Web Dynpro Java for Experts: Web Dynpro Component Interface Definitions In Practice - Exercises and Solutions

Summary

Two exercises describe how to define a Public Part and a component usage relation, how to define an external context mapping relation, how to define a component usage relation to a Web Dynpro component interface definition, how to retrieve configuration data with the (I)WDConfiguration service API, how to apply context-based dependency injection between a locator component and a root component, how to create a component instance for a used component interface definition at runtime and finally how to configure the used component implementation in the Visual Administrator so that no code-modification is required.

Author: Bertram Ganz

Company: SAP AG

Created on: 12 december 2006

Author Bio

After his studies in mathematics, physics and computer science Bertram finished his teacher training at a German grammar school stressing technical sciences. He is now working in the Web Dynpro Java Runtime team for 3 years concentrating on Web Dynpro Knowledge Transfer.

Table of Contents

Exercise 1 – Defining External Context Mapping

This first exercise dealing with Web Dynpro Componentization is based on two incomplete Web Dynpro DC templates. After having finished this exercise the following learning objectives will be fulfilled:

- Exposing a component to other Web Dynpro DCs with a *Public Part definition*.
- Defining a *Public Part Usage* in a Web Dynpro DC.
- Defining a *component usage relation* between root and UI components.
- Defining *an external context mapping relation* from a child (UI component) to its parent component (root component).

🕒 *Estimated time required for completion: 30 min*

Related Web Dynpro DCs

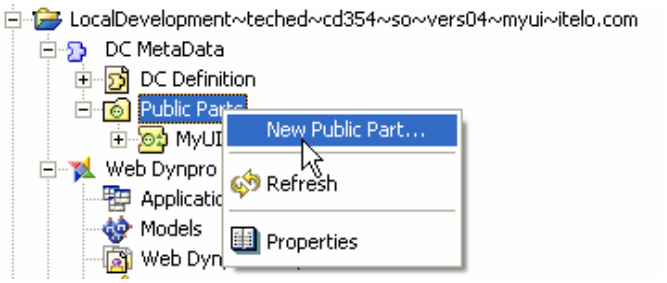
This Exercise is based on the following Web Dynpro DCs

1. **LocalDevelopment~teched~cd354~so~vers04~myui~itelo.com**
2. **LocalDevelopment~teched~cd354~so~vers04~root~itelo.com**

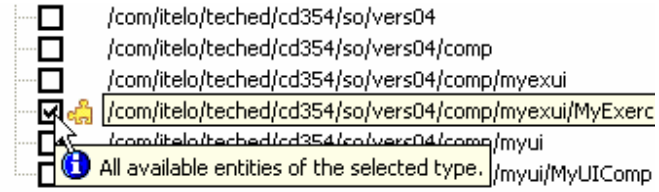
STEP 1 – Adding a Public Part to the UI Component's DC …~so~vers04~myui~itelo.com

The UI component *MyExerciseUIComp* is exposed to the root component of a separate Web Dynpro DC by adding it to a new Public Part of the DC **~vers04~myui~itelo.com.**

1. Open the Web Dynpro DC **LocalDevelopment~teched~cd354~so~vers04~myui~itelo.com** within the Web Dynpro Explorer
2. Define a new **Public Part** with the context menu item *New Public Part …* of the node *DC Metadata → Public Parts.*
3. Create new Activity *CD354 – Exercise 1*

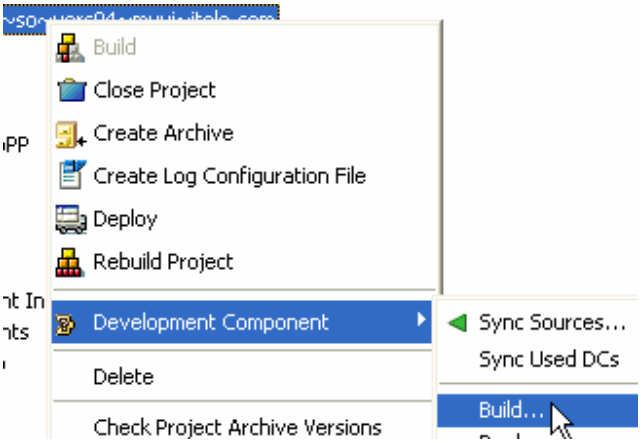


4. The *Add Public Part window* appears. Enter the name: **MyExerciseUICompPP**
5. Within the next and final wizard step you add a new *Public Part Entity* of type *Web Dynpro Component*:
   a. Select Entity Type: *Web Dynpro Component*
   b. Select Entity: */com/itelo/teched/cd354/so/vers04/comp/myexui/MyExerciseUIComp*



The newly added Public Part gets only visible outside the UI DC after having created the related *Public Part archive*:

6. Open the context menu of the Web Dynpro DC node **LocalDevelopment~teched~cd354~so~vers04~myui~itelo.com**
7. Select the menu item *Development Component → Build …*



Result

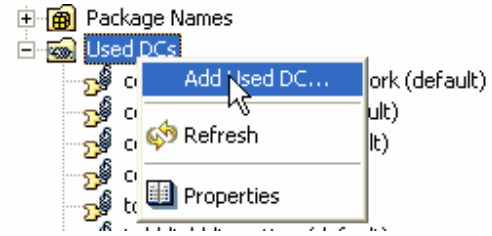
The new Public Part *MyExerciseUICompPP* is added to the Web Dynpro DC **…~so~vers04~myui~itelo.com.** Via the generated Public Part archive it can now be used by the root DC **…~so~vers04~root~itelo.com.**

STEP 2 – Adding a Public Part Usage within the Root Components's DC …~so~vers04~root~itelo.com

The two components *MyExerciseUIComp* and *ExerciseRootComp* are contained in two separate DCs. To define the usage of component *MyExerciseUIComp* within the component *ExerciseRootComp* you must define a *Public Part usage relation*. This usage specifies a dependency between a Development Component and the entities exposed by the Public Part definition of another Development Component.

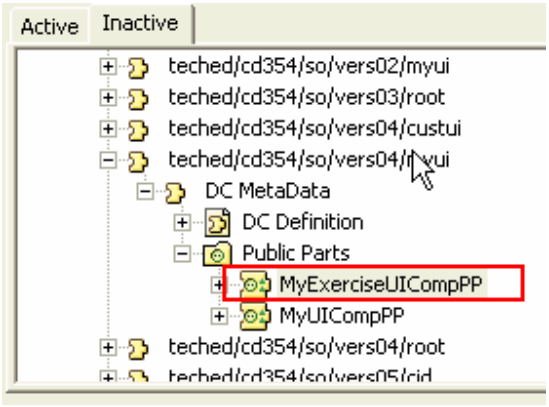
1. Open the Web Dynpro DC **…~vers04~root~itelo.com** within the Web Dynpro Explorer.

2. Define a new Public Part Usage with the context menu item *Add Used DC…* of the node *DC Metadata → DC Definition → Used DCs*



The *Add Public Part window* appears.

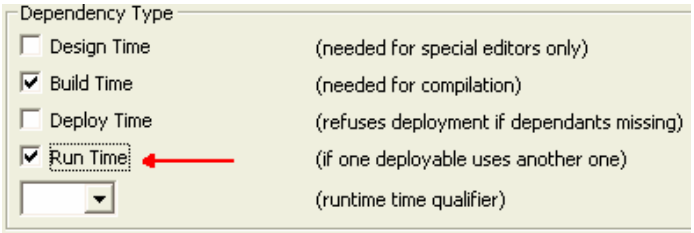
3. Within *the Inactive DCs* tab select the node *teched/cd354/so/vers04/myui → DC MetaData → Public Parts →* ***MyExerciseUICompPP***



The Public Part dependency type *Build Time* is marked by default. When using the Public Part of a Web Dynpro DC you should also make the following definition:

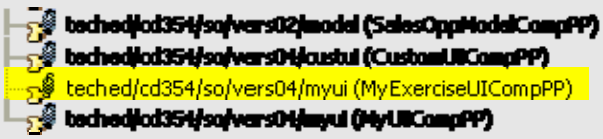
4. Mark the checkbox *Dependency Type → Run Time*

The definition of a new Public Part usage is now completed.



Result

The new Public Part usage ***teched/cd354/so/vers04/myui (MyExerciseUICompPP)*** is displayed in the Web Dynpro Explorer.

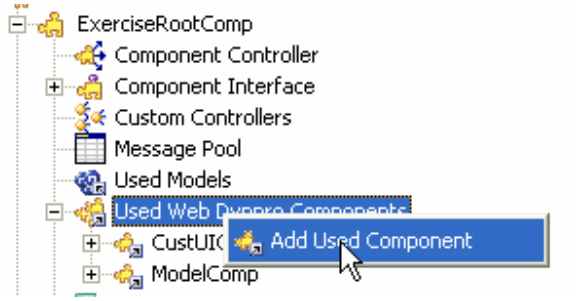


Continue with Step 3 on next page …

STEP 3 – Defining a Component Usage Relation between the Components *ExerciseRootComp* and *MyExerciseUIComp*

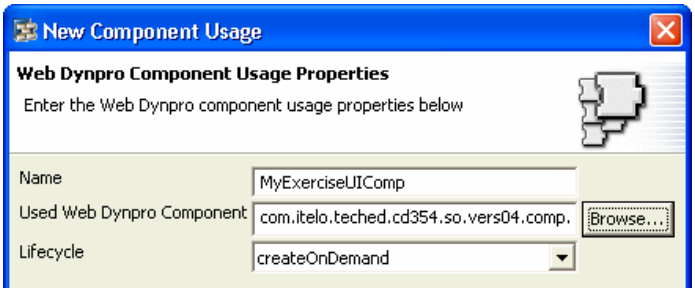
After having defined the usage of Public Part ***MyExerciseUICompPP*** we can define a component usage relation of the *ExerciseRootComponent* to the *MyExerciseUICompoent*.

1. In DC **…~vers04~root~itelo.com** open the WD component ***ExerciseRootComp***
2. Define a new component usage with the context menu item *Add Used Component* of the node *ExerciseRootComp → Used Web Dynpro Components.*



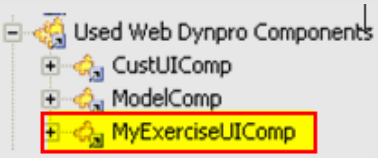
3. Within the next wizard step define the following component usage:
   a. Name: **MyExerciseUIComp**
   b. Used Web Dynpro Component: browse to *com.itelo.teched.cd354.so.vers04.comp*

      .***MyExerciseUIComp***
   c. Lifecycle: *createOnDemand*



Result

The usage of component *MyExerciseUIComp* is now defined within the root component *ExerciseRootComp*.

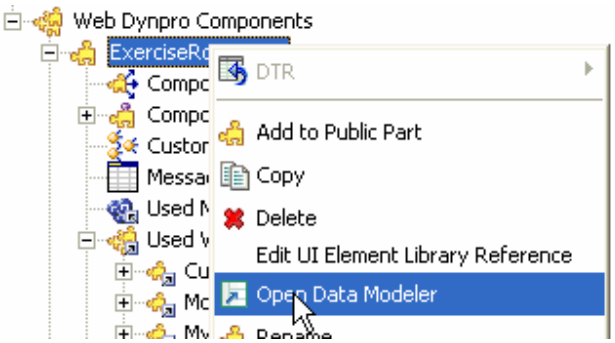


Continue with Step 4 on next page …

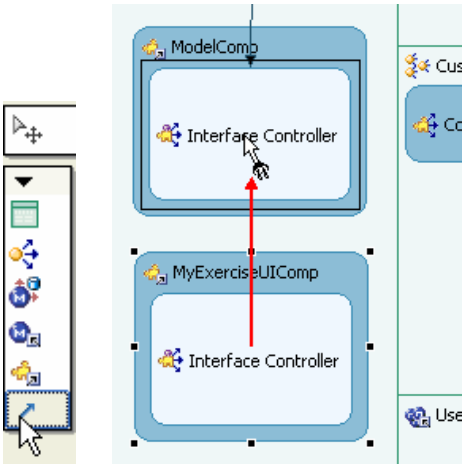
STEP 4 – Defining an External Context Mapping Relation

To feed the interface context of the used UI component *MyExerciseUIComp* with the context data provided by the model component you must define an *external context mapping relation*.

1. In DC **…~so~vers04~root~itelo.com** navigate to node *Web Dynpro→ Web Dynpro Components→* ***ExerciseRootComp***
2. Double-click this node to open the *data modeler*. Alternatively you can select the context menu item *Open Data Modeler.*

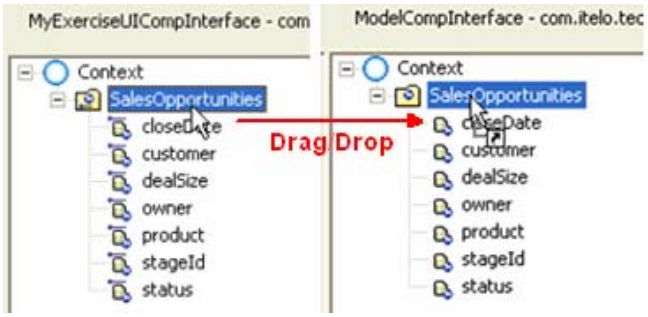


3. In the left toolbar select the icon *Create a data link*
4. Draw a *data link* from the interface controller of the *MyExerciseUIComp* to the interface controller of the used *ModelComp.* Keep the mouse button pressed while drawing the data link.

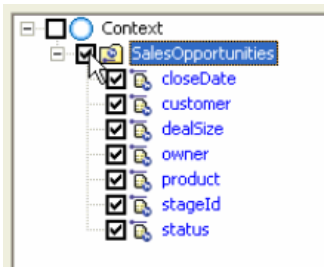


In the appearing dialog window you can define the *external context mapping* relation between the two context nodes with name *SalesOpportunities*.

5. At the left side click on the node *SalesOpportunities* (interface context of the *MyExerciseUIComp*) and move the cursor to the node *SalesOpportunities* at the right side (model interface controller context). Stop pressing the mouse button.



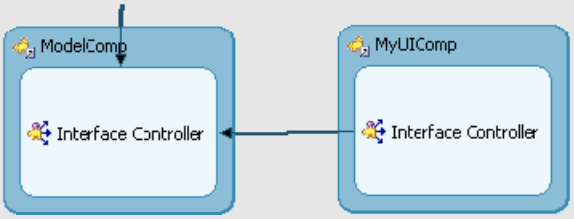
6. In the next wizard step mark the checkbox for the model node *SalesOpprotunities* (and all its model attributes). Choose *Ok*.
7. Complete this step with choosing *Finish* in the last wizard window.



Result

With this step the (external) context mapping chain gets closed:

- In **MyExerciseUIComp**: *View context → Component Context → Component Interface Context (input node with isInputElement=true) →*
- In **MyExerciseUIComp usage***: Component Interface Context (externally mapped node) →*
- In **ModelComp usage**: *Component Interface Context → Model Component Context (data node)*



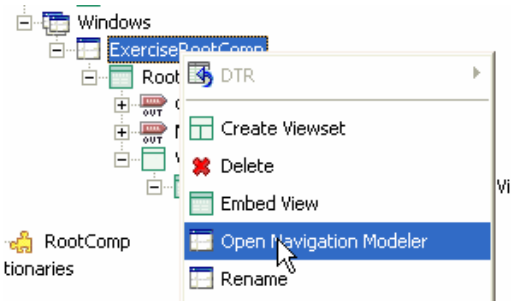
Continue with Step 5 on next page …

STEP 5 – Visually Embedding the Component Interface View of the UI Component within the Root Component

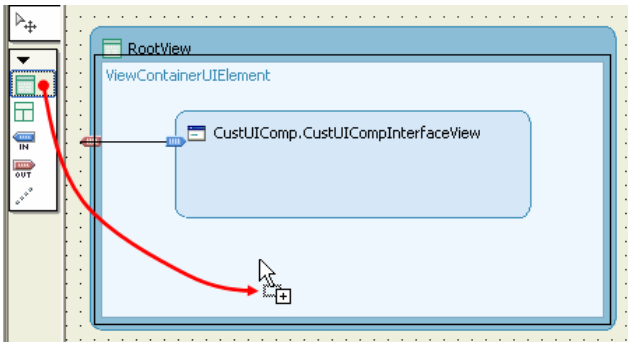
The visual interface of the used *MyExerciseUIComponent* (component interface view) can now be embedded in the *ViewUIElementContainer* of the *RootView* (in *ExerciseRootComponent*). This step is done using the navigation modeler.

1. In DC **…~so~vers04~root~itelo.com** navigate to node *Web Dynpro → Web Dynpro Components → ExerciseRootComp → Windows →* ***ExerciseRootComp***

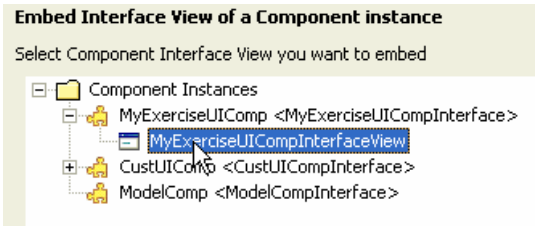
2. Double-click this node to open the *navigation modeler*. Alternatively you can select the context menu item *Open Navigation Modeler.*



3. In the left toolbar select the icon *Embed a view.* Afterwards this icon is marked grey.
4. Move the mouse cursor above the area *RootView – ViewContainerUIElement* and click the mouse button.
5. The *Embed view window* appears. Select item *Embed Interface View of a Component Instance* and click on *Next*.

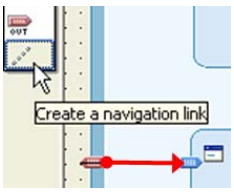


6. In the next step you can embed the interface view of the component usage *MyExerciseUIComp*. Select the node *Component Instances → MyExerciseUIComp → MyExerciseUICompInterfaceView* and click on *Finish*.
7. Make sure that the embedded interface view of component *MyExerciseUIComp* is the **default view** within the *ViewContainerUIElement*.



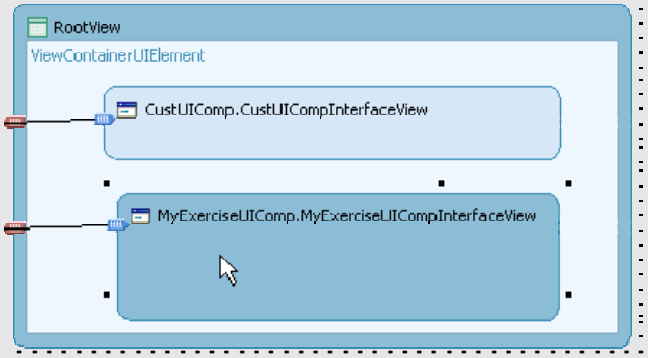
The embedded *MyUIExerciseUICompInterfaceView* is displayed in the navigation modeler. To navigate to this interface view at runtime we must define a navigation link next.

8. Create a navigation link from *outbound* plug *MyUIOut* (of *RootView*) to inbound plug *Default* (of embedded interface view)



Result

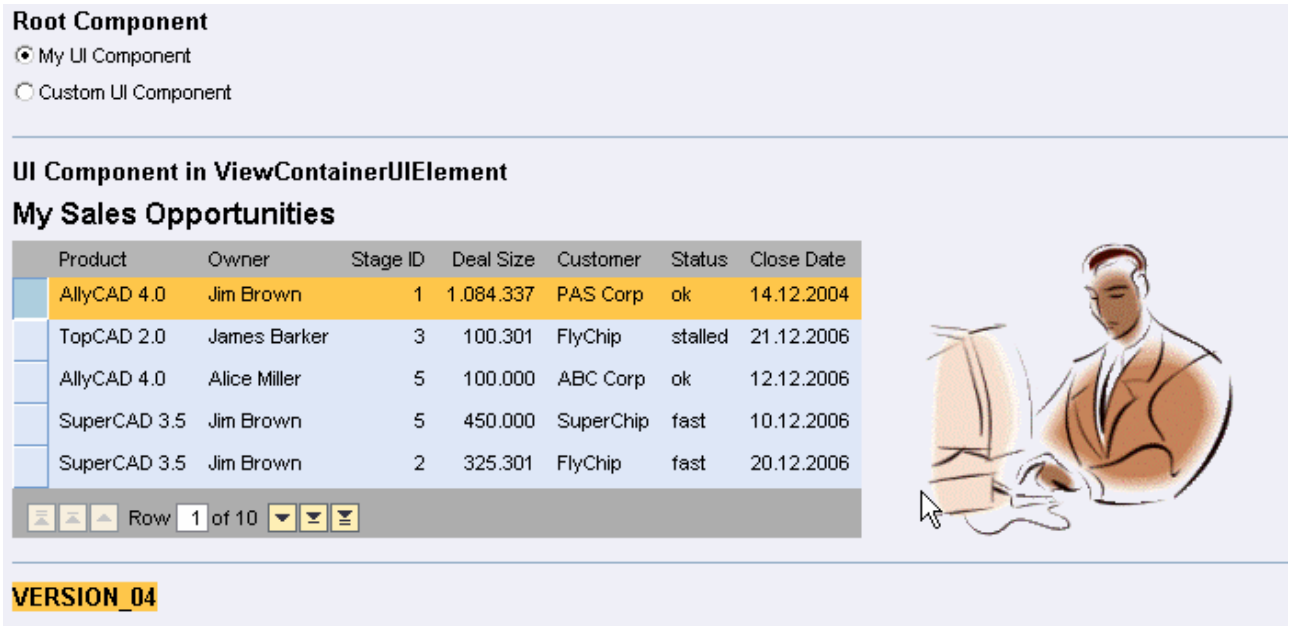
You have successfully embedded a component interface view of the used *MyExerciseUIComponent* in the root component window.



Testing the Exercise Application …

Testing the Exercise Application

1. To save the metadata of your project, choose  (*Save All Metadata*) in the toolbar.
2. In the Web Dynpro Explorer, open the node ***…~teched~cd354~so~vers04~root~itelo.com*** → *Web Dynpro* → *Applications* → *Exercise_SalesOppUIApp*
3. In the node's context menu, choose ***Deploy new Archive and Run***. The exercise application is then started in a new browser window.
4. In case you have successfully completed the exercise steps the interface view of component *MyExerciseUIComp* is displayed in the browser client:



Exercise 2 – Configuring a UI Component Implementation with a Locator Component

This second exercise dealing with Web Dynpro Componentization is based on four Web Dynpro DCs and two partially incomplete Web Dynpro DC templates. After having finished this exercise the following learning objectives will be fulfilled:

- Understanding the component-oriented and DC-separated architecture of the *SalesOpportunites* sample application .
- Defining a component usage relation to a Web Dynpro component interface definition.
- Reading configuration data with the *(I)WDConfiguration* service API.
- How to apply *context-based dependency injection* between a *locator component* and a CID (component interface definition) client component (here the root component).
- How to create a component instance for a component usage of type *ISalesOppUIComp* (CID) at runtime.
- Configuring the used component implementation in the Visual Administrator (→ no code modification required).

🕒 *Estimated time required for completion: 30 min*

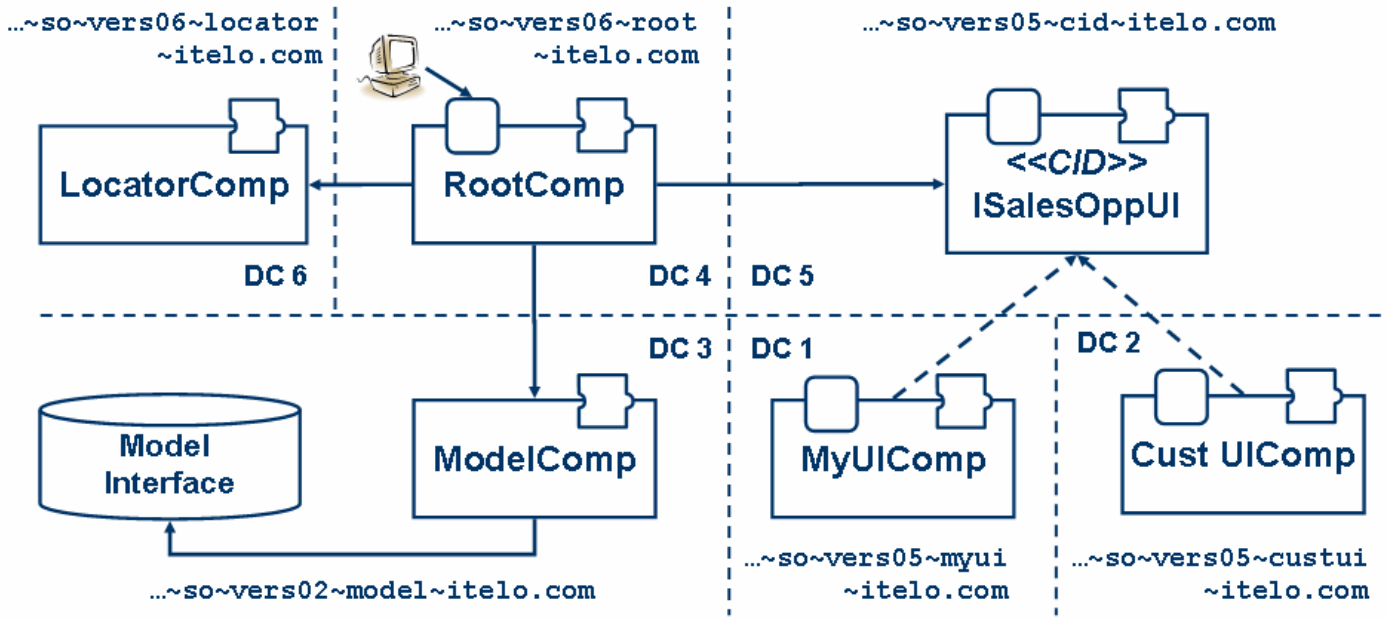
Related Web Dynpro DCs

This Exercise is based on the following Web Dynpro DCs. The incomplete DCs used in this exercise are marked **bold**.

1. LocalDevelopment~teched~cd354~so~vers05~myui~itelo.com
2. LocalDevelopment~teched~cd354~so~vers02~model~itelo.com
3. LocalDevelopment~teched~cd354~so~vers05~custui~itelo.com
4. **LocalDevelopment~teched~cd354~so~vers06~root~itelo.com**
5. LocalDevelopment~teched~cd354~so~vers05~cid~itelo.com
6. **LocalDevelopment~teched~cd354~so~vers06~locator~itelo.com**

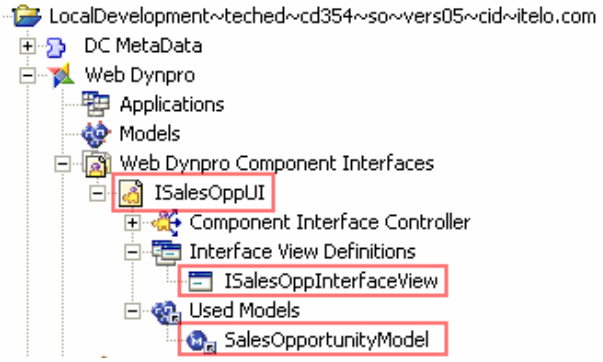
STEP 1 – PREPARATION – Understanding the Component Architecture of the Sales Opportunities Sample Application

Before you start with completing the prepared exercise components you should have a look at the component architecture of the Sales Opportunities sample application.

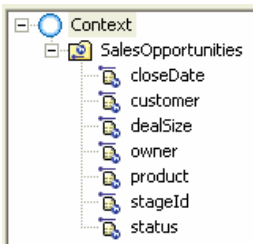


Component Interface Definition *ISalesOppUI*

1. Select the Web Dynpro DC **...~cd354~so~vers05~cid~itelo.com** within the Web Dynpro Explorer
2. Look at the component interface definition (*CID*) **ISalesOppUI** under the node *Web Dynpro → Web Dynpro Component Interfaces.*
3. The interface view **ISalesOppInterfaceView** was defined explicitly.
4. As the context of the component interface controller is bound to the model *SalesOpportunityModel*
5. A *model usage relation* must additionally be defined in the CID.

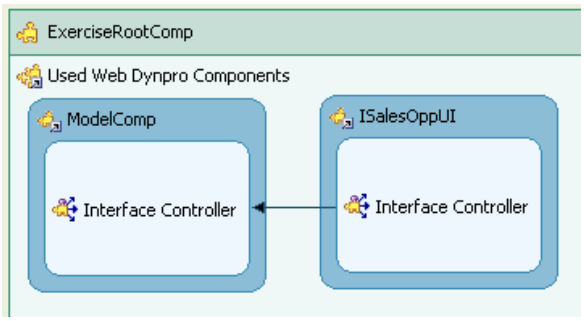


Interface context of the **ISalesOppUI** component interface definition.



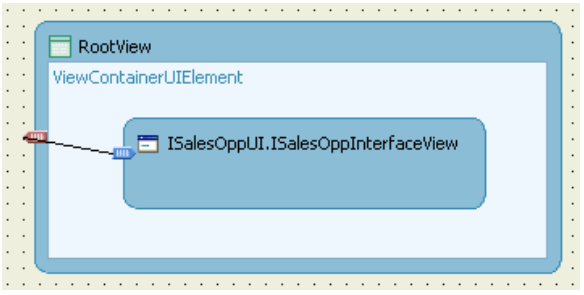
6. The model nodes and attributes are defined as *input*-Elements to allow external context mapping on the component usage level (in root component).

7. Select the DC **...~so~vers06~root~itelo.com** and open the *data modeler* for component ***ExerciseRootComp***
8. Within the *ExerciseRootComponent* a component usage relation to the component interface definition **ISalesOppUI** is defined.
9. The interface controller context of the **ISalesOppUI** component is externally mapped to the interface context of the used model component.



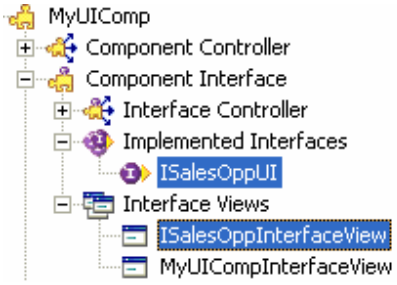
10. Open the *navigation modeler* for the window *Web Dynpro → Web Dynpro Components → ExerciseRootComp → Windows →* ***ExerciseRootComp***
11. The interface view *ISalesOppInterfaceView* of the used CID **ISalesOppUI** is embedded in the root component's *RootView*.

    Independent from different component implementations used at runtime all declarations are based on a single component usage (pointing to a component interface definition) at design time.



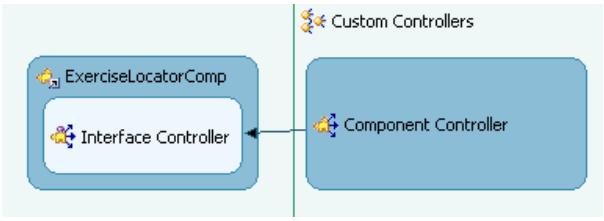
Definition of the CID implementation dependency

12. Select the Web Dynpro DC **...~cd354~so~vers05~myui~itelo.com** and navigate to the component *MyUIComp*
13. Open the nodes *Web Dynpro → Web Dynpro Components → MyUIComp → Component Interface → Implemented Interfaces →* ***ISalesOppUI*** and … *→ Ineterface Views →* ***ISalesOppInterfaceView***



Locator Component *ExerciseLocatorComp*

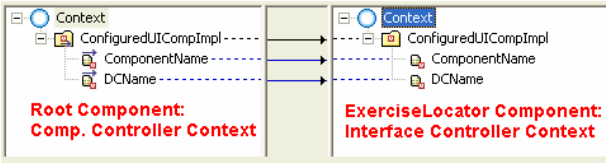
14. Select the Web Dynpro DC **...~cd354~so~vers06~root~itelo.com** and open the component modeler for the component *ExerciseRootComp.*
15. A component usage relation to the *ExerciseLocatorComp* is already defined in the exercise root component.
16. A data link visualizes the context mapping from the component controller context to the interface context of the



ExerciseLocator component.

17. The locator component provides access to the fully qualified name of the configured UI component implementation via two value attributes of type *String*:
    a. *ComponentName*: qualified name of the configured UI component implementation
    b. *DCName*: name of the Web Dynpro DC in which the implementing UI component is deployed.

Context-based dependency injection between the locator component (service component) and the root component (client component)



Configuration Properties File

18. The configuration properties file **uiCompImpl.properties** of the component *ExerciseLocatorComp* is stored under the folder *src/configuration/Components/com.itelo.teched.cd354.so.vers06.comp.exlocator.ExerciseLocatorComp*

19. The fully qualified name of a UI component implementing the CID *ISalesOppUI* is defined with the two property keys **dcname** and **compname**.

 # SalesOppImpls.properties

dcname = itelo.com/teched~cd354~so~vers05~myui

compname = com.itelo.teched.cd354.so.vers05.comp.myui.MyUICo mp

STEP 2 – Reading Configuration Data (Fully Qualified UI Component Name) in the ExerciseLocator Component

The properties file for configuring the used UI component implementation is deployed with the archive of the locator component. All key-value pairs are persisted in the database of the SAP J2EE Engine and can be retrieved at runtime using the special Web Dynpro service APIs **IWDConfiguration** and **WDConfiguration**.

1. Select the Web Dynpro DC **…~so~vers06~locator~itelo.com** and open the component controller of the component *ExerciseLocatorComp* via the node *Web Dynpro → Web Dynpro Components → ExerciseLocatorComp → Component Controller.*
2. Implement the following lines of code in the controller's *readConfiguration()* method.

ExerciseLocatorComp.java - Component Controller - DC …~so~vers06~locator~itelo.com

```
//@@begin javadoc:readConfiguration()
/** Declared method. */
//@@end
public void readConfiguration( )
{
  //@@begin readConfiguration()
  //       TODO 1 - Ex 2 - read configuration data and set context attributes
```



```
  try {
    // Load configuration file which is deployed with the Locator Component-
    // The configuration file 'ISalesOppUICompImpls.properties' is located under ...
    //    src/configuration/Components/com.itelo.teched.cd354
    //     .so.vers06.comp.exlocator.ExerciseLocatorComp
    IWDConfiguration configuration =
      WDConfiguration.getConfigurationByName(
        wdComponentAPI.getDeployableObjectPart(),
        "uiCompImpl");
    // read configuration data defining the fully qulaified name of the
    // configured UI Component implementation.
    wdContext.currentConfiguredUICompImplElement().setDCName(
      configuration.getStringEntry(ConfigKey.DC_NAME));
    wdContext.currentConfiguredUICompImplElement().setComponentName(
      configuration.getStringEntry(ConfigKey.COMP_NAME));
  } catch (WDInvalidConfigParameterException e) {
    wdComponentAPI.getMessageManager().reportException(e.getLocalizedMessage(), true);
  } catch (WDConfigurationNotFoundException e) {
    wdComponentAPI.getMessageManager().reportException(e.getLocalizedMessage(), true);
  }
  //@@end
}
```

Result

The locator component has successfully retrieved the configured, fully qualified name of the *ISalesOppUI* component implementation to be created at runtime. This configuration data is exposed to the root component via the locator component's interface context. The root component can access this data by defining a corresponding context mapping relation (***context-based dependency injection***) – see *Step 1/ Preparation/ 11-13* within this exercise.

STEP 3 – Creating a Component Instance for the Component Usage of type *ISalesOppUIComp*

The root component can create a component instance of type *ISalesOppUI* by reading the fully qualified component name from the interface context of the locator component (context mapping relation).

1. Select the Web Dynpro DC **…~so~vers06~root~itelo.com** and open the *component controller* via the node *Web Dynpro → Web Dynpro Components → ExerciseRootComp → Component Controller.*
3. Implement the following lines of code in the controller methods *wdDoInit()* and *createUICompInstance()*.

ExerciseRootComp.java - Component Controller - WD DC …~so~vers06~root~itelo.com

```
//@@begin javadoc:wdDoInit()
/** Hook method called to initialize controller. */
//@@end
public void wdDoInit()
{
  //@@begin wdDoInit()
  // TODO 2 - Ex 2 - invoke private method to create a UI component instance
  wdThis.createUICompInstance();
  //@@end
}
```

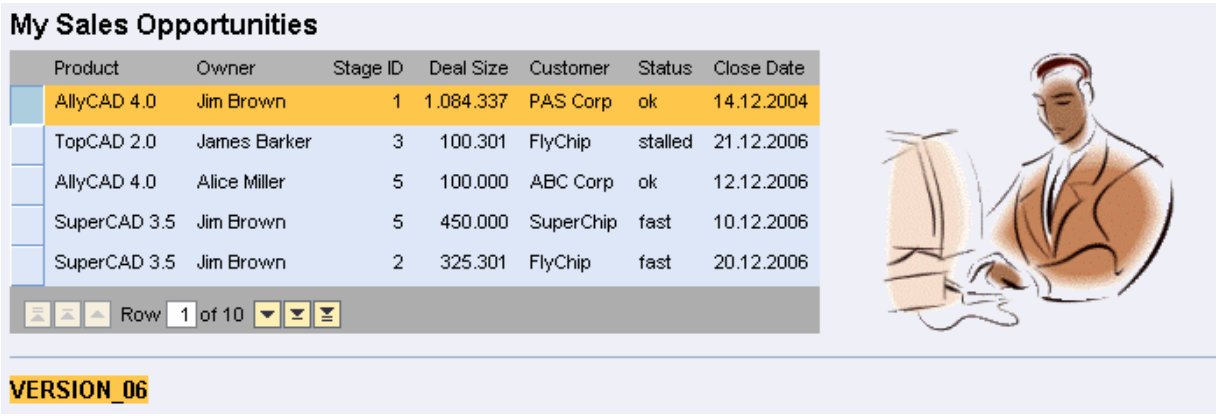
4. The creation of a component instance which implements the component interface definition *ISalesOppUIComp* is done by reading the fully qualified component name from the context. The context attributes **ComponentName** and **DCName** are mapped to the interface context of the *ExerciseLocator* component. The locator component retrieves the configuration data from the database of the *SAP NetWeaver Application Server for Java*.

```
//@@begin javadoc:createUICompInstance()
/** Declared method. */
//@@end
public void createUICompInstance( )
{
  //@@begin createUICompInstance()
  // TODO 3 - Ex 2 - create an instance of the configured UI component
  wdThis.wdGetISalesOppUIComponentUsage().createComponent(
    wdContext.currentConfiguredUICompImplElement().getComponentName(),
    wdContext.currentConfiguredUICompImplElement().getDCName());
  //@@end
}
```

Testing the Exercise Application …

Testing the Second Exercise Application

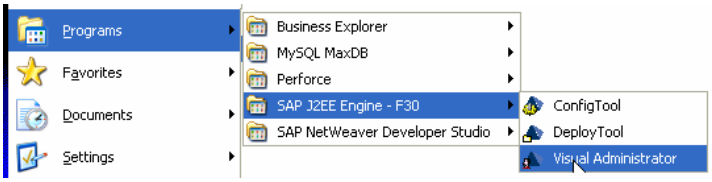
5. To save the metadata of your Web Dynpro DCs, choose  (*Save All Metadata*) in the toolbar.
6. Build and deploy the changed Web Dynpro DC ***…~teched~cd354~so~vers06~locator~itelo.com*** comprising the *ExerciseLocatorComp*.
7. In the Web Dynpro Explorer, open the node ***…~teched~cd354~so~vers06~root~itelo.com*** → *Web Dynpro* → *Applications* → ***Exercise_SalesOppUIApp***
8. In the node's context menu, choose ***Deploy new Archive and Run***.
9. In case you have completed the exercise steps successfully the interface view of component *MyExerciseUIComp* is displayed in the browser client; this means the configuration data was read from the database:



STEP 4 – Configuring the Used Component Implementation in the Visual Administrator

Within the Visual Administrator UI an administrator can define a **custom configuration** which replaces the deployed *configuration base* or d*efault configuration.*

1. Start the **Visual Administrator** of your running SAP J2EE Engine by selecting the Windows menu item *Start* → *Programs* → *SAP J2EE Engine – F26* → *Visual Administrator*



2. Login with the user/password provided by the speaker.
3. Open the **Configuration Adapter** under the node Server → *Services.*
4. In the work area open the tab ***Runtime → Display configuration.***
5. Switch to **Edit mode** by pressing the edit button and confirm the next dialog.

6. Select the configuration property sheet **uiCompImpl** under the node *Configurations → webdynpro → itelo.com → teched~cd354~si~vers06~locator → Components → com.itelo.teched.cd354.so.vers06 .comp.exlocator.ExerciseLocatorComp →* ***Propertysheet uiCompImpl.***

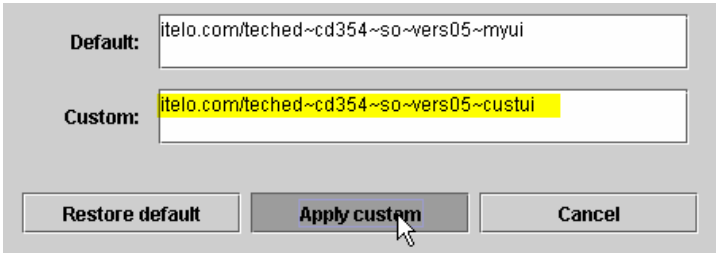
7. You can edit a selected property sheet by pressing the edit button in the toolbar or via double-click.

8. Change the configuration property **compname** from *default* to *custom*:

   com.itelo.teched.cd354.so.vers05.comp.custui .CustUIComp

9. Apply this new custom value.

10. Change the configuration property **dcname** from *default* to *custom*:
11. *Custom*: **itelo.com/teched~cd354~so~vers05~custui**
12. Apply this new custom value.



13. Re-start the *Sales Opportunity Application* in the Web Browser Client by pressing the **Refresh** button.

Result

As the locator component retrieved a different fully qualified name of the configured component implementation the UI component of type *ISalesOppUI* is replaced with the customer's UI component implementation **CustUIComp.**

Customer's Sales Opportunities

Opportunities

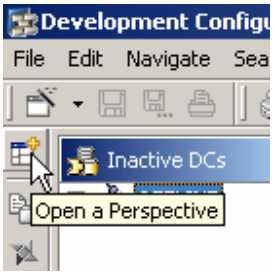
	Product	Owner	Stage ID	Deal Size	Customer	Status	Close Date
	AllyCAD 4.0	Jim Brown	1	1.084.337	PAS Corp	ok	14.12.2004
	TopCAD 2.0	James Barker	3	100.301	FlyChip	stalled	21.12.2006
	AllyCAD 4.0	Alice Miller	5	100.000	ABC Corp	ok	12.12.2006
	SuperCAD 3.5	Jim Brown	5	450.000	SuperChip	fast	10.12.2006
	SuperCAD 3.5	Jim Brown	2	325.301	FlyChip	fast	20.12.2006
	TopCAD 2.0	Alice Miller	3	100.301	PC4U	stalled	15.10.2006
	AllyCAD 4.0	John Wheaton	4	600.000	MBI	ok	14.12.2006
	SuperCAD 3.5	Jim Brown	6	750.000	FlyChip	committed	20.12.2006
	TopCAD 2.0	Fred Parsons	3	800.301	FlyChip	stalled	21.12.2006
	AllyCAD 4.0	John Wheaton	5	300.000	ABC Corp	ok	12.12.2006

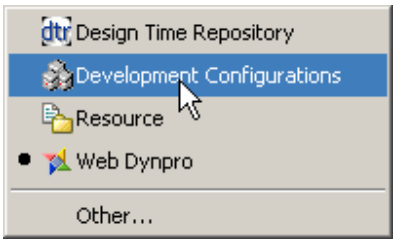
Row 1 of 10

VERSION_06

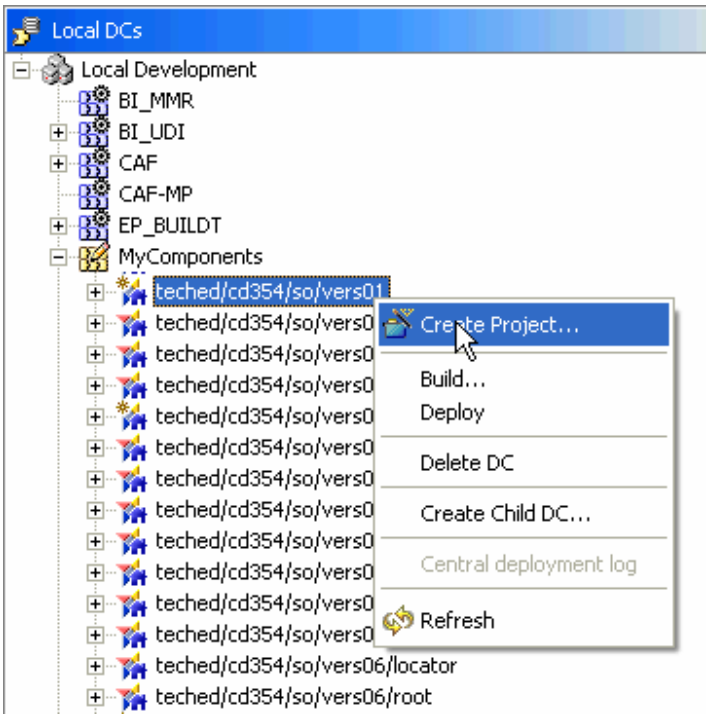
Appendix A – Opening Exercise Web Dynpro DCs in the Development Environment

1. In the toolbar on the left side select the button  on top for opening a new perspective.
2. Select the list item *Development Configurations*. If this item is not listed select the list item *Other ...* for opening the *Development Configurations* perspective in the next popup window.





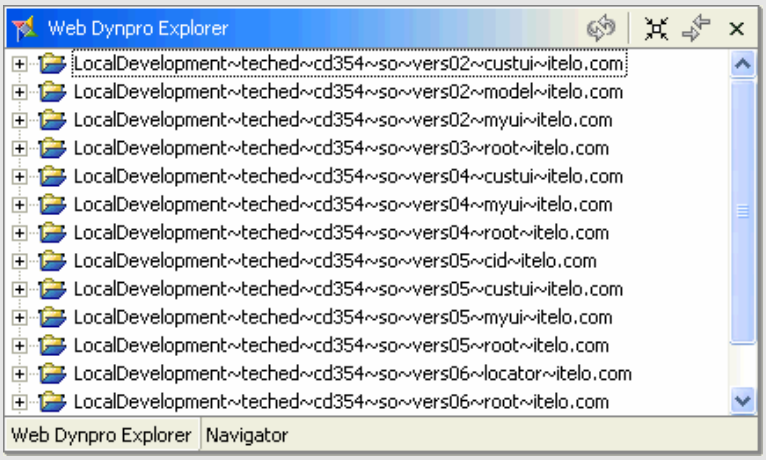
3. Select the view ***Local DCs*** in the new perspective.
4. Select the node *Local Development* → *MyComponents.*
5. Select all local development components starting with *teched/cd354/so*.
6. Open the context menu and select the menu item *Create Project …*



7. Confirm the popup dialog *Confirm Development Components* by pressing *Ok*.
8. The developer studio now starts to sync and build all selected development components. This process will take two minutes.

Result

All Web Dynpro Development Components on *Web Dynpro Component Interface Definitions in Practice* are now listed inside the Explorer of the Web Dynpro perspective.



Appendix B – Complete Custom Controller Code

Model Component Controller – ModelComp.java

```
//@@begin javadoc:wdDoInit()
/** Hook method called to initialize controller. */
//@@end
public void wdDoInit(){
  //@@begin wdDoInit()
  wdContext.nodeSalesOpportunities().bind(opportunityModel.getOpportunities());
  //@@end
}
//@@begin others
// ======= PRIVATE MEMBERS ============================
private SalesOpportunityModel opportunityModel = new SalesOpportunityModel();
//@@end
```

Root Component Controller – RootComp.java

```
//@@begin javadoc:wdDoInit()
/** Hook method called to initialize controller. */
//@@end
public void wdDoInit()  {
  //@@begin wdDoInit()
  wdThis.createUICompInstance();
  //@@end
```



```
}
//@@begin javadoc:createUICompInstance()
/** Declared method. */
//@@end
public void createUICompInstance( ) {
  //@@begin createUICompInstance()
  wdThis.wdGetISalesOppUIComponentUsage().createComponent(
    wdContext.currentConfiguredUICompImplElement().getComponentName(),
    wdContext.currentConfiguredUICompImplElement().getDCName());
  //@@end
}
```

Locator Component Controller – LocatorComp.java

```
//@@begin javadoc:wdDoInit()
/** Hook method called to initialize controller. */
//@@end
public void wdDoInit()  {
  //@@begin wdDoInit()
  wdThis.readConfiguration();
  //@@end
}
//@@begin javadoc:readConfiguration()
/** Declared method. */
//@@end
public void readConfiguration( )  {
  //@@begin readConfiguration()
  try {
    // Load configuration file which is deployed with the Locator Component.
    IWDConfiguration configuration = WDConfiguration.getConfigurationByName(
        wdComponentAPI.getDeployableObjectPart(), "uiCompImpl");
   // Read configuration data defining the fully qulaified
   // name of the configured UI Component implementation.
    wdContext.currentConfiguredUICompImplElement().setDCName(
      configuration.getStringEntry(ConfigKey.DC_NAME));
    wdContext.currentConfiguredUICompImplElement()
      .setComponentName( configuration.getStringEntry(ConfigKey.COMP_NAME));
  } catch (WDInvalidConfigParameterException e) {
    wdComponentAPI.getMessageManager().reportException(e.getLocalizedMessage(), true);
  } catch (WDConfigurationNotFoundException e) {
    wdComponentAPI.getMessageManager().reportException(e.getLocalizedMessage(), true);
  }
  //@@end
 }
```


Copyright