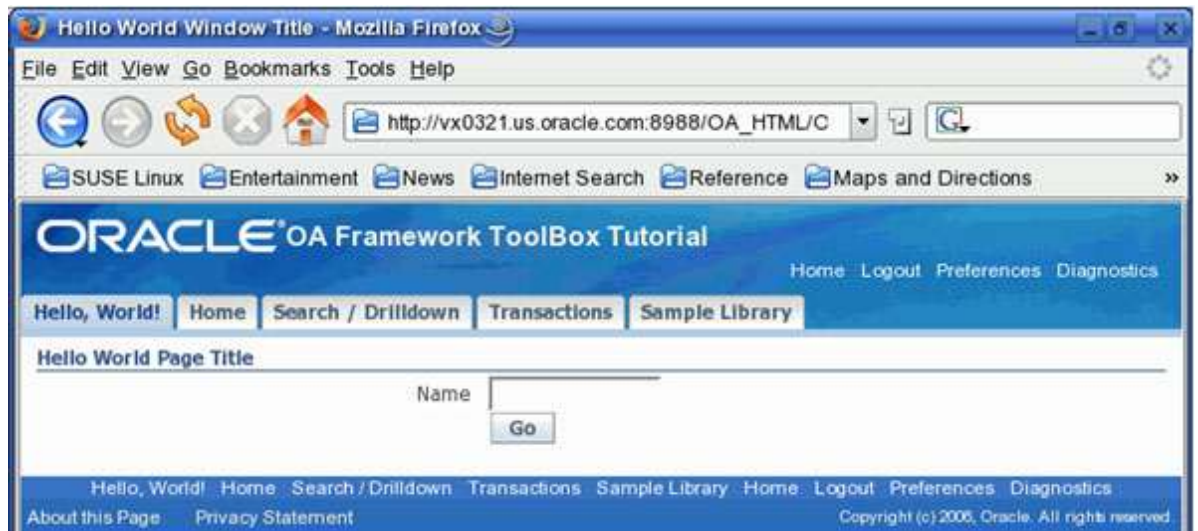
Test the connection

Finish

2. Click the Applications Navigator Tab
3. Choose File > Open > myprojects > toolbox.jws
4. Expand toolbox in the Application Navigator Panel.
5. Double – Click Tutorial to open Project Properties window
6. Expand the Oracle Applications Menu Entry
7. Choose Oracle Applications > Runtime Connection
8. Set the DBC Filename to the directory and name for your DBC file.
9. Click the Save All icon to save your progress
10. Expand Tutorial
11. Expand the Web Content folder
12. Right – Click test\_fwktutorial.jsp and choose Run.
13. You should see the test\_fwktutorial.jsp page shown below.



14. You should see the Hello, World! Page shown below.



We have connected to database or else test\_fwktutorial.jsp would not have worked.

We have connected to Oracle e-Business Suite application instance or else Hello, World! Would not have worked

We have a user on Oracle e-Business Suite instance named FWKTESTER with a password of FWKDEV, assigned to the AK product/module and given the FWK\_TBX\_TUTORIAL responsibility.

Now we are complete with installation and configuration and setting up the OA Framework development environment

## OA Framework Development Runtime Configuration:

Once we are complete with setting up the development environment for OA Framework for starting the development activity next we need to understand how to configure our OA Framework runtime for pages in development.

### Page Test Modes:

While during the development activity there are several test modes that can be leveraged at different points during the development and testing cycle.

Following are the different page test modes:

- **OADeveloperMode** – Performs various coding standard checks  
It should always be enabled during development
- **OADiagnostic** – Enables the Global Diagnostics button at the top of the page. It should always be enabled during development.
- **OABackButtonTestMode** – Test's the page's browser Back button support. This should be used only when performing Back button support tests.
- **OAPassivationTestMode** – Test's the page's support for passivation. This should be used only while performing passivation tests.
- **OADumpUITree** – Writes out the component tree for a page  
This should be used only when we need to diagnose a problem.

Project Settings (For enabling the different test modes):

- Select the project in Oracle JDeveloper 10g system navigator and select Project > Project Settings from the main menu.
- In the Project Settings dialog select the Common > Oracle Applications > Run Options Page.
- In the Run Options page select the test modes that you want to enable in the Off Options list and shuttle them to the On Options list.
- Select OK to save the changes.

Test JSP Cookies (For enabling the different test modes):

The test\_fwktutorial.jsp that we ran as described in Setting up the Development Environment includes the following cookie definitions. We can modify this jsp file appropriately to enable or disable different test modes.

Include the below code inside the jsp file.

```
<SCRIPT LANGUAGE="JavaScript">
document.cookie = "OADiagnostic=1";
document.cookie = "OADEveloperMode=1";
document.cookie = "OABackButtonTestMode=0";
document.cookie = "OAPassivationTestMode=0";
document.cookie = "OADumpUITree=0";
</SCRIPT>
```

Additional to this if we need to know what version of the OAFramework and Java we are running then see the instructions in Discovering Page, Technology Stack and Session Information.

## OA Framework Essentials

### Anatomy of an OA Framework Page:

The Anatomy of an OA Framework Page describes about the components and sub – components included in designing an OA Framework Page in Oracle e-Business Suite. This chapter goes on to explain the following in detail:

- Page Basics
- The Model
- The View
- The Controller
- Web Bean Architecture

### Page Basics:

At the browser level an OA Framework page like any other web page presents the page output in standard HTML. In the middle tier this OA framework page is implemented in memory as a hierarchy of Java Beans something similar to a Java Client User Interface. Each User Interface like buttons, a table, the tabs, and the application branding image that displays in the page corresponds to one or more web beans in the hierarchy.

When the browser issues a request for a new page OA Framework reads the page's metadata definition to create the web bean hierarchy. For each bean component which has an associated User Interface controller OA Framework calls code that we write to initialize the page. When the page processing completes OA framework hands the web bean hierarchy to the UIX framework so it can generate and send HTML to the browser.

When the browser issues a form submit, OA framework recreates the web bean hierarchy and then calls any event handling code that we have written for the page beans. When page processing completes the page HTML is generated again and sent to the browser.



## The Model:

The Model Layer encapsulates the underlying data and business logic of the application. The Model Layer represents the BC4J components which consist of the following:

- **Application Module**  
The Application Module defines the logical data and business methods. The Application Module handles the transaction and interacts with the client. It serves as containers for related BC4J objects. Application Module is responsible for establishing database connections and transaction context. Application Modules can be nested to provide more complex application modules. Every OA Framework page should have at least one application module associated with it.
- **Entity Object**  
Entity Object encapsulates the business rules and logic. Entity object is used when there is a Insert, Update or deletion of data. All data validations across the application are provided by the Entity Object. Entity Object can be linked with each other to create an association object.
- **View Object**  
View Object encapsulates or joins a database query. It iterates over the result set. View object may be based on a single or multiple Entity Objects. View Object acts as a single point of contact for getting and setting entity object values. View objects can be linked to form View links. All view objects should be associated with an Application Object before usage.
- **Association Object**  
Association Object is used to join two or more entity objects together
- **View Link**  
A View Link is an active link between view links. A view link can be created by providing the source and destination views and source and

destination attributes. There are two modes of View Link operation that can be performed. A document and Master/Detail operation.

## The View:

The View Layer in MVC Architecture (Model View Controller) consists of a Page which in turn contains Regions and Items. View is different from View object.

View object is part of BC4J Layer (Model).

The View layer formats and displays data from the model to the User.

Now let us see what a Region is all about.

Regions are containers for different items in a Page.

A Region is one of the most important components in a Page.

## View: OA Framework-Based Page

**ORACLE** Toolbox

Home Logout Preferences Diagnostics

Hello, World! Home Search / Drilldown Transactions Sample Library

### Purchase Orders

This is the instruction text that applies to the entire page.

#### Search

This is the instruction text that applies to this region.

Purchase Order

Created

☐ Show my orders only

This is a field-level hint (also known as a "Short Tip").

This is the instruction text that applies to this region.

✓TIP This is an OATipBean.

Number	Description	Employee Name	Created	Supplier	Currency	Order Total	Details
No search conducted.							

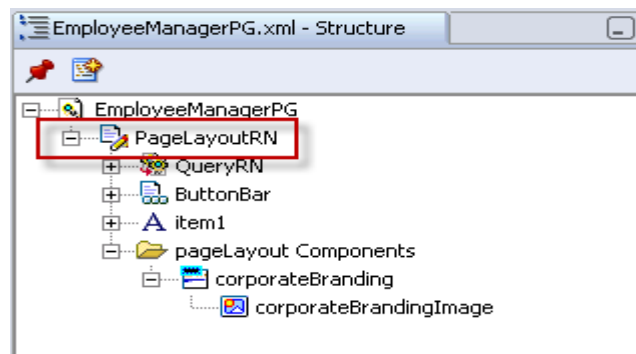
## Types of Regions in a Page:

In a Page there can be multiple regions inside a region and multiple items inside that region. Every Region is a Java Bean which acts as a container for regions under it. A Region inside another Region is called Child Region and that which is present in the same level as another region is called as Sibling. Region properties can be set like Style, ID, Rendered etc.

The commonly used Region Styles are:

### 1. pageLayout Region

pageLayout Region is the highest level layout. By default this is the top most region for every page. Any no. of regions can be created under the pageLayout region. The java bean associated with this pageLayout region is OAPageLayoutBean.

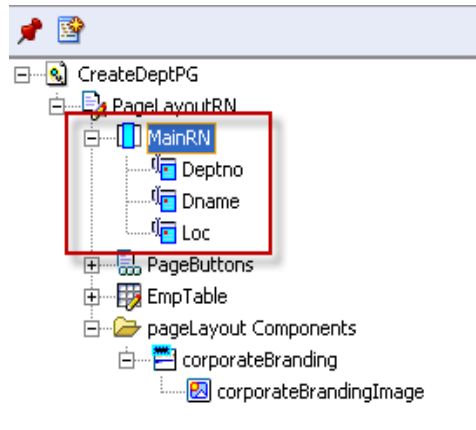


### 2. messageComponentLayout Region

This Region is a very common region style used which contains all the items starting with message. For example messageTextInput, messageStyledText etc. This region can have only message style of items.

Non – Message items has to be placed inside messageLayout under the messageComponentLayout region. The items can be displayed in multiple columns and rows. The Java Bean associated with this messageComponentLayout region

is `OAMessageComponentLayoutBean`.



\* Indicates required field

\* Name

\* Code

(Example: APPLICATION\_LOOKUP)

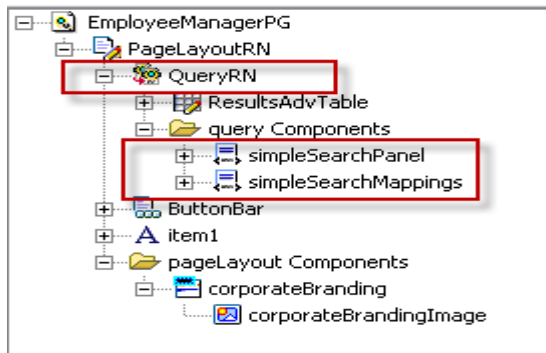
Description

\* Application Name

Access

### 3. queryBean Region

queryBean as the name suggests is used when we have to perform any search on the page. It has different panels like simple search panel, advanced search panel and Views panel. We can enable or disable these panels using the property inspector. The different construction modes available with this region are Results based search and Auto Customization criteria and none. The Java Bean associated with the queryBean region is `OAQueryBean`.



Name	Application
No search conducted.	

#### 4. hideShow Region


In a page if we want to hide and show the items or regions we can set the top region's style to hideShow. By doing this we can hide or show a part of the information or the entire section. The Java Bean associated with this hideShow region is `OADefaultHideShowBean`.


Note that the search is case insensitive


Project


To qualify for the above search, the project must have been created or updated before:

☐ Hide More Search Options

Organization  

Project Manager  

Customer  

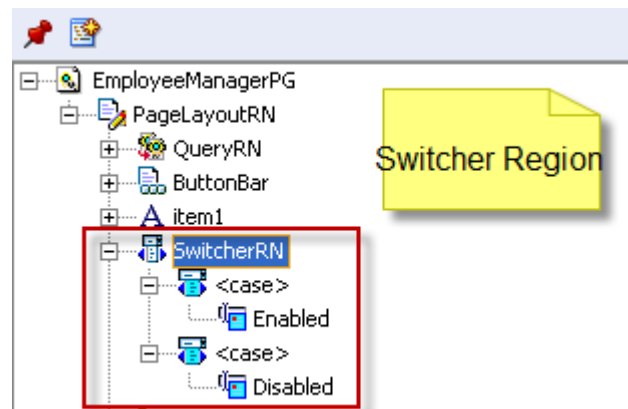
Project Set  

Project Status

hideShow Region

## 5. switcher Region

The switcher region is used at runtime to decide which item needs to be rendered. It is always bound to an attribute in a View Object. A view instance and a view attribute are specified with switcher. The Java Bean associated with this switcher region is OASwitcherBean.



## 6. defaultSingleColumn Region

The defaultSingleColumn as the name suggest is used for holding the items in a single column. This region is not to be used in a page. Whenever there is a need for showing a single column region use the messageComponentLayout region. Sometimes we may have to select the defaultSingleColumn region to use the Region using wizard functionality but change the region style to messageComponentLayout immediately after using the wizard. The Java Bean associated with defaultSingleColumn region is OADefaultSingleColumnBean.

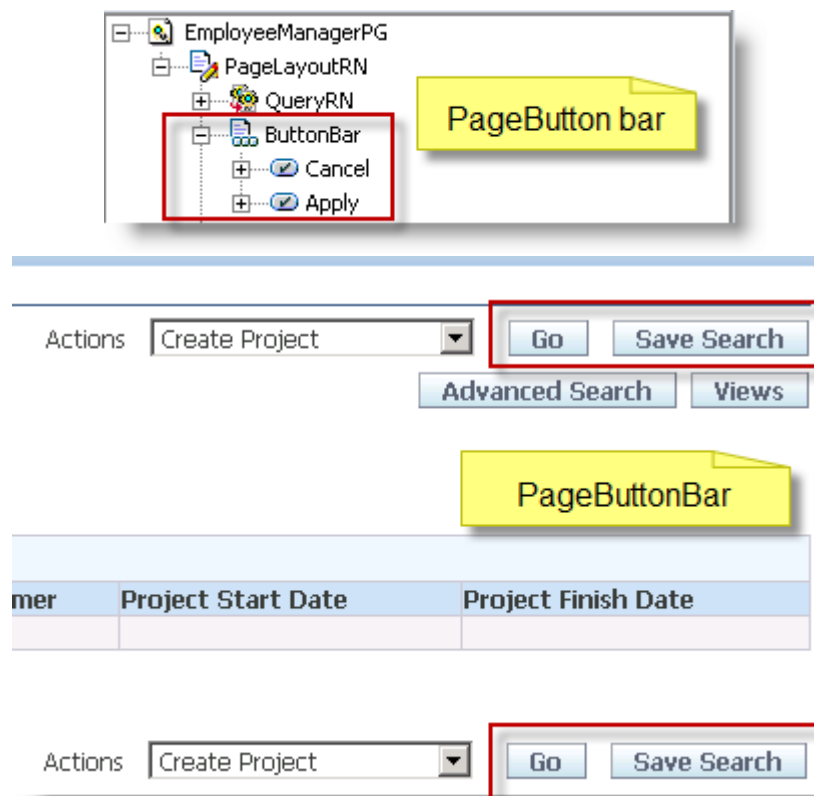
## 7. defaultDoubleColumn Region

The defaultDoubleColumn region is used for holding the items in a double column. This is similar to defaultSingleColumn region and should not be used. The Java Bean associated with defaultDoubleColumn region is OADefaultDoubleColumnBean.

## 8. pageButtonBar Region

Whenever there is a need of any item to be created at a page layout level we use the pageButtonBar region. It is the child of pageLayout level. The items created under this region will be displayed at the bottom (below footer) and top of the page (below page title).

The Java Bean associated with pageButtonBar region is OAPageButtonBarBean.



## 9. tableLayout Region

The tableLayout region is a wrapper containing rowLayout and cellFormat regions. This can be easily mapped to the HTML table. tableLayout region is used when we need more control on placing items in page. The Java Bean associated with tableLayout region is OATableLayoutBean.



## 10.rowLayout Region

The rowLayout region is used when we want to hold a cell format inside as a child to it. It can be an independent region or a child of a tableLayout or advancedTable. The Java Bean associated with rowLayout region is OARowLayoutBean.

## 11.cellFormat Region

cellFormat region is a container of regions or items. The Java Bean associated with cellFormat region is OACellFormatBean.

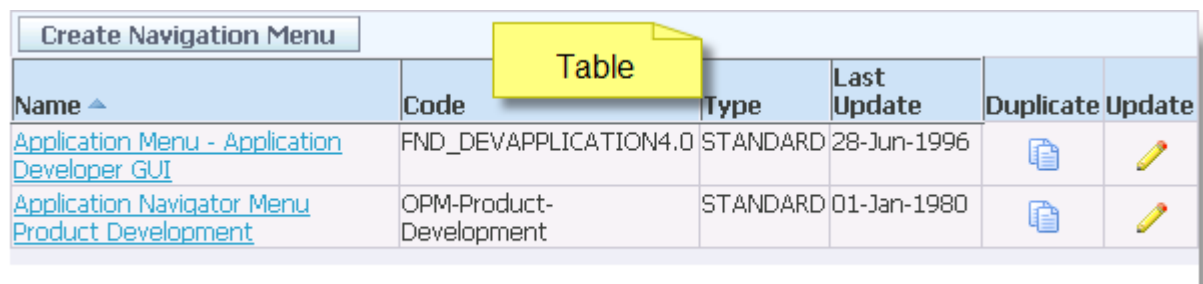
## 12.advancedTable Region





advancedTable region is used when there is a need to perform create, update and delete functionality on the same page without navigating to another page. This will provide an Oracle Forms functionality to the OA page. The Java Bean associated with advancedTable region is OAAAdvancedTableBean.

Select	Prompt	Sub Menu	Sub Menu Type
<input type="checkbox"/>	Register		
<input type="checkbox"/>	Form		
<input type="checkbox"/>	Function		
<input type="checkbox"/>	Menu		
<input type="checkbox"/>	Messages		
<input type="checkbox"/>	Database	Database Menu - Applicat	STANDARD
<input type="checkbox"/>	Lookups	Lookups Menu - Applicati	STANDARD
<input type="checkbox"/>	Validation	Validation Menu - Applicat	STANDARD
<input type="checkbox"/>	Chola Ftp Page		
<input type="checkbox"/>	Manager Page		

### 13.table Region

As the name suggests table region is used for displaying the data in a tabular format. The Java Bean associated with table Region is OATableBean.











Name ▲	Code	Type	Last Update	Duplicate	Update
<a href="#">Application Menu - Application Developer GUI</a>	FND_DEVAPPLICATION4.0	STANDARD	28-Jun-1996		
<a href="#">Application Navigator Menu Product Development</a>	OPM-Product-Development	STANDARD	01-Jan-1980		

### 14.flowLayout Region

When we want to display table actions for a table or advancedTable we can create a flowLayout. Mainly it is for buttons we use this layout. The Java Bean associated with flowLayout region is OAFlowLayoutBean.

### 15.hGrid Region

When the data needs to be displayed in a hierarchical structure the hGrid region is used. Detailed information for each row can be given. The Java Bean associated with hGrid region is OAHGridBean.

Select Focus		Personalization Repository	Path
<input type="checkbox"/>		<input type="checkbox"/> oracle	/oracle
<input type="checkbox"/>		<input type="checkbox"/> apps	/oracle/apps
<input type="checkbox"/>		<input type="checkbox"/> ams	/oracle/apps/ams
<input type="checkbox"/>		<input type="checkbox"/> campaign	/oracle/apps/ams/campaign
<input type="checkbox"/>		<input type="checkbox"/> ap	/oracle/apps/ap
<input type="checkbox"/>		<input type="checkbox"/> customizations	/oracle/apps/ap/customizations
<input type="checkbox"/>		<input type="checkbox"/> invoice	/oracle/apps/ap/invoice
<input type="checkbox"/>		<input type="checkbox"/> oie	/oracle/apps/ap/oie
<input type="checkbox"/>		<input type="checkbox"/> ar	/oracle/apps/ar

HGrid

## 16.train Region

When we have multiple pages then we use the train region.

A highlighted mark shows the current page which we are in. The Java Bean associated with train region is OATrainBean.

The screenshot displays the Oracle Expenses application interface. At the top, there is a navigation bar with tabs: Expenses Home, Expense Reports, Credit Card Transactions, Accruals, and Projects and Expenses. A yellow sticky note labeled 'Train' is placed over the 'Expense Reports' tab. Below the navigation bar is a horizontal breadcrumb trail with six items: General Information (highlighted with an orange circle), Credit Card Transactions, Credit Card Expenses, Cash and Other Expenses, Expense Allocations, and Review. Below the breadcrumb trail is an 'Information' section with a message: 'You have 0 corporate credit card transactions in the selected reimbursement currency.' Below this is the 'Create Expense Report: General Information' form. The form includes a 'Name' field with the value 'Marlin, Ms. Amy (34)', a 'Services Department' field with the value '422' and a magnifying glass icon, and a 'Reimbursement Currency' dropdown menu with the value 'USD - US dollar'. The form also includes 'Save' and 'Cancel' buttons, a 'Step 1 of 6' indicator, and a 'Next' button. The 'Train' region is highlighted with a red border.

So far we have seen different regions we can use while designing the OA Framework page.

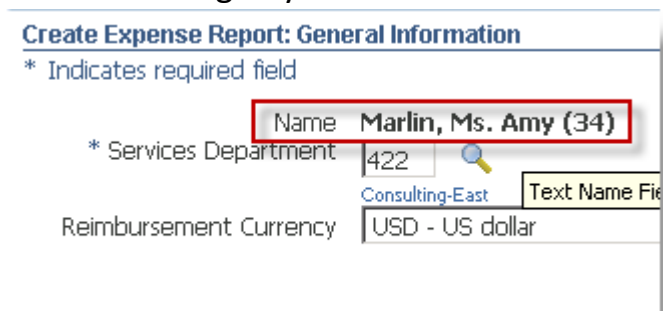
Now let us see what are the Items we can use while designing a OA Framework page.

## Types of Items in a Page:

There are a variety of items that can be created in a page. For example if a user wants to enter some data in the page then we make use of the messageTextInput item type.

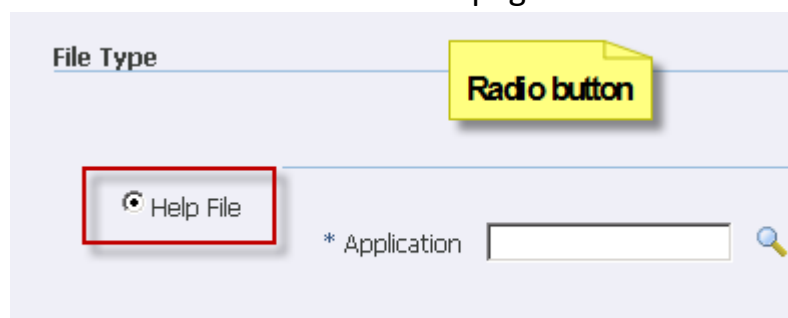
- messageStyledText

When we want to display only the text and the user should not be able to edit the text then we use the messageStyledText.



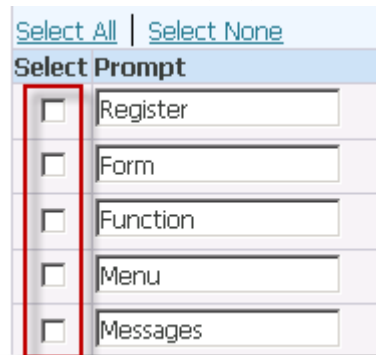
- messageRadioGroup

This helps us to create radio buttons in our page.



- messageCheckBox

This helps us to create check box in our page.

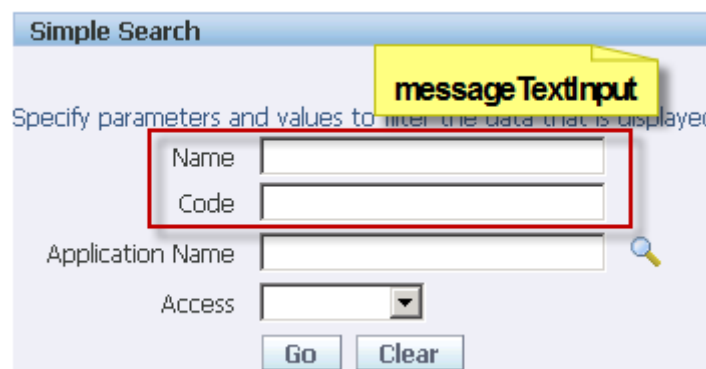


A screenshot of a 'Select Prompt' dialog box. At the top, there are two links: 'Select All' and 'Select None'. Below them is a table with a 'Select' column containing checkboxes and a 'Prompt' column containing text. The items are: Register, Form, Function, Menu, and Messages. A red rectangle highlights the 'Select' column.

Select	Prompt
<input type="checkbox"/>	Register
<input type="checkbox"/>	Form
<input type="checkbox"/>	Function
<input type="checkbox"/>	Menu
<input type="checkbox"/>	Messages

- messageTextInput

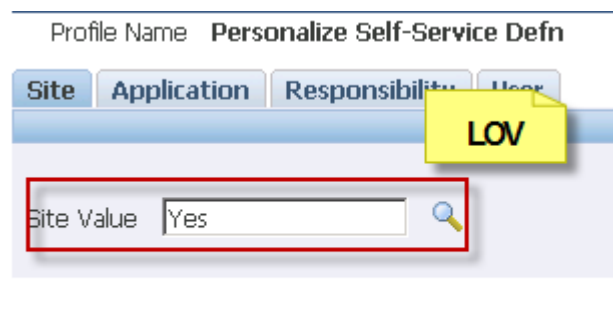
This helps the users to enter data into this box.



A screenshot of a 'Simple Search' form. It contains several input fields: 'Name', 'Code', 'Application Name', and 'Access'. A red rectangle highlights the 'Name' and 'Code' fields. A yellow callout box with the text 'messageTextInput' points to the 'Name' field. Below the input fields are 'Go' and 'Clear' buttons.

- messageLovInput

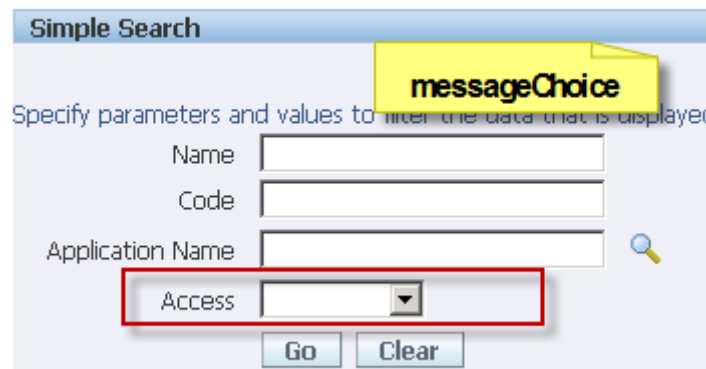
When we want the users to select from a list of values and quickly select the values then we use the messageLovInput. This will open a pop – up and we can search for values and select from the results.



A screenshot of a 'Personalize Self-Service Defn' form. It has tabs for 'Site', 'Application', 'Responsibility', and 'User'. The 'Site' tab is selected. Below the tabs is a 'Site Value' input field with the text 'Yes'. A red rectangle highlights the 'Site Value' field. A yellow callout box with the text 'LOV' points to the 'Site Value' field.

- messageChoice

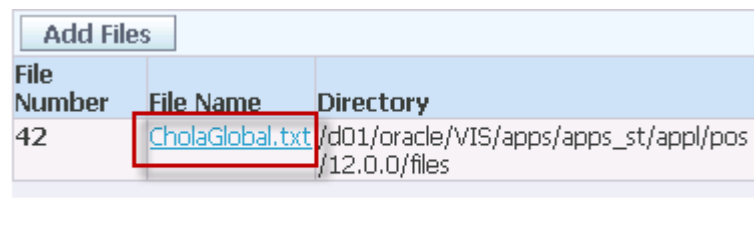
When we want to display a list of values in a drop down fashion then we use the messageChoice.



The image shows a 'Simple Search' form. It has a title bar 'Simple Search' and a subtitle 'Specify parameters and values to filter the data that is displayed'. There are three text input fields: 'Name', 'Code', and 'Application Name'. Below them is a 'messageChoice' dropdown menu, which is highlighted with a yellow callout box and a red rectangle. The dropdown is currently empty. To the right of the dropdown is a magnifying glass icon. At the bottom are 'Go' and 'Clear' buttons.

- messageDownload

This is used when the user needs to download a file or an attachment

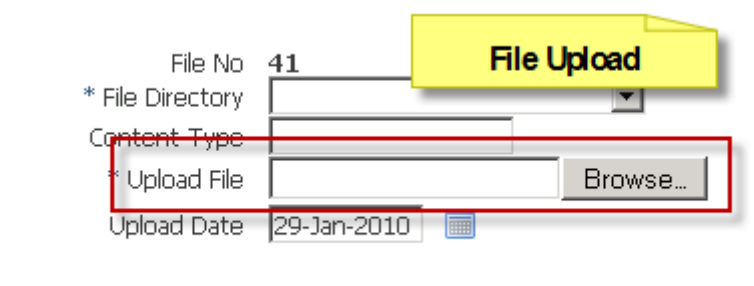


The image shows a table with the title 'Add Files'. The table has three columns: 'File Number', 'File Name', and 'Directory'. The first row has the values '42', 'CholaGlobal.txt', and '/d01/oracle/VIS/apps/apps\_st/appl/pos/12.0.0/files'. The 'File Name' cell is highlighted with a red rectangle.

File Number	File Name	Directory
42	CholaGlobal.txt	/d01/oracle/VIS/apps/apps_st/appl/pos/12.0.0/files

- messageFileUpload

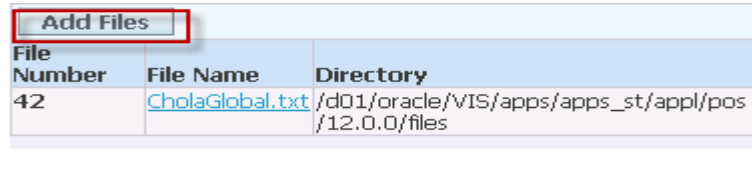
This is used for displaying a button with a messageTextInput. User can browse from the local machine and upload a file using this.



The image shows a 'File Upload' form. It has a title bar 'File Upload'. There are several fields: 'File No' with value '41', '\* File Directory' with a dropdown, 'Content Type' with a text input, '\* Upload File' with a text input and a 'Browse...' button, and 'Upload Date' with a date picker showing '29-Jan-2010'. The '\* Upload File' field and the 'Browse...' button are highlighted with a red rectangle.

- button

It is a general button which has an action associated with it. We can set the fire action property which will in turn call a method to perform that action

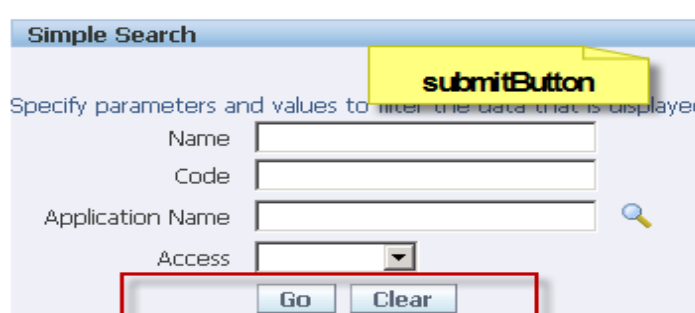


The image shows a screenshot of a web application. At the top, there is a button labeled 'Add Files' which is highlighted with a red rectangular box. Below the button is a table with three columns: 'File Number', 'File Name', and 'Directory'. The table contains one row of data.

File Number	File Name	Directory
42	<a href="#">CholaGlobal.txt</a>	/d01/oracle/VIS/apps/apps_st/appl/pos/12.0.0/files

- submitbutton

This is the button used when we want to submit data to a OAF page.



The image shows a screenshot of a web application titled 'Simple Search'. It contains a form with the following fields: 'Name', 'Code', 'Application Name', and 'Access'. Below these fields are two buttons: 'Go' and 'Clear', which are highlighted with a red rectangular box. A yellow callout box with the text 'submitButton' points to the 'Go' button. The form also includes a search icon (magnifying glass) and a description: 'Specify parameters and values to filter the data that is displayed'.

- Link

- formValue

This is used to send value to a OAF page but without being displayed in the page.

- spacer

This is used when we want to add space between our items placed in the page.

- separator

This creates a horizontal line in an OAF page.

- flex

This is a Flexfield tool. There are two types Key and Descriptive Flexfield.

- urlInclude

This helps us to include HTML content

- tip

Helps to give hint to the user.



## The Controller:

Controller handles the user actions. It responds to user actions and directs application flow.

OA Framework Controller has three methods. Below are the lists of methods:

- processRequest – Fires when a page is rendered
- processFormData – Fires when page submit happens
- processFormRequest – Fires when Fire Action/Fire Partial Action and Page Submit happens.

For those of you who worked in Oracle Forms we can relate these methods like triggers in Oracle Forms.

WHEN-NEW-FORM-INSTANCE can be mapped to processRequest and WHEN-BUTTON-PRESSED can be mapped to processFormRequest.

### processRequest:

Whenever we need some logic to get executed when a page is rendered write that set of code in the processRequest. For example we programmatically need to set the Page Title to “Welcome!!”

This property needs to be set whenever we navigate to this page. So we write this set of code in the processRequest of the controller.

### processFormData:

We don't write any code in this method because it is used by the framework to bind the data from the page to BC4J component (view object)

### processFormRequest:

Whenever there is a post request in the page the method that is called is the processFormRequest. So all the code for any form action is written in this method. For example after entering all the values and we want to submit the data then the logic for submitting the data is written in the processFormRequest of the controller by capturing the form event or action.

The whole logic of the application should not be written in the controller. The controller should delegate it to application module by invoking methods in the application module to perform that logic.

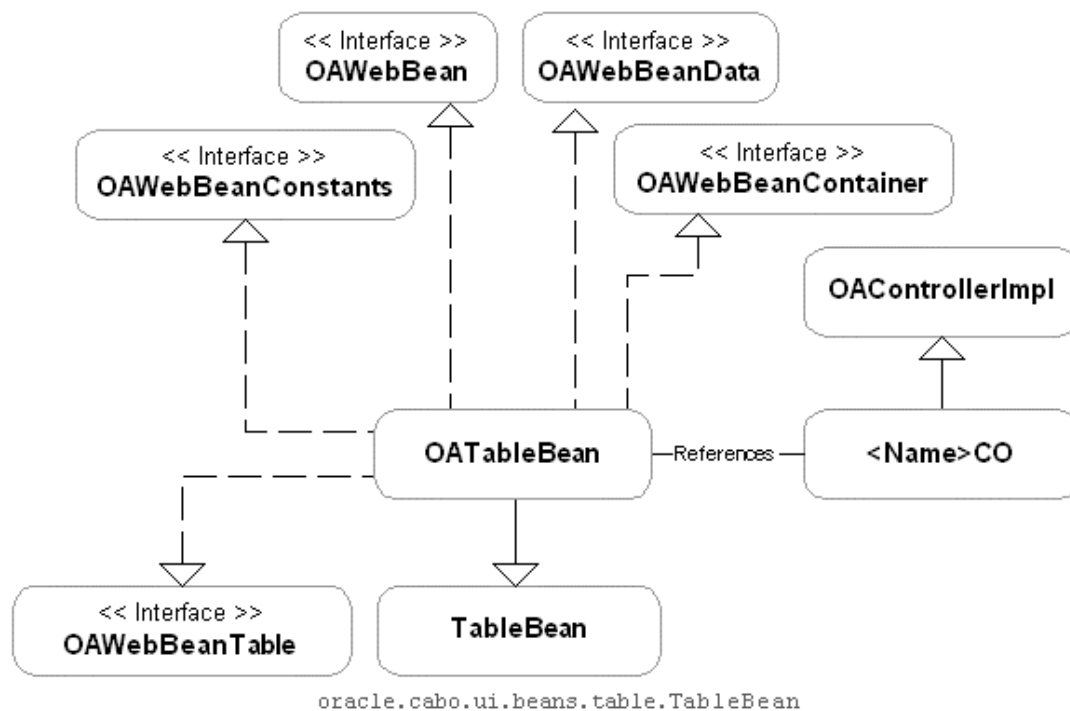
## Web Bean Architecture:

All OA Framework web beans subclass the corresponding beans in UIX framework. For example `OATableBean` extends

an `oracle.cabo.ui.beans.table.TableBean` (cabo was an earlier name for the UIX framework and the package definitions use the same old name)

Each OA framework web bean implements a group of interfaces whose implementations collectively define the behaviors that the OA Framework adds to the base of UIX beans.

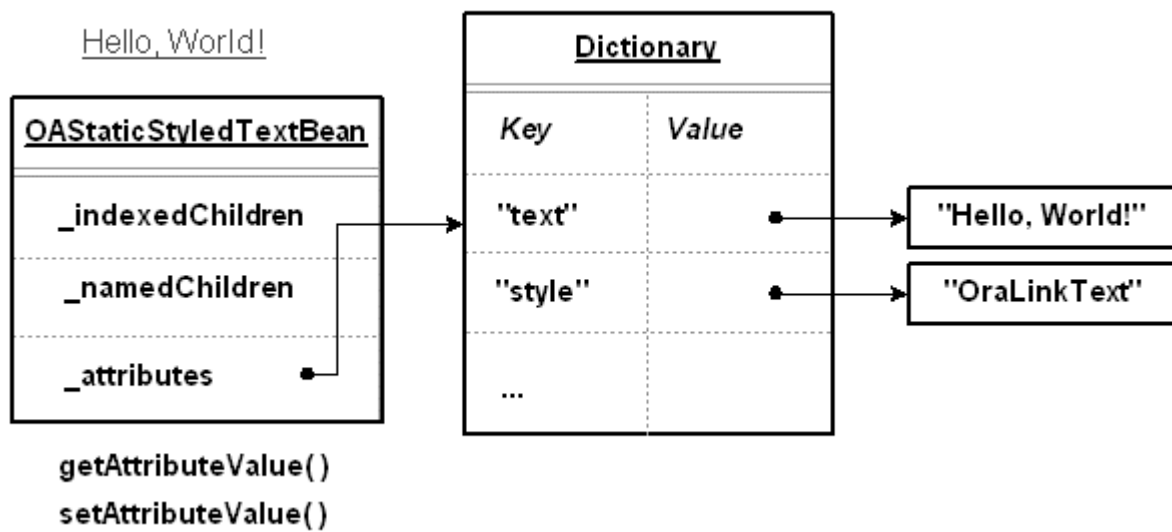
- `oracle.apps.fnd.framework.webui.beans.OAWebBean` – This defines the core behavior common to all web beans (for example this defines the `processRequest`, `processFormData` and `processFormRequest` methods that individual beans implement)
- `oracle.apps.fnd.framework.webui.beans.OAWebBeanData` – This defines common personalization definition and data source management behavior.
- `oracle.apps.fnd.framework.webui.beans.OAWebBeanConstants` – This is a collection of constants used in the view/controller modules.
- `oracle.apps.fnd.framework.webui.beans.OAWebBeanContainer` – This defines the characteristics of all web beans that can act as containers for other web beans. All the layout web beans discussed in the page regions implement this interface.
- `OAWebBean<Type>` - This defines a bean's inherent behaviors within the context of OA Framework. For example the `OATableBean` implements `oracle.apps.fnd.framework.webui.beans.OAWebBeanTable` interface.



## Internal Bean Structure:

Each web bean maintains the following information about itself:

1. `_indexedChildren` – child web beans.
2. `_namedChildren` – child web beans that UIX tags for special behavior.
3. `_attributes` – web bean characteristics



## State Persistence Model ('Passivation'):

Passivation is the process of saving application state to a secondary medium (the database) at specific points so it can be restored (activated) when needed.

OA Framework provides following state management features:

- Scalable Applications

When the resource consumption is high, rather than creating a new dedicated resource instance for each new server thread, OA framework saves application state for idle threads and reclaims their resources for use by others. When the idle user thread wakes up, the saved application state is restored.

- Session Time – Out Recovery

Servlet sessions can time out without forcing the user to restart an incomplete transaction.

The BC4J transaction undo feature is implemented with passivation.

### Enabling Passivation:

To enable passivation for our application we must complete the following steps in sequence:

- Set the FND: Passivation Level system profile option to Resource Threshold or Request.
- Set the FND: Session Timeout Recovery Enabled profile option to Yes at the Application Level in Profile Values window. Each product/module team is responsible for explicitly setting this profile option as part of the passivation certification process.
- Set each root application module's Retention Level property to `MANAGE_STATE`
- Ensure that every application page observes the OA Framework statement management coding standards.
- Certify the pages for passivation by performing the tests outlined in Testing OA Framework Applications: Passivation Test Mode.

### Passivation Profile Options:

- **FND: Passivation Level**

FND: Passivation Level indicates whether passivation is enabled and if enabled at what level of frequency. Valid Values include:

1. None – No passivation support

2. Resource Threshold – Transaction state is passivated just before the server resource owned by the user gets recycled and reused by another thread under high system load. In other words when the resource threshold as specified by the FND: Application Module Pool Recycle Threshold is reached. The user state is reconstituted from the saved state when the user requests the server resource again.

This option also passivates the user transaction state upon servlet session time-out which lets users continue transactions even after a time – out.

This option does not provide any state persistence support when the user request is redirected to another Java Virtual Machine.

3. Request – Transaction state is passivated for every browser request. This option enables persistence when the user request is redirected to another java virtual machine. This even passivates the user transaction state upon servlet session time – out which let users continue transactions even after a time out

- **FND: Session Timeout Recovery Enabled**

This profile option applies only if the FND: Passivation Level profile option is set to Resource Threshold or Request. It indicates whether servlet session time – out recovery is enabled for application. Valid values include:

Yes – When a browser request is issued after a servlet session time – out OA framework restores saved application state in a new servlet session so that the user can continue working uninterruptedly.

No – When a browser request is issued after a servlet session time – out, OA framework displays a standard state loss error page.

As discussed above, the user transaction state persists for the duration of Oracle Applications user session when the

FND: Passivation Level profile option is Resource Threshold or Request.

The Oracle Applications user session length is determined by the following profile options:

ICX: Limit Time – Maximum Oracle Applications user session length (default value is 4 hours)

ICX: Session Timeout – Maximum idle time for an Oracle Applications user session (specified in minutes).

ICX: Session Timeout value should be longer than the servlet session time – out value. This allows users to resume suspended transactions without being redirected to the login page.

## Application Module Retention Level:

Each OA Framework page is associated with a root application module and each application module instance has a new OA Framework property application module retention level. This property indicates how the application module should be managed between requests until it is released.

The Retention Level can be set to the following valid values. If the system profile FND: Passivation Level is set to None, then the retention behavior defaults to RESERVE\_FULL.

In RESERVE\_FULL the root application module and its connection are reserved for exclusive use by the current user thread between requests.

The Retention Level may go to MANAGE\_STATE also.

In MANAGE\_STATE the root application module and its connection may be released for use by another thread between requests, but the application module state is guaranteed to be preserved.

The Retention Level may go to CONNECTION\_AGNOSTIC also.

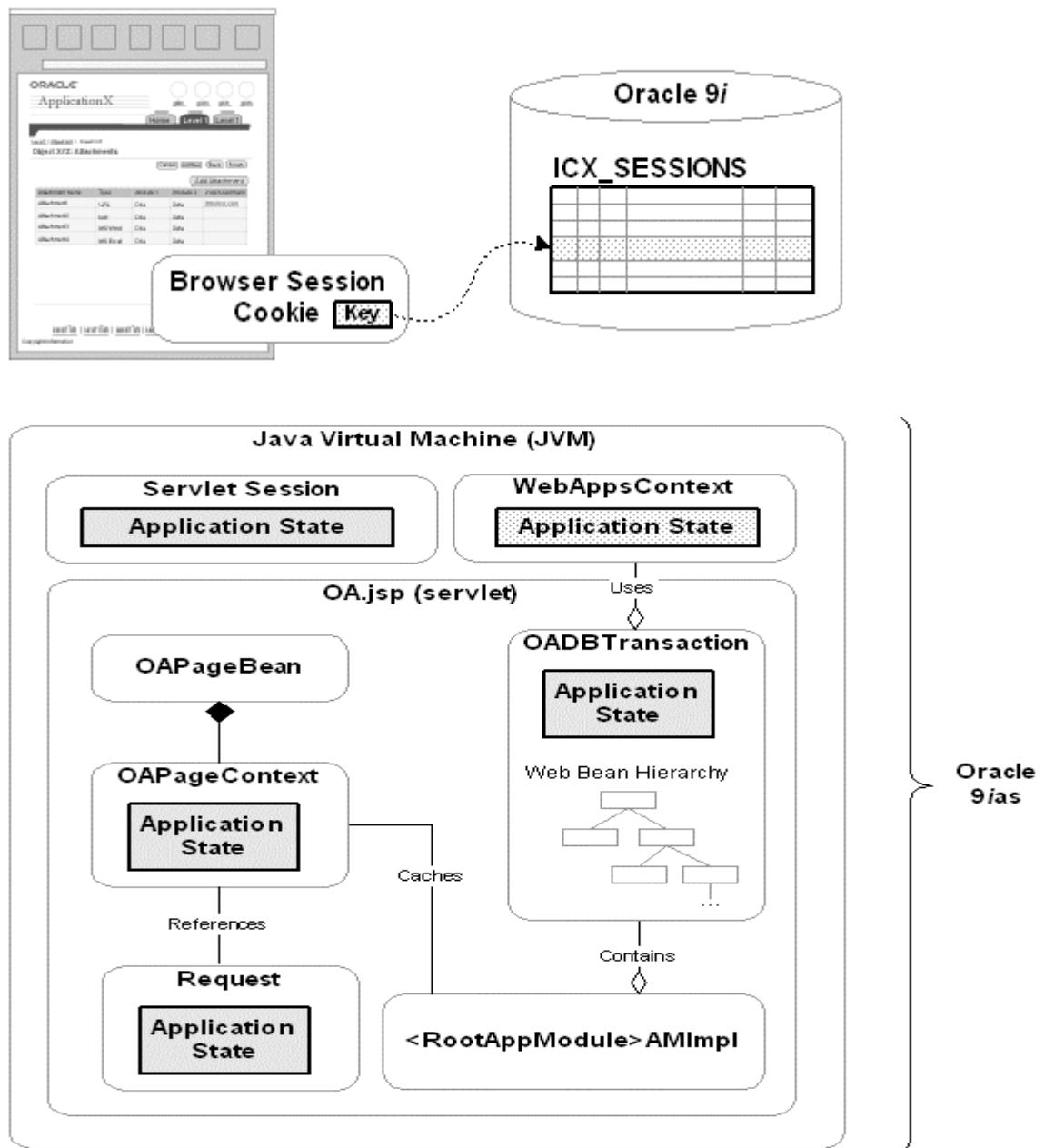
In CONNECTION\_AGNOSTIC the root application module's connection may be released for use by another thread between requests.

Do not set the root application module's retention level to MANAGE\_STATE unless we are ready to implement and certify our page for passivation support.

When we set an application module's Retention Level to `MANAGE_STATE` the application module and its contents becomes passivated, if passivation is enabled in the system.

### What Data is Passivated?:

Let us assume that Passivation is enabled let us see what data is – and is not – passivated.





OA Framework passivates any objects that is stored on the servlet session by calling the `OAPageContext.putSessionValue ()` method. Values stored by calling `OAPageContext.putTransientSessionValue ()` are not passivated.

For Entity Objects and View Objects i.e BC4J passivates view object rows with pending changes. This includes new rows inserted by calling `insertRow ()` on the view object

On the submitted Form Input Field values the request will persist until the user changes the input field value, or until the application module is released.

OA Framework does not passivate states that we save on using the `oracle.apps.fnd.common.WebAppsContext` object.

BC4J provides three levels of passivation control for view objects:

- Enable/Disable passivation for the view object as a whole by setting the Enable Passivation property.
- Force passivation on all transient attributes by setting the Enable Passivation for all Transient Attributes property for the view object.
- Selectively passivate individual transient attributes.

To set the Enable Passivation property open the view object wizard and navigate to the Tuning page to select/deselect the Enable Passivation checkbox.

Following lists the passivation database tables:

- FND\_PS\_TXN

This table stores snapshot of pending changes made to the BC4J application module instances.

- FND\_SESSION\_VALUES

This table stores the servlet session values for a specific user session.

The Delete Data from Temporary Tables concurrent program deletes the passivation records written to the tables mentioned above. These passivation records are deleted together with the ICX session and transaction records.

## Building an OA Framework Applications (Basics):

### Implementing the Model:

This chapter goes on to explain us on how to implement our model objects.

To understand more on implementing the Model we need to understand how to design model objects, understand business components packages.

It is assumed here that we have understood on how to build a

‘Hello, World!’ along with understanding the anatomy of a OA Framework page and understand more on passivation.

### Designing Model Objects:

Within the Model – View – Controller (MVC) architecture, OA Framework draws a clear distinction between “client” and “server” classes, that distinction which may seem to conflict with JSP application architecture.

A typical JSP application has 3 physical tiers:

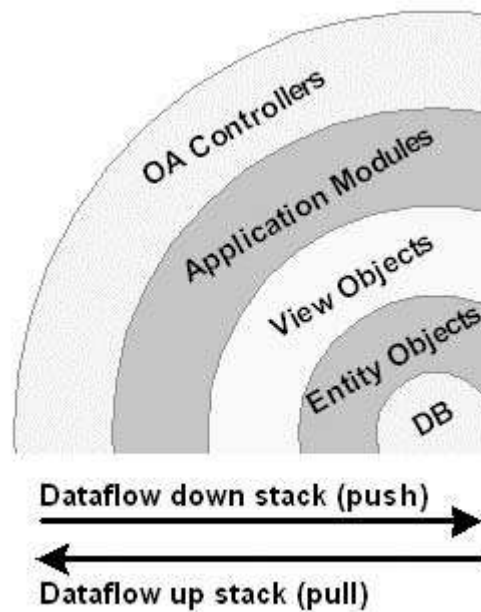
1. The Browser (client where our HTML displays and users interact with the user interface)
2. The Web Application Server (middle tier where our User interface web bean hierarchy is constructed and the business logic executes)
3. The Database Server

Within the middle tier, OA framework draws a further distinction between client and server classes as follows:

1. Client Classes (View and Controller code) drive the HTML user interface
2. Server Classes (Model code) supports any client user interfaces.

This distinction is important because it preserves the ability to use server code with different clients.

The below image illustrates the relationships between a page’s User Interface controllers, application modules, user interface specific view objects and the underlying entity objects.



1. Model code should never reference controller code directly.  
For example, view objects and application modules should not call methods in user interface controllers and entity objects should not reference user interface application modules and view objects.
2. Never reference/import any server – side implementation classes or interfaces on the client side. For example, do not call methods on an `Oracle.apps.fnd.framework.OADBTransaction` in our controller.
3. If we need the server code to do some work for us, always redirect the calls through the root application module using generic 'remotable' `invokeMethod()` method on the `oracle.apps.fnd.framework.OAApplicationModule` interface, or create an interface for the application modules so that we can call typed methods.
4. Never include JDBC or other server – side processing directly in the client code. If the user interface client needs information from the server it should ask the application module, which then delegates or implements the request accordingly.

## Business Components Packages:

All BC4J model components must belong to a Business Components (BC4J) package. The below section assumes that we have set up our development environment, created an OA Workspace, and defined a working datasource.

The following steps would help us in defining our own package for a business component (BC4J):

1. In the JDeveloper navigator, select the OA project where we want to create our package.
2. From the main menu, choose File > New to open the New Object gallery.
3. In the categories tree, expand the Business Tier node, and select Business Component (BC4J).
4. In the Items list, select Business Component Package to open the Business Components Package wizard. We can also right – click on the OA project and select new Business Components Package to navigate directly to the Business Components Package wizard.
5. In Step 1 of 3, enter a Package Name that complies with the OA Framework File/Directory/Package structure standards. Also select Entity objects mapped to database schema objects in the business entity modeling section. Select the next button.
6. In Step 2 of 3, verify our Connection name (it should point to the database where we want to work; JDeveloper uses this information in the entire BC4J components wizard). Set the SQL type and the type map to 'Oracle'
7. Select the Finish button to save your changes.

If we have changed our development environment we need to change the database connection which would already be associated with an existing BC4J package we need to follow the steps:

1. Select the OA project with the business components package we want to edit
2. Right – Click and Select Edit Business Components Project.
3. In the Business Components project wizard select Connection
4. Specify the new database
5. Select OK to save changes

## Designing the User Interface:

All OA Framework applications must be designed in accordance with the Oracle Browser Look – And – Feel (BLAF) user interface guidelines. When these guidelines are followed effectively it brings about the following benefits:

1. Users interact with a consistent, predictable, attractive interface as they navigate from one application to the next application.
2. If properly designed, our applications are likely to be more usable since the user interface guidelines themselves have been usability tested. Further to this the results of product team usability testing are considered and addressed on an ongoing basis.
3. The underlying UIX beans that the OA Framework extends implement the user interface guidelines and we can use the OA web beans 'out of box' without extensive programmatic effort.

In the case of Oracle e-Business Suite applications, any deviations from these guidelines mentioned above must be approved by the corporate user interface design and usability team.

## Implementing the Controller:

To understand more on implementing the Controller we need to understand the generic concepts behind implementing the controller like designing an OA Controller, handling an HTTP GET request, modifying the bean properties, handling an HTTP POST (Form Submit), error handling etc.. We assume here that before getting to know more on implementing the controller we have learnt how to build “Hello World”, Anatomy of an OA Framework Page, Passivation Concept, implementing the model, implementing the view.

## Designing an OA Controller:

As already described in Anatomy of an OA Framework page, the OA Controller is where we define how web beans behave. We write controller code to:

1. Manipulate/Initialize the user interface at runtime (including any programmatic layout that we are unable to do it declaratively)
2. Intercept and handle user events like a button press

Controllers should never include any kind of business logic of its own. This business logic belongs only in our model classes.

Before getting to understand on how to design a OA Controller it is important to consider whether we even need to create a controller. We should write a controller code only if it is absolutely essential. Programmatically created web beans cannot be extended, nor personalized or reused. Further to this some hard – coded layouts may fall out of compliance with Oracle Browser Look – and – Feel (BLAF). While implementing a view, all top – level regions in a shared component must have an associated controller.

OA Controllers can be associated with any region (any web bean that implements `oracle.apps.fnd.framework.webui.beans.OAWebBeanContainer`) we cannot associate

controller with items. Many new OA framework developers wonder just how big a controller should be. This all depends on the exact user requirements.

First and foremost, in a really simple page, we might not have any controllers. If we need to write code, we should consider the following before deciding what controllers to create:

1. The benefits of encapsulation
2. Component reuse
3. Practical Code usability

With all these considerations in mind, there are quite a few guidelines that might help the decision making easier:

1. Never set properties on a parent/grandparent web bean from a child bean
2. Always define controllers so they control the regions with which they are associated or set properties on children/grandchildren of this region. If we want a controller to manage multiple child/grandchild web beans, it should be associated with an appropriate parent/grandparent bean.
3. For complex beans (like `OATableBean`) we should associate a controller with the bean or with a simple containing bean.

In general we should create the least number of controllers per page that satisfies the rules and considerations mentioned above. For a simple page, it is very common to associate a single controller with the `pageLayout` region. For a more complicated page, we might create a few controllers for some functional components. For example a searchable summary page has a 'Search' region controller and a 'Results' region controller.

Within any group of related pages, we can typically find some options to reuse our code. We can add our own private methods to our controllers; we can create a common controller those subclasses

`oracle.apps.fnd.framework.webui.OAControllerImpl` and then subclass this as needed for individual pages and regions. We can even create helper classes to which the controller code can delegate. These classes are not required to subclass/implement any OA Framework classes/interfaces and should be included in the same package as the controller.



The OA Framework is designed to support multithread web bean access. Most is transparent but there are certain rules that we need to follow in our controller code:

1. If we use static methods in our controllers or helper classes, never include the state.
2. Always pass the page's `OAPageContext` to any web bean accessors. For example choose `setText (OAPageContext pageContext, String text)` instead of `setText (String text)`

## Handling an HTTP Get:

During GET processing, each controller's `processRequest (`

`OAPageContext pageContext, OAWebBean webBean)` method is called as the web bean hierarchy is instantiated. Processing begins with the `pageLayout` bean, and proceeds recursively throughout the entire hierarchy. Code that initializes the page – or affects the web bean hierarchy in any way belongs in the `processRequest ()` method.

The `oracle.apps.fnd.framework.webui.OAWebBean` parameter passed to the `processRequest ()` method is the region with which the controller is associated.

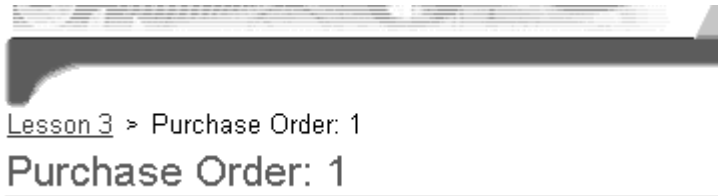
The following example attached explains about the `processRequest ()` method. It illustrates the initialization of a view – only 'detail' page based on request parameter (for a selected purchase order) passed from a 'search' page.



**processRequest.txt**

After calling `super.processRequest (pageContext, webBean)` the example code gets the value for a request parameter named "headerId" (the purchase order number the search page which passed on the request). This value is then displayed in the page title as context for the current user and then passed to the model so that the purchase order can be queried.

Following image displays a dynamically defined page title using page title's value.



Since all the values displayed in the page must be translatable (in different languages) we create a message named FWK\_TBX\_T\_PO\_HEADER\_TEXT in oracle e-business suite message dictionary with the text “Purchase Order: &PO\_NUMBER”.

The code defines the purchase order number as the replacement value for the token PO\_NUMBER and then obtains a translated version of this message from oracle.apps.fnd.framework.webui.OAPageContext. This then sets the translated String as page’s title.

This ‘read-only’ details page automatically queries the given purchase order whenever it is rendered. It does this by passing the purchase order number to a method called initDetails () in the page’s root application module. The application module then passes this parameter to the appropriate view object, which binds the WHERE clause parameter and executes its query.

## Modifying Bean Properties:

It is always preferred to modify web bean properties using partial page rendering (PPR) and SPEL bindings. We normally make changes to web bean properties using the processRequest ().

If we need to programmatically modify the hierarchy in response to a form submit event (For example, the user presses a button) we must forward back to the same page so our processRequest () code can be executed.

To modify a web bean’s properties, we simply need to find the correct bean in the hierarchy using its name (the ID we assigned in JDeveloper) and call the appropriate method as shown in the below example.

When we get a handle to a web bean, always check whether the value is null before calling any of its methods.

```
processRequest (OAPageContext pageContext, OAWebBean webBean)
{
// Always call this before adding our own code.
super.processRequest (pageContext , webBean);
OATableBean table =
(OATableBean) webBean.findIndexedChildRecursive("OrdersTable");
If (table = null)
{
MessageToken [] tokens = {new MessageToken ("OBJECT_NAME", "OrdersTable")}
throw new OAException ("ICX", "FWK_TBX_OBJECT_NOT_FOUND", tokens);
}
// Set the purchase – order specific "control bar" select text:
// "Select Purchase Order (s) and... "
String selectPOText = pageContext.getMessage ("ICX", "FWK_TBX_T_SELECT_PO",
null);
table.setTableSelectionText (selectPOText);
}
```

Starting with the controller's region's children, the findIndexedChildRecursive (String name) method searches the entire web bean hierarchy looking for the first indexed child with a matching name. If the web bean that we want to modify is a UIX named child use the

findChildRecursive (String name) method.

If we need to modify properties on the controller region, cast the processRequest () OAWebBean parameter to the right type and call whatever method we need.

## Handling an HTTP POST (Form Submit):

During HTTP POST processing, the OA Framework first checks to see if the page's web bean hierarchy is available in its cache. If not the user navigates with the browser Back button, the OA Framework must recreate the web bean hierarchy before proceeding. This means that all of the `processRequest ()` code is re-executed as if the browser had issued an HTTP GET request.

This primary POST processing occurs in two separate passes over the web bean hierarchy:

1. First, OA Framework writes from data to the model by calling `processFormData ()` on each web bean recursively until the entire hierarchy is traversed. Any code that needs to be executed during this processing phase should be added in the controller's `processFormData(OAPageContext pageContext, OAWebBean webBean)` method.
2. Assuming that no exceptions were thrown during the first processing phase, OA Framework proceeds to the second phase which involves calling `processFormRequest(OAPageContext pageContext, OAWebBean webBean)` on each web bean.

### `processFormData ()`:

In almost all the cases of the pages we build we will have no cause to overwrite this `processFormData ()` method. If the data source for a region is not a view object, so the view instance and attribute properties are not defined for the individual web beans, then we could code the region's `processFormData ()` to write the child web bean data to the appropriate data source. The OA Framework implements `processFormData ()` at the item level, but we can overwrite it only at the region level, so we must process all of the region's items.

processFormRequest ():

Any code that handles user form submit actions belongs in the processFormRequest () method. The below example gives us an idea on how to write the code in processFormRequest () method. It illustrates how to determine that a particular form submit component was selected and how to initiate a query in the model code and how to perform a JSP forward back to the same page so that the web bean properties can be changed in the processFormRequest () method.

```
public void processFormRequest (OAPageContext pageContext, OAWebBean
webBean)
{
// Always call this before adding the code
super.processFormRequest(pageContext, webBean);
// Pressing the Go button causes the search to be executed.
If (pageContext.getParameter("Go") != null)
{
String orderNumber = pageContext.getParameter("SearchOrder");
String created = pageContext.getParameter("Created");
String showMyOrders = pageContext.getParameter("MyOrders");
OAApplicationModule am = pageContext.getApplicationModule(webBean)
// All parameters passed using invokeMethod () must be serializable.
Serializable [] parameters = { orderNumber,created , showMyOrders };
am.invokeMethod("search", parameters);
// Now forward back to this page so we can implement user interface //changes as a
consequence of query in processRequest (). Never make //user interface changes in
processFormRequest ()
pageContext.setForwardURLToCurrentPage(null , true)
} end processFormRequest ();
```

## Error Handling:

The OA Framework automatically displays any error messages thrown in the model layer (business logic layer). We don't need to do anything in our controller to facilitate this.

We can see the error handling for information about throwing exceptions in our controller code and to display Error, Warning, Confirmation and information messages at top of each OA Framework page.

To understand more on Error Handling in OA Framework page we need to have a basic knowledge on the following:

1. Exception Types
2. Exception Classes
3. Bundled Exceptions
4. Exception Examples
5. Dialog Pages and Message Boxes

### Exception Types:

The OA Framework handles three basic types of exceptions: general, validation and severe.

**General Exceptions** – Errors in the BC4J framework are handled by throwing an implicit (runtime) exception of the type `oracle.jbo.JBOException`. The OA Framework has its own specialization of this called `oracle.apps.fnd.framework.OAException`. This specialization provides a mechanism for bundling multiple exceptions together while also translating the exception messages using Oracle Applications Message Dictionary, so that useful messages can be displayed.

**Validation Exceptions** – Validation Exceptions are thrown from entity objects and view objects for both row and attribute level validation failures.

1. `oracle.apps.fnd.framework.OAAttrValException` – specialization of `OAException` used for attribute level validation failures.

2. `oracle.apps.fnd.framework.OARowValException` – specialization of `OAException` used for row (entity) level validation failures.

The OA Framework displays error messages to the user at different levels as follows:

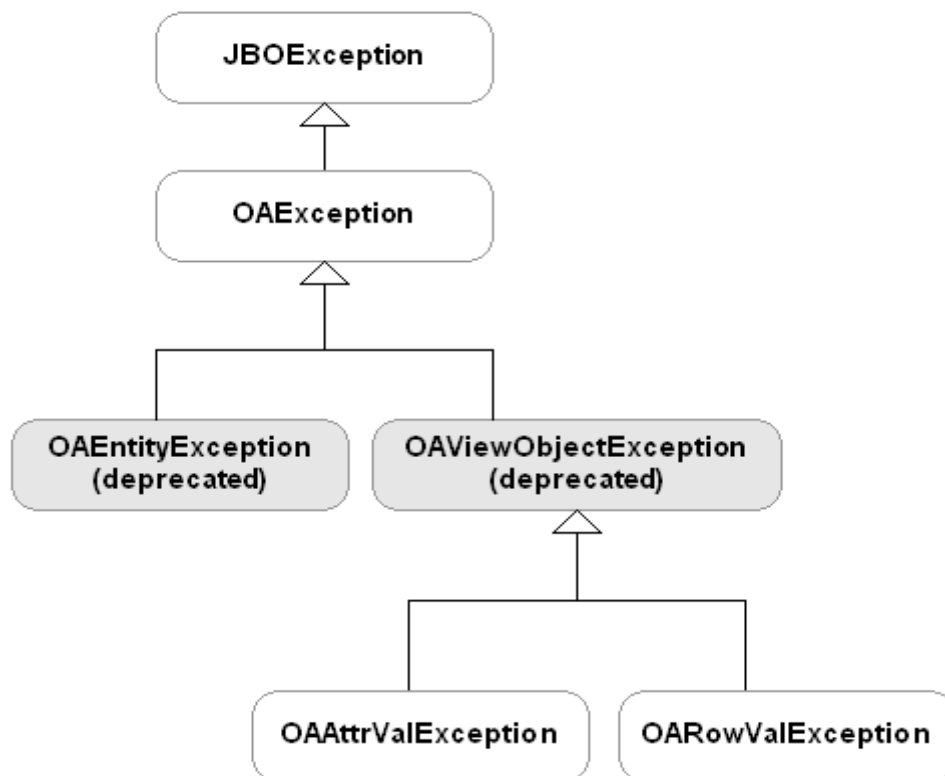
1. Attribute – level exceptions are visually indicated on the error items and at the top of the page.
2. Row – Level exceptions are visually indicated on the error row and at the top of the page.
3. Page – Level exceptions are visually indicated at the top of the page.

**Severe Exceptions** – Severe (or “fatal”) exceptions include unexpected system – level errors (like a `NullPointerException`) and selected `JBOExceptions` like `NoDefException`. We can also deliberately throw a severe exception in our code. Since OA Framework release 11.5.57, if a fatal exception occurs, the user is directed to the `OAErrorPage`. If the fatal exception occurs in the middle of page rendering, the page is partially displayed with a user – friendly error message that includes a link to the details of the exact error. The `OAErrorPage` also displays a user – friendly error message with a link.

The OA Framework also ships a seeded business event (`oracle.apps.fnd.framework.OAFatalError`) which sends a notification to the `SYSADMIN` user whenever OA Framework page reports a severe or fatal exception. This notification includes the detailed error for the fatal exception and information about the user who has encountered the exception.

## Exception Classes:

The OA Framework exception inheritance hierarchy is shown in figure below. The `OAException` superclass extends `JBOException`. `OAAttrValException` and `OARowValException` extend `OAViewObjectException`, a deprecated class that extends `OAException`.



## OAException:

`OAException` is the exception that we would throw in generic error cases. For example if we have encountered an unexpected problem in some controller code as shown:

```
OACellFormatBean shipTermsCell =  
(OACellFormatBean) webBean.findIndexedChildRecursive("ShipTermsCell");  
if (shipTermsCell = null)  
{
```



```
MessageToken[] tokens = { new MessageToken("OBJECT_NAME",  
"ShipTermsCell")};
```

```
Throw new OAException("AK", "FWK_TBX_OBJECT_NOT_FOUND", tokens);
```

We have created a message in Oracle Applications Message Dictionary (FWK\_TBX\_OBJECT\_NOT\_FOUND) for AK application. The message was defined with a token (OBJECT\_NAME) that we replace with the name of the user interface component. The OA Framework will use this information to display a properly translated error message at the top of the page.

Since the OAException is a flexible object that can be used to display other types of messages to the user (Information, Confirmation and Warning) we can also instantiate it with a message type.

```
MessageToken [] tokens = { new MessageToken("SUPPLIER_NAME",name),  
new MessageToken("SUPPLIER_NUMBER", supplierId) };  
OAException confirmMessage = new  
OAException("AK","FWK_TBX_T_SUPPLIER_CREATE_CONF",tokens,  
OAException.CONFIRMATION,null);
```

The OAException , OAAtrValException and OARowValException classes all include constructors (i.e methods) which accept a message type parameter. The message type parameter tells OA Framework the type of message to display to a user. Valid options in this include:

- 1.OAException.ERROR
- 2.OAException.WARNING
- 3.OAException.INFORMATION
- 4.OAException.CONFIRMATION
- 5.OAException.SEVER

## OAAttrValException:

If any attribute – level validation fails in a view object row or an entity object, we can throw OAAttrValException as shown below.

To instantiate this exception we must pass the following information :

- 1.Source object type (OAException.TYP\_ENTITY\_OBJECT or OAException.TYP\_VIEW\_OBJECT)
- 2.Full entity definition name or view instance name
- 3.Primary key of the entity or row
- 4.Attribute name being validated
- 5.Attribute value that has failed validation
- 6.Error Message application short name
- 7.Error Message Name

An example of Entity Object:

```
public void setSalary(Number value)
{
    If (value != null)
    {
        //verify value is > 0
        If (value.compareTo(0) <= 0)
        {
            throw new OAAttrValException(OAException.TYP_ENTITY_OBJECT,
            // this indicates EO source
            getEntityDef().getFullName(), //entity name
            getPrimaryKey(), //entity primary key
            "Salary", // attribute name
            Value, // bad attribute value
            "AK", // message application short name
            "FWK_TBX_T_EMP_SALARY_REQUIRED"); //
            message_name
        }
        setAttributeInternal(SALARY, value);
    }
}
```

```
}  
} // end setSalary()
```

An example of View Row:

```
setDescription(String value)  
{  
    if ("XXX.equals(value)  
    {  
        throw new OAAtrValException (  
            OAException.TYP_VIEW_OBJECT, // indicates VO row source  
            getViewObject().getFullName(), // View object full usage name  
            getKey(), // row primary key  
            "Description", // attribute name  
            value, // bad attribute value  
            "FND", // message application short name  
            "ATTR_EXCEPTION"); // message name  
    }  
    setAttributeInternal("Description",value);  
} // end setDescription()
```

### OARowValException:

If any row – level validation fails in a view object row or an entity object, we can throw a OARowValException as shown.

To instantiate this exception we must pass the following information:

- 1.Full Entity definition name or view instance name
- 2.Primary key of the entity or row
- 3.Error message application short name
- 4.Error Message name

Entity Object Example:

```
protected void validateEntity()  
{
```

```
super.validateEntity();
if (attr1 != attr2)
throw new OARowValException (
getEntityDef().getFullName(), // entity full definition name
getPrimaryKey(), // entity object primary key
"FND", // message application short name
"ATTR_EXCEPTION"); //message name
}
```

View Row Example:

```
protected void validate()
{
super.validate();
if (attr1 != attr2)
throw new OARowValException (
getViewObject().getFullName(), // View object full usage name
getKey(), // row primary key
"FND", // message application short name
"ATTR_EXCEPTION"); // message name
}
```

-

## Hands – On Exercises:

Please consult with your faculty.