Creating and Integrating Custom Controls in Mobile Applications for Handhelds

Applies to:

SAP NetWeaver Mobile 7.1 For more information, visit the Mobile homepage.

Summary

This document tells about how to customize the look and feel of embedded Standard Widget Toolkit (eSWT) controls and use it in mobile applications for handhelds.

Author: Dhanya K Moni

Company: SAP Labs India Pvt. Ltd.

Created on: 14 December 2009

Table of Contents

Introduction

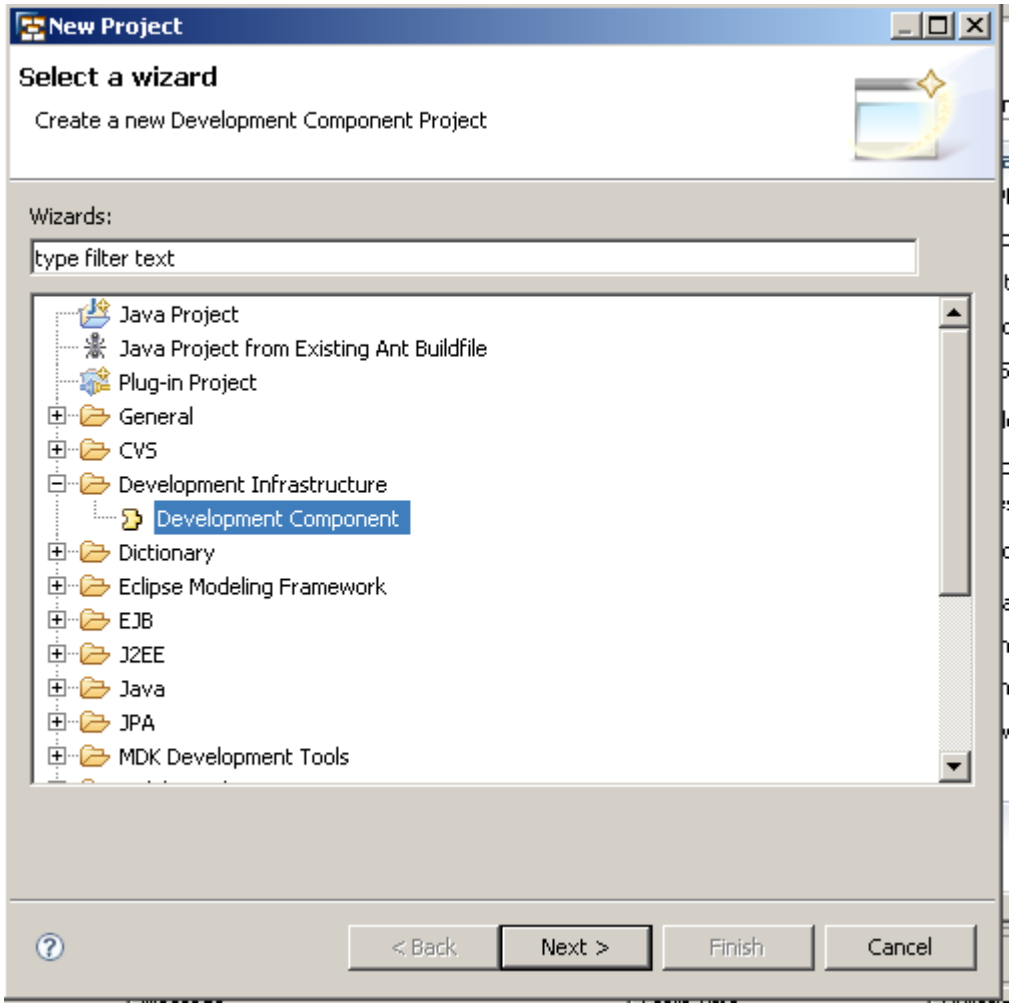
Many times you might be having a requirement of changing the look and feel of existing embedded Standard Widget Toolkit (eSWT) controls. SAP NetWeaver Mobile for handhelds provide such a facility to the application developers to customize eSWT controls by changing the look and feel of these controls, and add them to a mobile application for handhelds. These controls can be embedded in the standard SAP widget library. There are three pieces required to integrate a custom into the Mobile Client.

- A working native control.
- IUIElementFactory: To instantiate and configure the control during runtime.
- UiElementMethodCallAdapter: To interact with the Mobile Client. For example, Context Binding

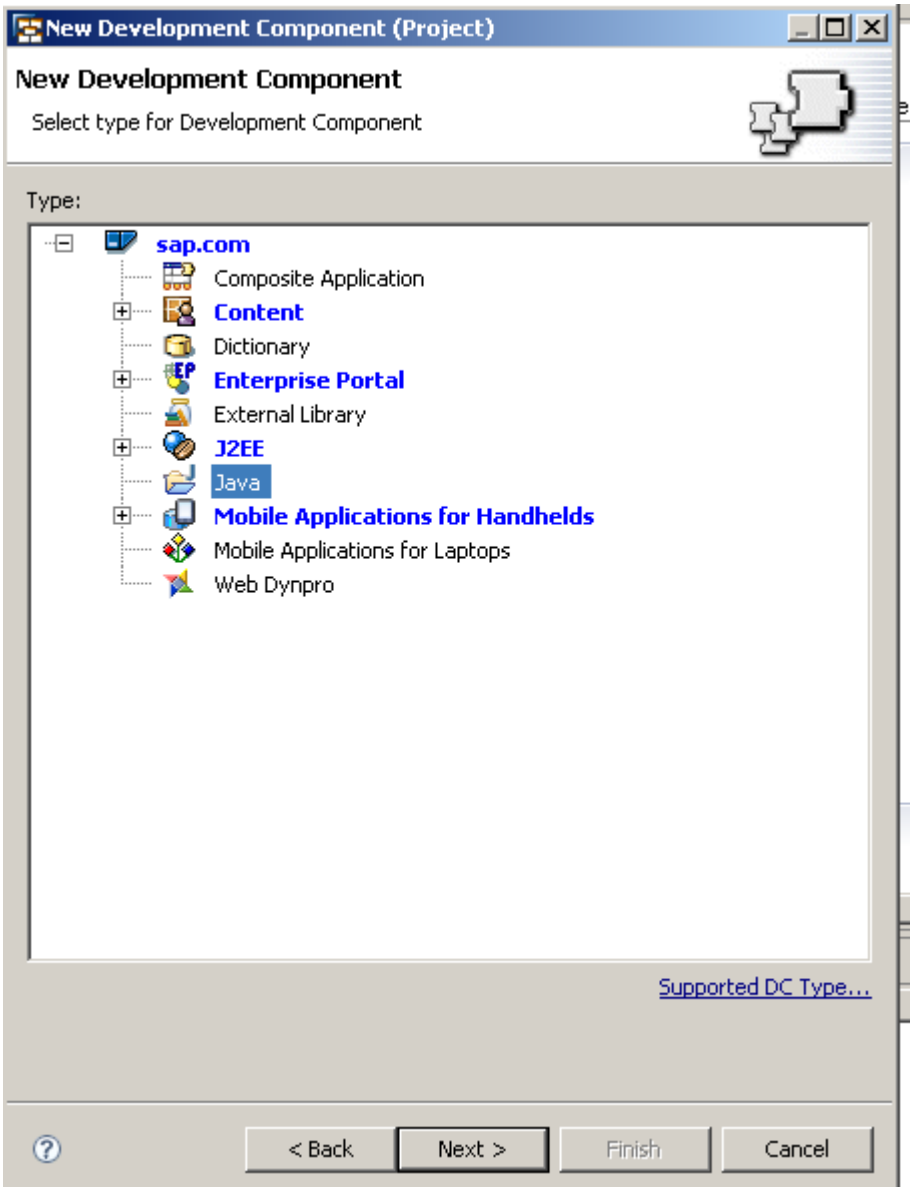
Creating a simple SWT custom control

The following procedure is an example for creating a button using eSWT (Embedded Standard Widget Toolkit).

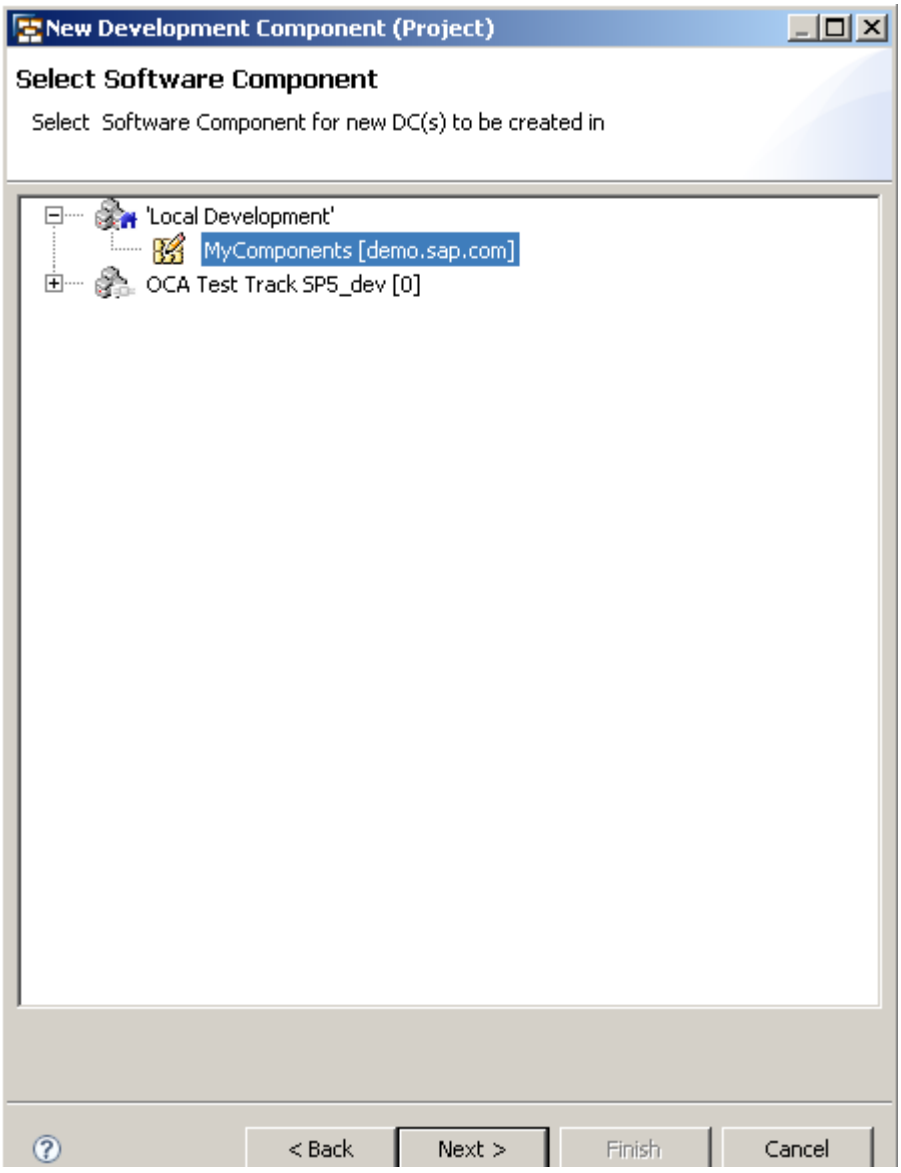
1. In the Mobile Application for Handhelds perspective of SAP NetWeaver Development Studio, click the *New* icon and choose *Project* to start the *New Project* wizard.
2. Expand the *Development Infrastructure* node and choose *Development Component*.



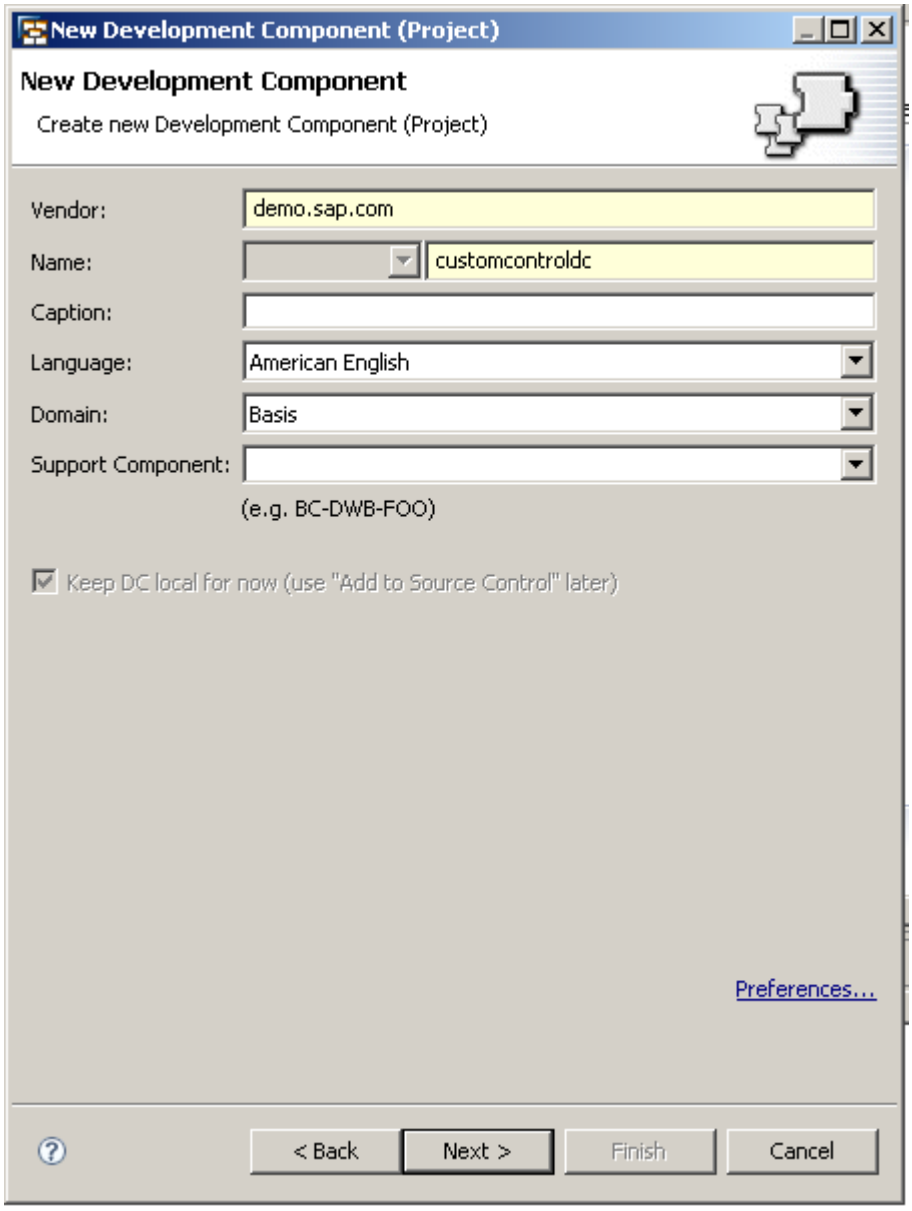
3. Choose Java from the *New Development Component* Wizard.



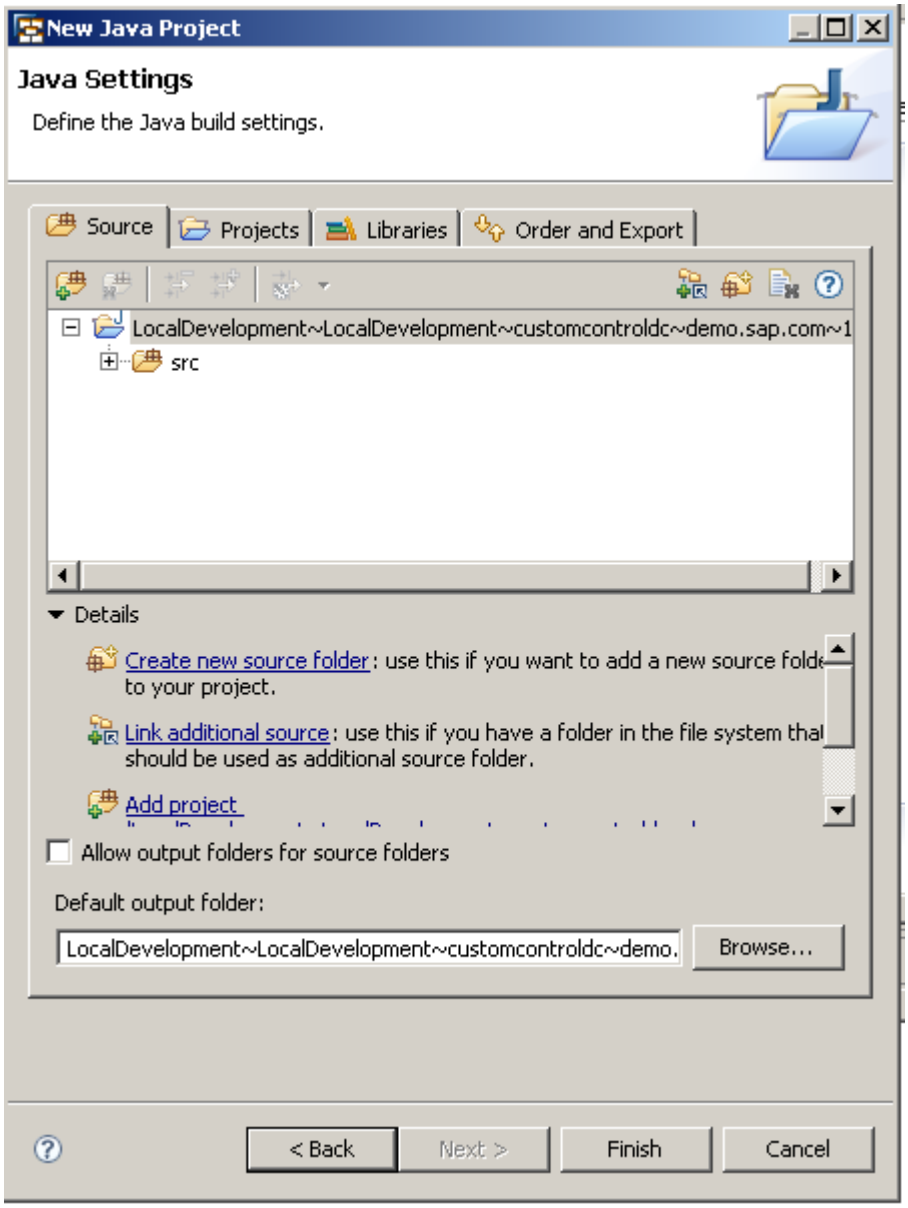
4. Expand the *Local Development* node and choose *MyComponents*.



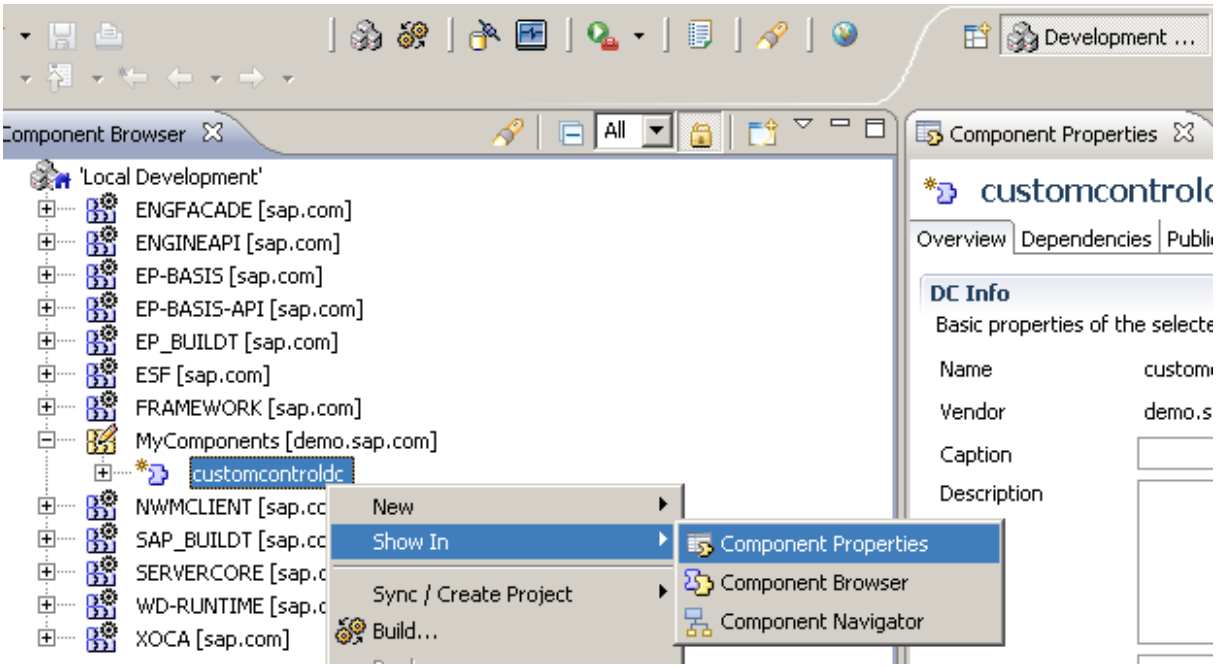
5. Enter the Java Development Component *Name*.



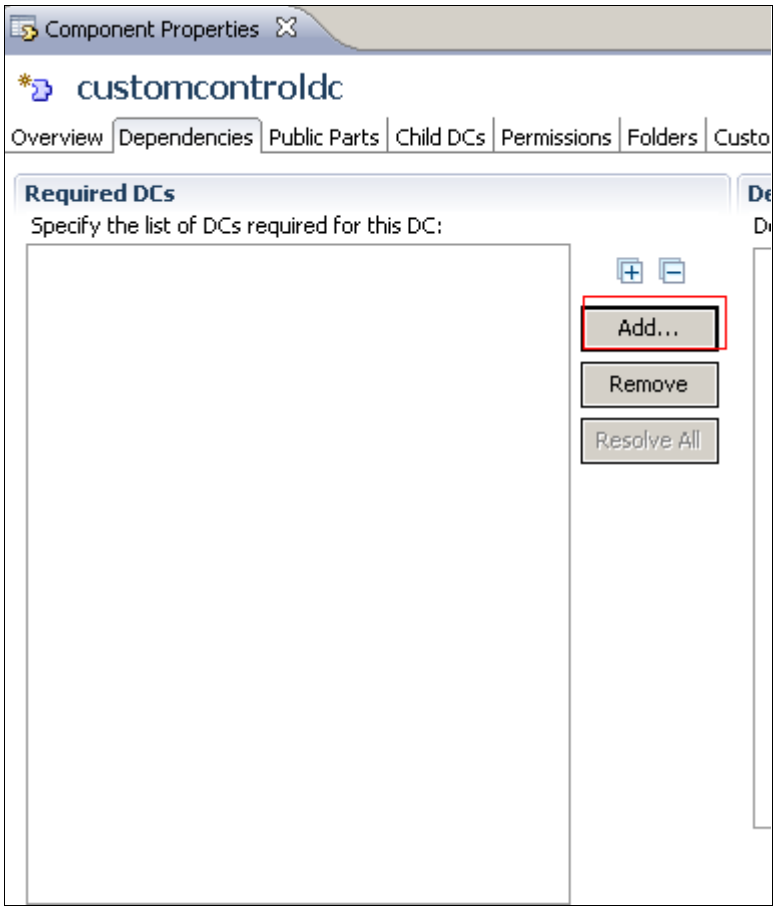
6. Click Next to view the *New Java Project* Wizard.



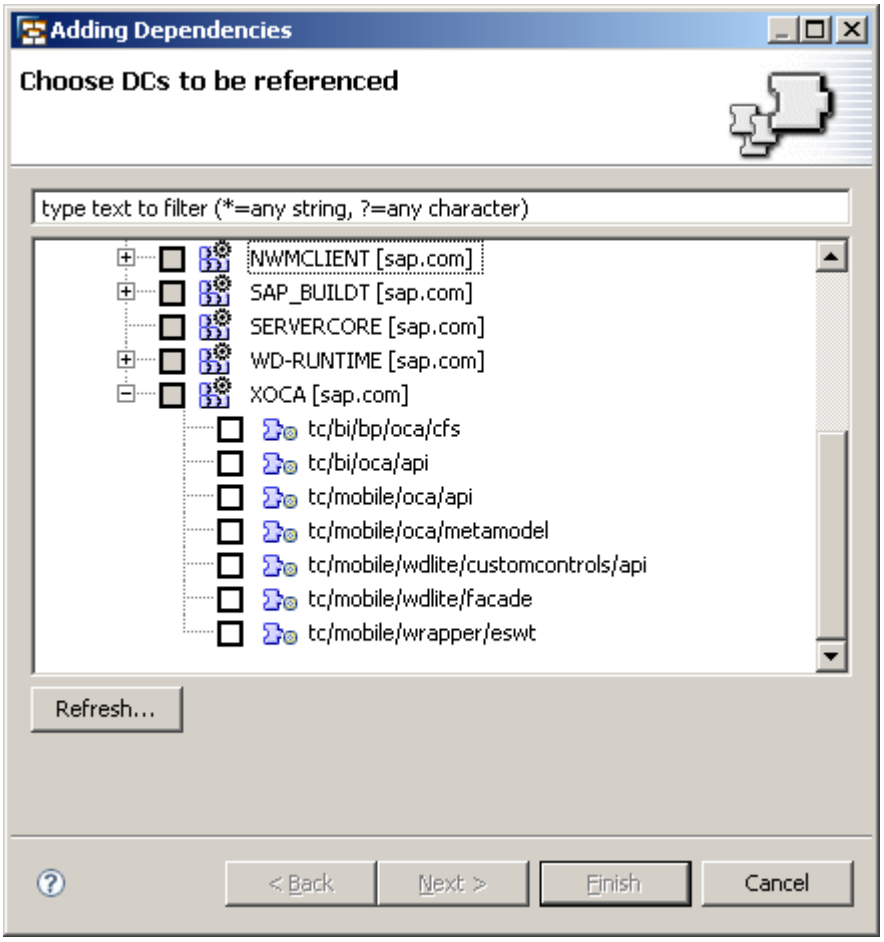
7. Choose Finish.
8. Go to Window -> Open Perspective and choose Development Infrastructure perspective.
9. From the Component Browser select the customcontroldc. Open the context menu of the dc and choose, Show In -> Component Properties



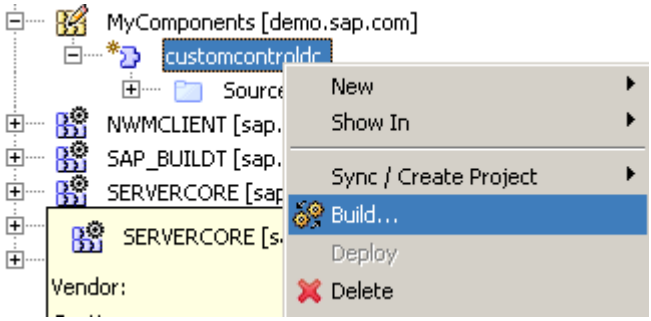
10. In the Component Properties view, open the *Dependencies* tab and choose *Add* to add dependencies to the Java component.



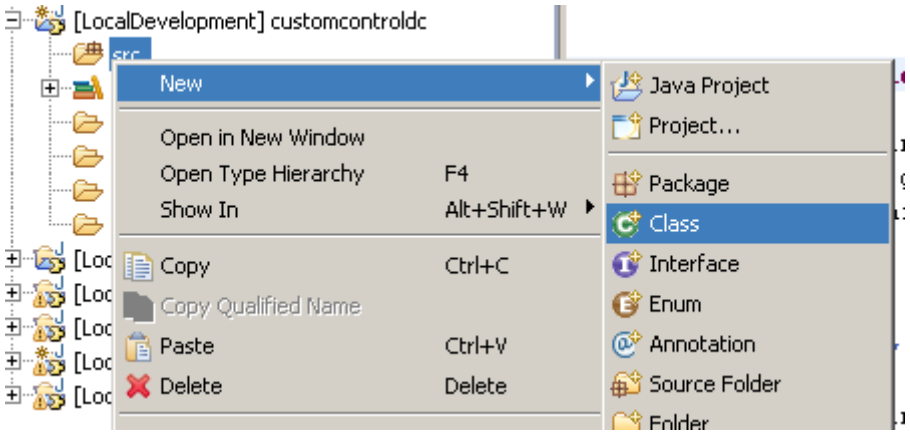
11. In the *Adding Dependencies* wizard, expand the *XOCA* node.



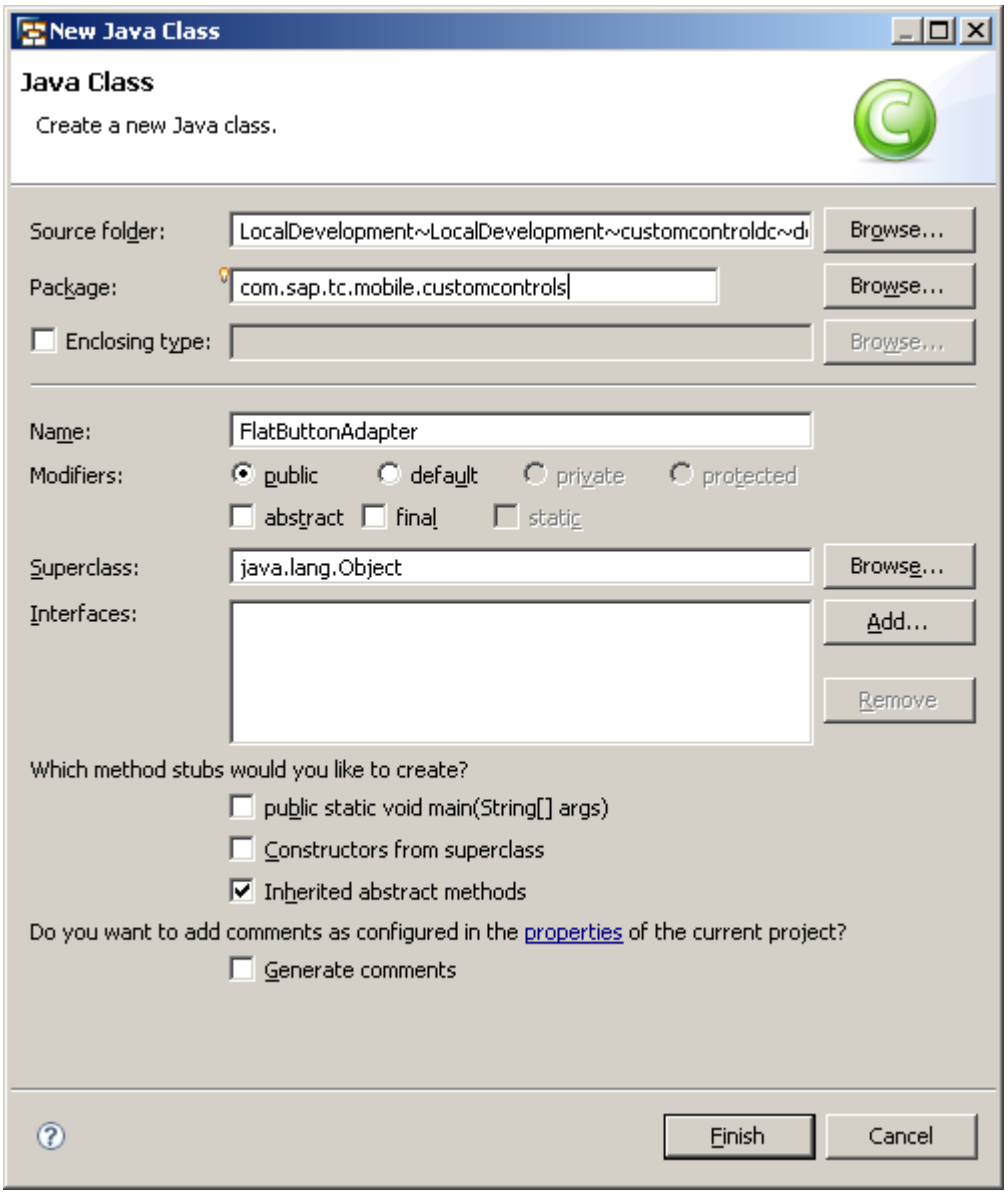
12. Select the following dependencies from the XOCA node and click *Finish* button.
    - tc/mobile/wdlite/customcontrols/api
    - tc/mobile/oca/api
    - tc/mobile/wrapper/eswt
13. From the context menu of the development component, choose *Build*.



14. Open the Java perspective and expand the custom control node and choose *src*. From the context menu choose, New->Class.

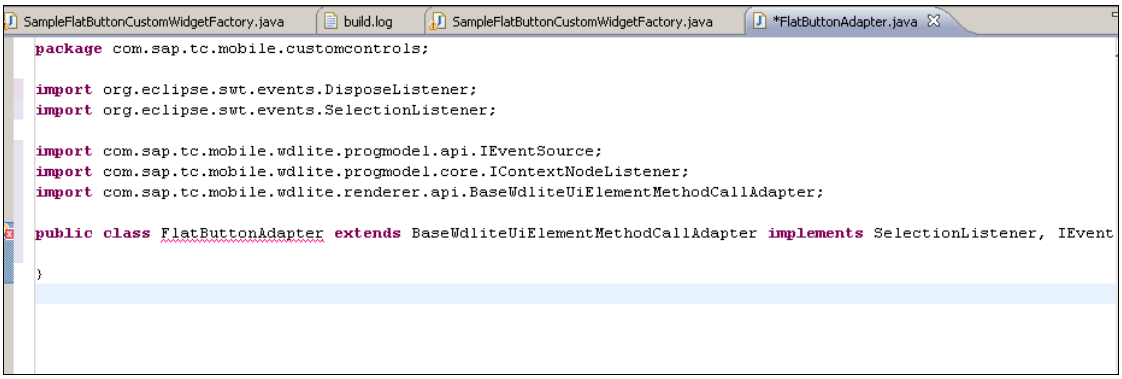


15. Enter the Java Class “FlatButtonAdapter” and Package “com.sap.tc.mobile.customcontrols”. Click *Finish* button.



16. To interact with the Web Dynpro framework, open the java file, extend `BaseWdliteUiElementMethodCallAdapter` class and implement the following interfaces
    - SelectionListener
    - IEventSource
    - DisposeListener
    - IContextNodeListener

```
public class FlatButtonAdapter extends BaseWdliteUiElementMethodCallAdapter
implements  SelectionListener,IEventSource,DisposeListener,IContextNodeListener {
}
```



17. Add the following unimplemented methods.

```
@Override
public void widgetDefaultSelected(SelectionEvent arg0) {
    // TODO Auto-generated method stub

}

@Override
public void widgetSelected(SelectionEvent arg0) {
    // TODO Auto-generated method stub

}

@Override
public void widgetDisposed(DisposeEvent arg0) {
    // TODO Auto-generated method stub

}

@Override
public void listenDomainContextNode(ContextNode contextNode) {
    // TODO Auto-generated method stub

}
```

18. Create variables of the following Type:

```
private final Button button;
private UiGenericElement wdliteUiGenericElement;
private IWDNode node;
private String buttonProb = "";
```

19. Create the following constructor:

```
public FlatButtonAdaptor(Button b){
        this.button = b;
        b.addSelectionListener(this);
        b.addDisposeListener(this);
}
```

20. Create a public method *onPropertyChange (String name, Object value)* which is called when the Property changes and add the following code.

```
        if ("textProp".equals (name) && value != null) {
                button.setText(value.toString());
                buttonProb = value.toString();
        }
```

Create a public method *setUiGenericElement (UiGenericElement wdliteUiGenericElement)*and add following code:

```
this.wdliteUiGenericElement = wdliteUiGenericElement;
```

21. Create a public method *setContextNode (IWDNode node)* and add following code. This method stores the context node.

```
        if (node == null)
                return;
this.node = node;
```

22. Create a public method *setText (String text)* and add following code. This method sets the Text.

```
button.setText(text);
```

23. In the method *widgetSelected* add the following code .This method translates the native SWT event into a wdlite event and forwards it.

```
UIEvent wdliteUiEvent = new UIEvent(this, IEvent.EVENT_ONACTION);
//add any data to the event, can be accessed by application developer
wdliteUiEvent.setEventData("someEventData", "yourcustomData");
//allows to distinguish between multiple events of a custom control
wdliteUiEvent.setEventData("eventType", "btnClick");
wdliteUiGenericElement.doHandleCustomControlEvent(wdliteUiEvent);
wdliteUiGenericElement.updateProperty("textProp", buttonProb + "updated");
```

24. In the method *widgetDisposed* add the following code:

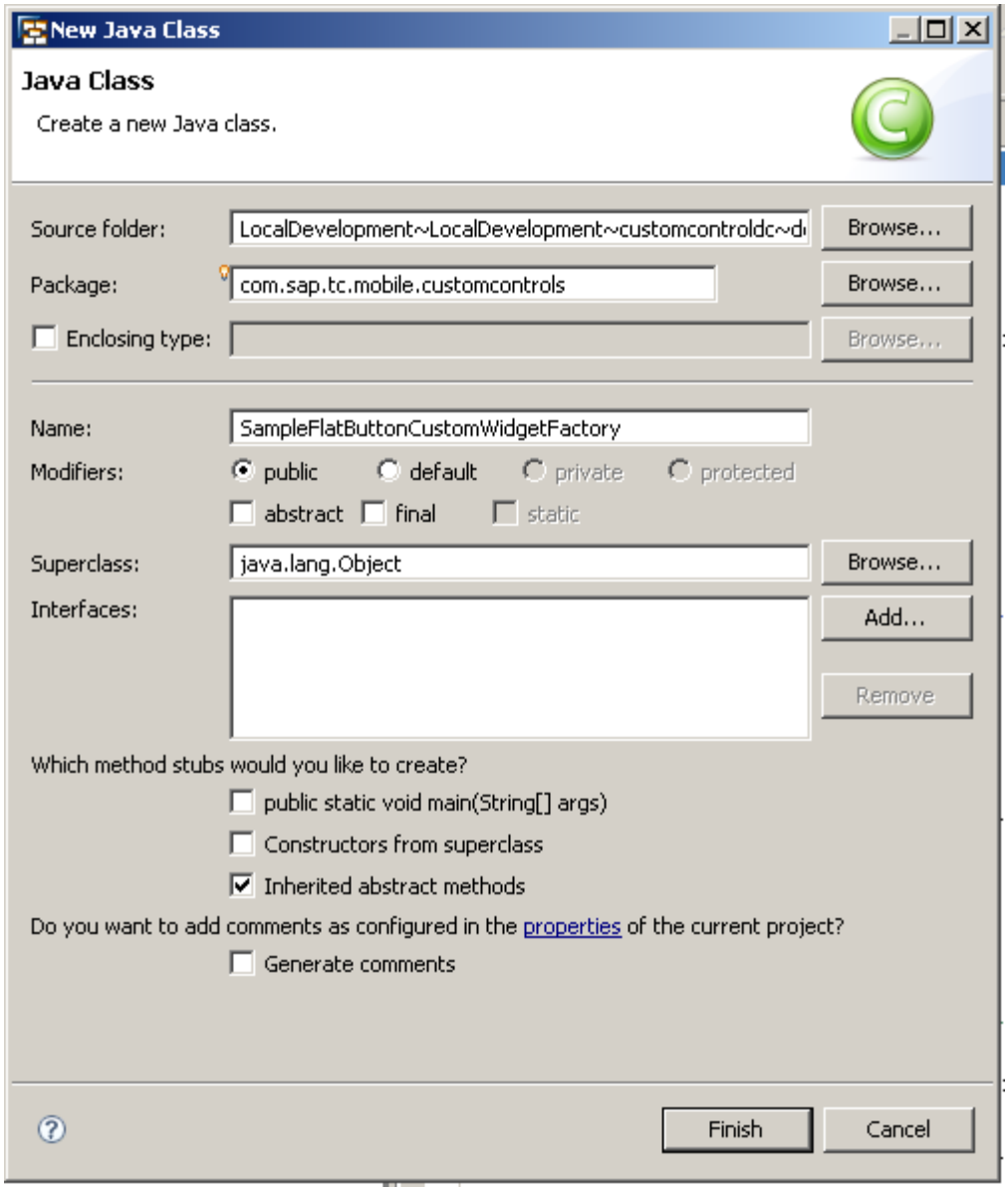
```
        if (node == null) return;
        ContextNode n = (ContextNode) node;
n.removeContextNodeListener (this);
```

25. In the method *listenDomainContextNode* add the following code:

```
        if (node.isEmpty()) return;
                node.getElementAt(1).getNode().getAttributes();
        for (Iterator iterator = node.getElementIterator();
        iterator.hasNext();) {
                IWDNodeElement next = (IWDNodeElement) iterator.next();
```

```
                    Object attributeValue = next.getAttributeValue("Name");
    }
```

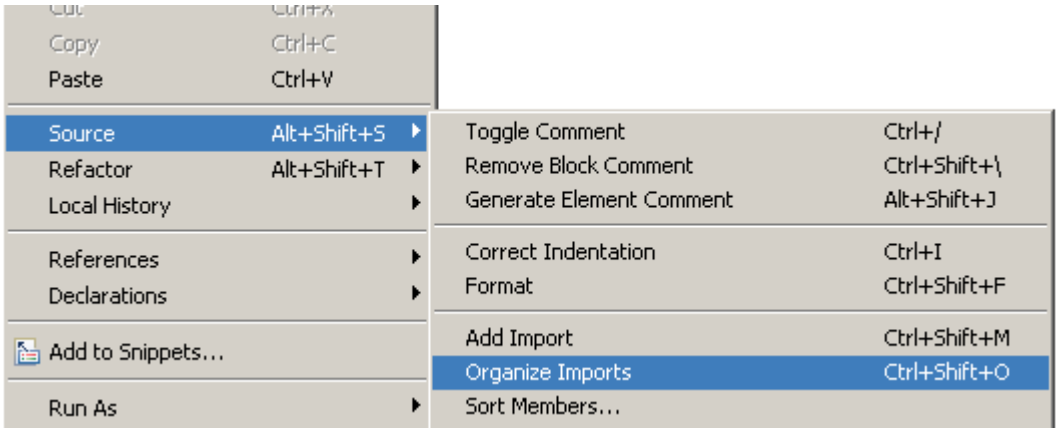
26. Repeat step 14 to create another Java class.
27. Enter the Java Class name “SampleFlatButtonCustomWidgetFactory” and Package name “com.sap.tc.mobile.customcontrols”.



28. To instantiate and configure the control during runtime, open the java file and implement the *IUIElementFactory* and add the unimplemented methods.

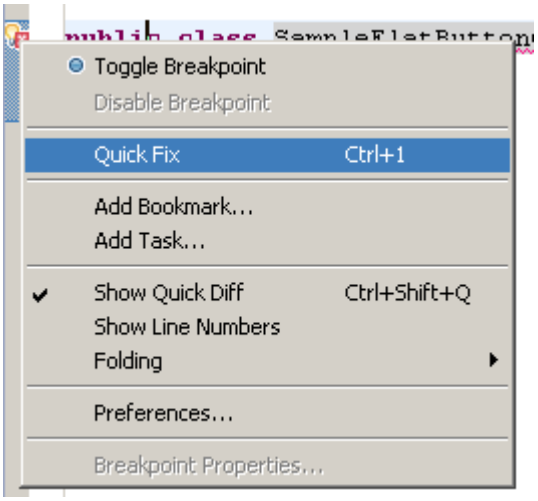
```
public class SampleFlatButtonCustomWidgetFactory implements IUIElementFactory
```

29. Navigate to the Package Explorer of the custom control.
30. From the context menu, choose Source -> Organize Imports.



31. To fix error in the application code, you can use the Quick Fix option. Choose Edit->Quick Fix from the main menu.

    Alternatively you can select the quick fix icon on the left side of the java editor, and from the context menu choose Quick Fix. Select *Add unimplemented methods* and save the file.



32. Create a private method called *getSwtCompositeForWdliteContainer* `(UIContainer parent)` and set the return type as `Composite`.

```
return SWTUtils.getSwtComposite(parent);
```

SampleFlatButtonCustomWidgetFactory.java

```
package com.sap.tc.mobile.customcontrols;
import java.util.Map;

import org.eclipse.swt.widgets.Composite;

import com.sap.tc.mobile.wdlite.renderer.api.IUIElementFactory;
import com.sap.tc.mobile.wdlite.renderer.api.UIContainer;
import com.sap.tc.mobile.wdlite.renderer.api.UIElement;
import com.sap.tc.mobile.wdlite.renderer.swt.SWTUtils;

public class SampleFlatButtonCustomWidgetFactory implements IUIElementFactory{
    private Composite getSwtCompositeForWdliteContainer (UIContainer parent){
        return SWTUtils.getSwtComposite(parent);
    }
    @Override
    public UIElement createNewInstance(int type, String subType,
            UIContainer container, Map style) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public UIElement createNewInstance(int type, String subType,
            UIContainer container, Map style, String glyph) {
        // TODO Auto-generated method stub
        return null;
    }
}
```

33. In the *createNewInstance (int type, String subType,UIContainer container, Map style)* method add the following codes(till step 37):

```
Button button = new Button(getSwtCompositeForWdliteContainer(container),
                    SWT.BORDER);
```

You need to create the native control, specify the parent container and apply the style parameters.

34. You need to provide the implementation of an iElementMethodCallAdapter for the custom control. It contains all code for interaction between the custom control and the WebDynpro Lite framework.

```
FlatButtonAdapter flatButtonAdapter = new FlatButtonAdapter (button);
```

35. You need to add the UiGenericElement which the Mobile Client uses to communicate with the custom control. This UiGenericElement needs a UiElementMethodCallAdapter as argument.

```
UiGenericElement wdliteElement = new
UiGenericElement(flatButtonAdapter);
flatButtonAdapter.setUiGenericElement(wdliteElement);
```

36. The GenericSwtControlWrapper is needed for the SWT abstraction of the Mobile Client. This is used to store the control, listen to events and forward them.

```
GenericSwtControlWrapper wrappedSwtControl = new
GenericSwtControlWrapper(container, wdliteElement, button);
wdliteElement.setUiWidgetAndParent(wrappedSwtControl, container);
return wdliteElement;
```

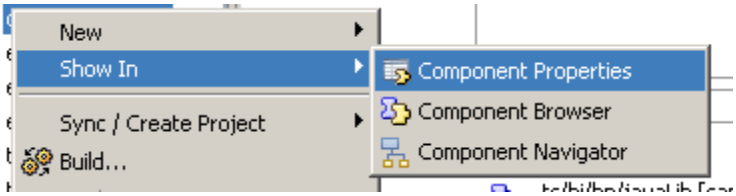
37. In the UI Element `createNewInstance(int type, String subType, UIContainer container, Map style, String glyph)` add the following code:

```
return createNewInstance(type, subType, container, style);
```

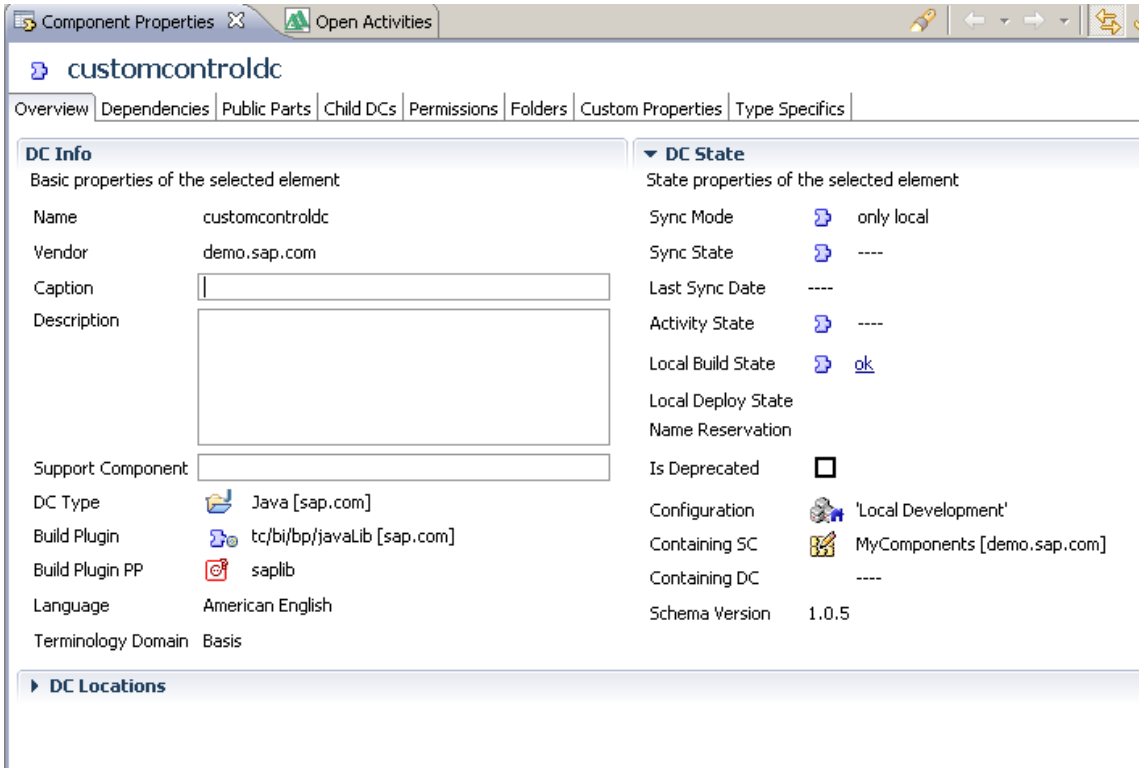
Integrating Custom control into mobile UI components

The custom control has to be referenced by another development component such as the Mobile UI Component or Mobile application component. By adding a dependency to the custom control you can integrate it with the Mobile UI Component.

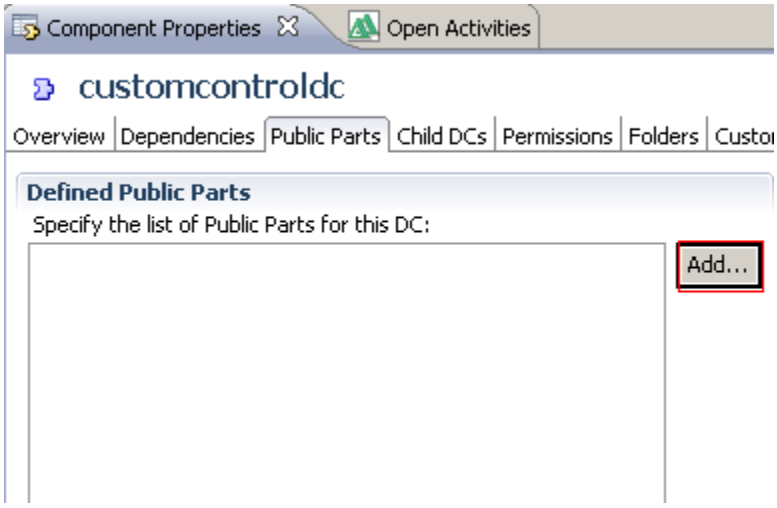
1. In the *Development Infrastructure* perspective, select the custom control.
2. From the context menu, choose *Show In -> Component Properties*.



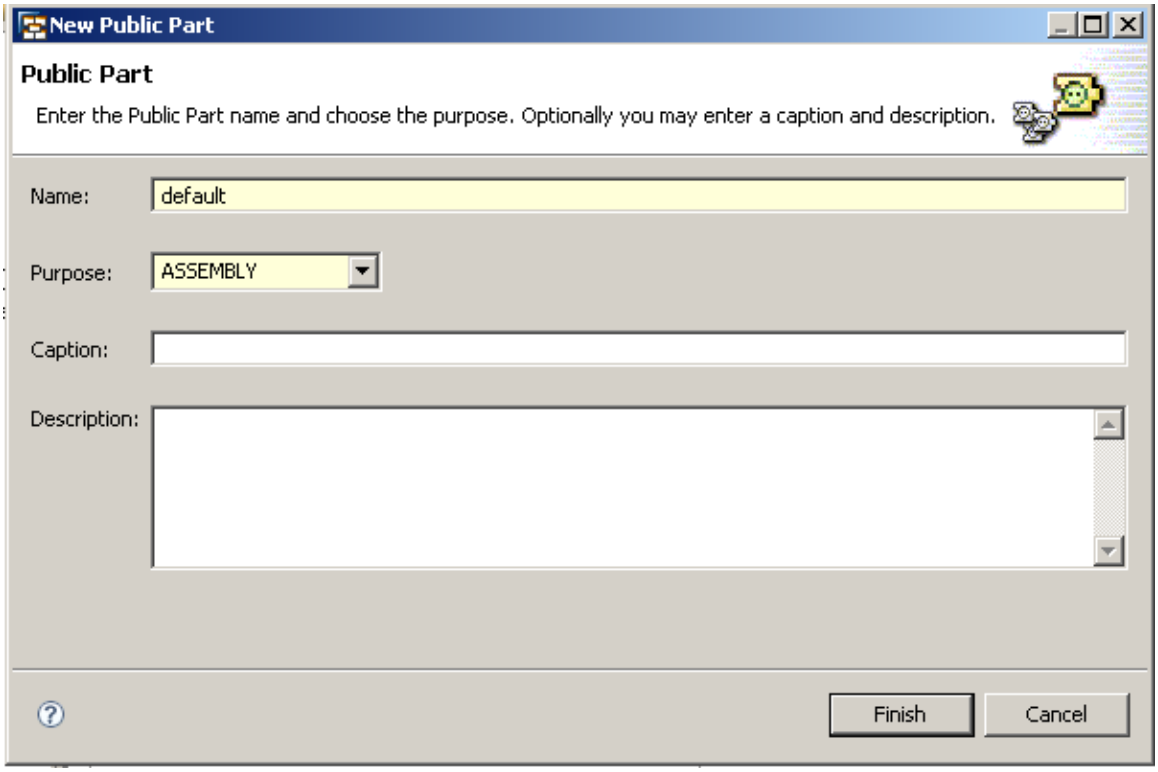
3. Navigate to the *Public Parts* tab page.



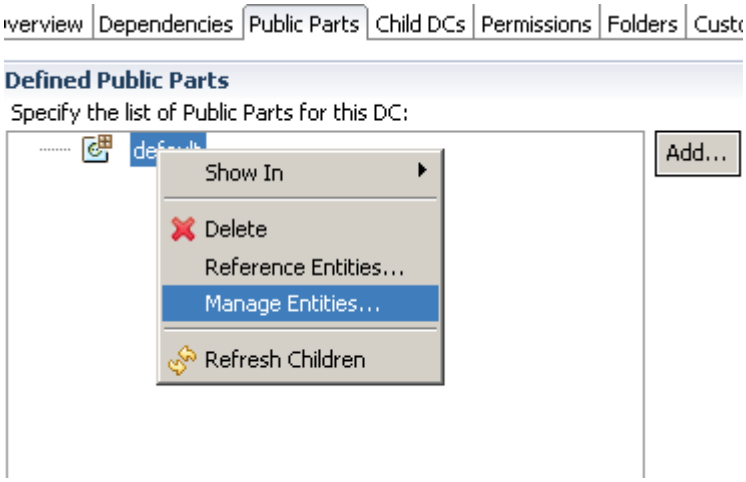
4. Open the *Public Parts* tab and choose *Add* to add a new public part.



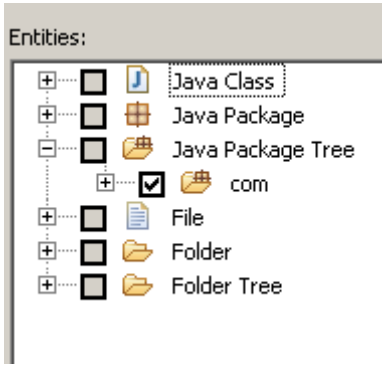
5. Enter the *Name* of the Public Part and choose the *Purpose*.



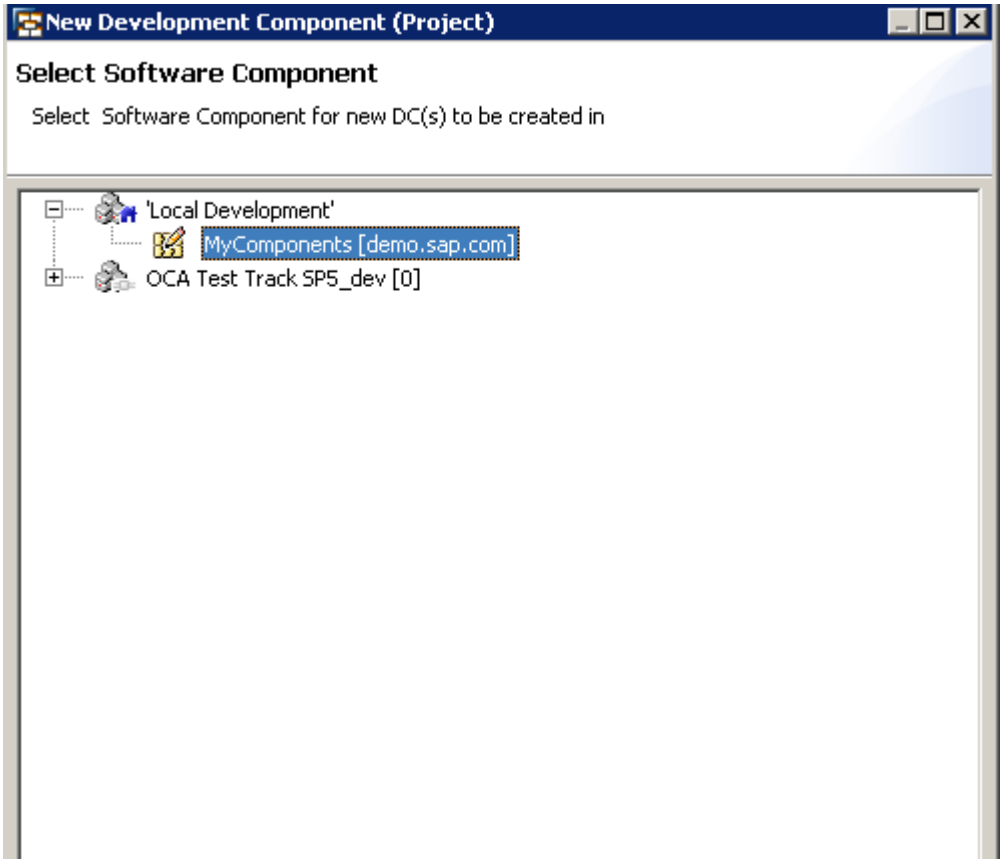
6. From the context menu of the newly created public part, choose *Manage Entities*.



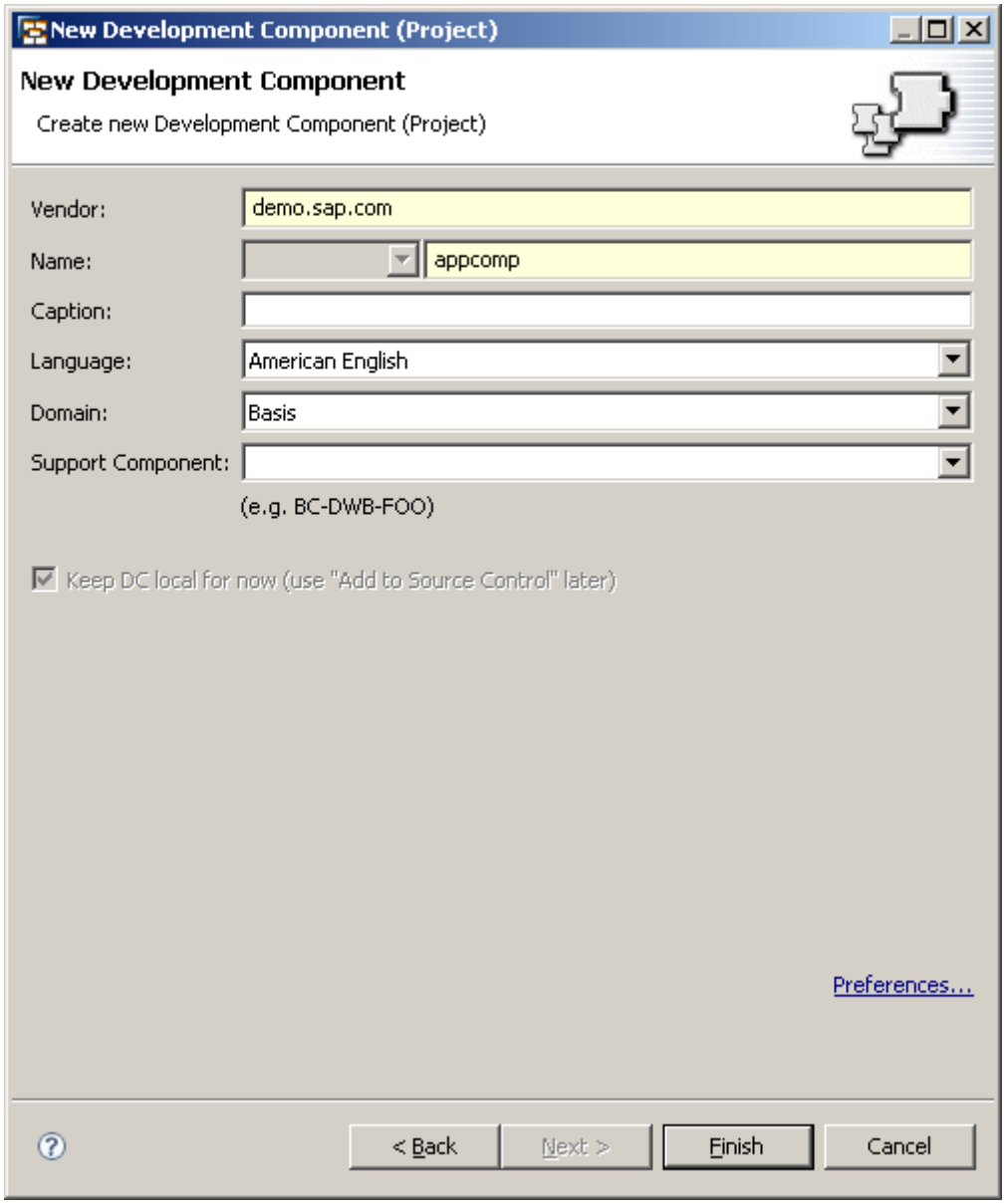
7. Expand the *Java Package Tree* node and select the *com* node.



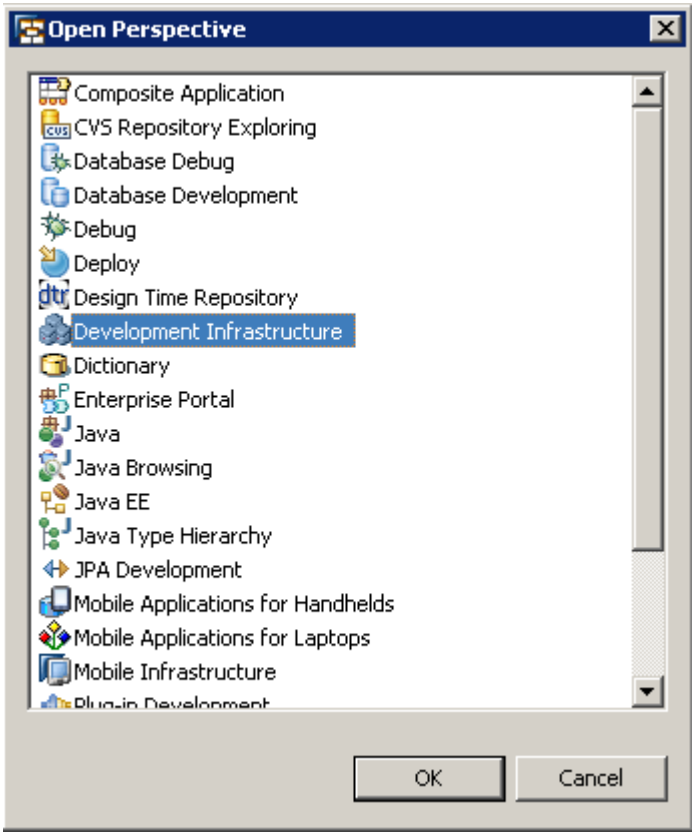
8. Choose Finish.
9. From the context menu of the custom control, choose Development Component -> Build.
10. Create a new Application component by selecting the *New* icon.
11. From the *New* wizard, expand the Mobile Applications node and choose Mobile Application.
12. In the *New Development Component* wizard, expand the *LocalDevelopment* node.



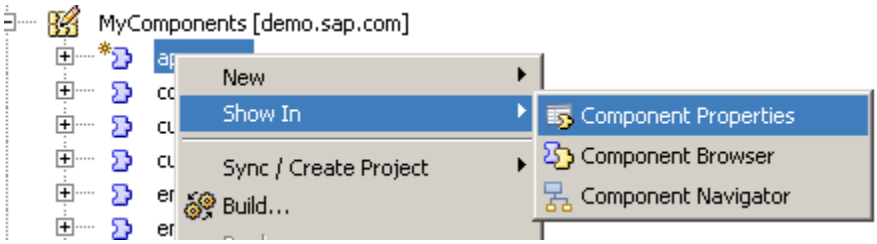
13. Select *MyComponent* node.
14. Enter the *Name* of the component.



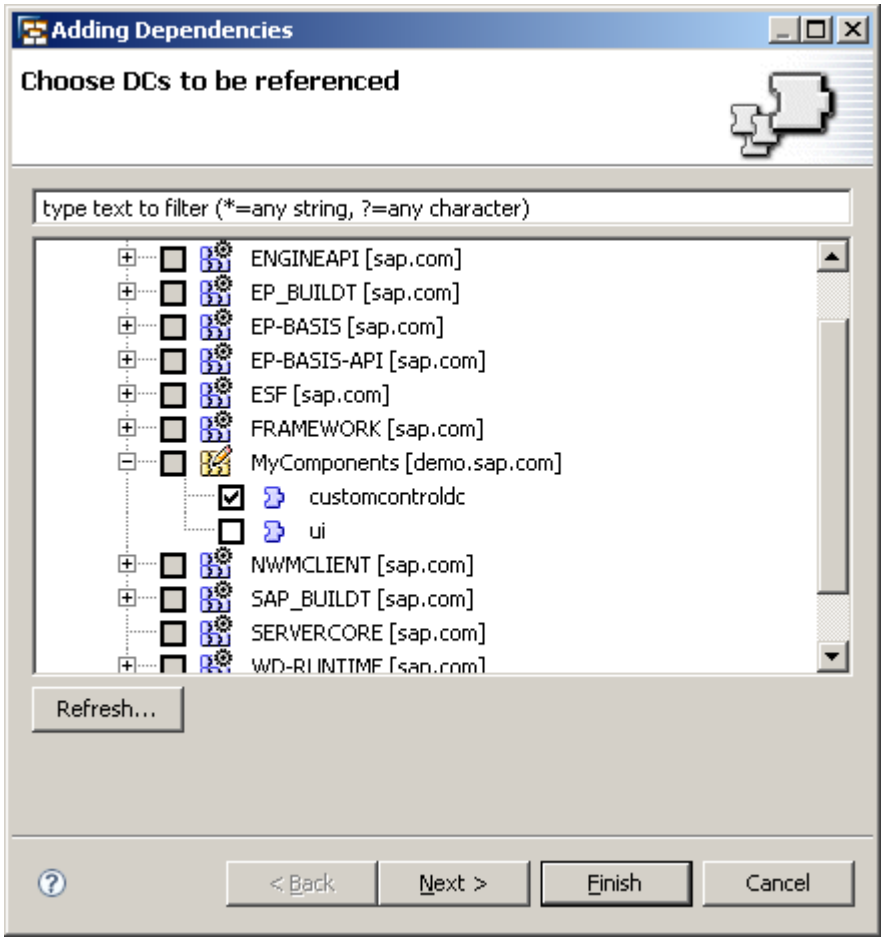
15. Navigate to the Development Infrastructure perspective by choosing, *Window->Open Perspective->Other->Development Infrastructure.*



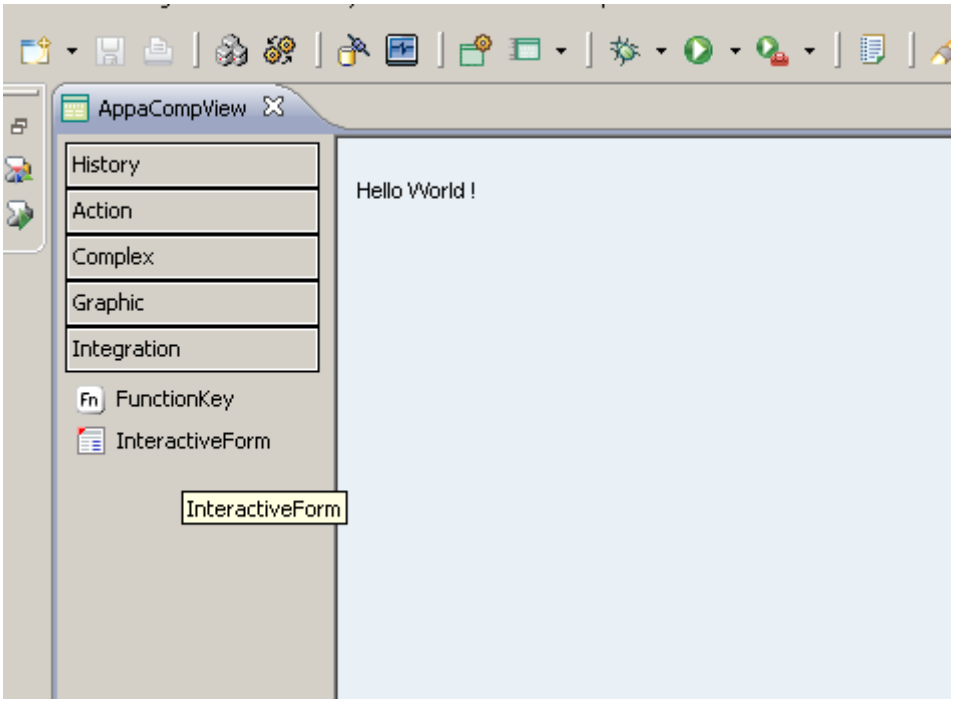
16. From the Component Browser, expand the MyComponents node.
17. Select the application component.
18. From the context menu, choose Show In-> Component Properties.



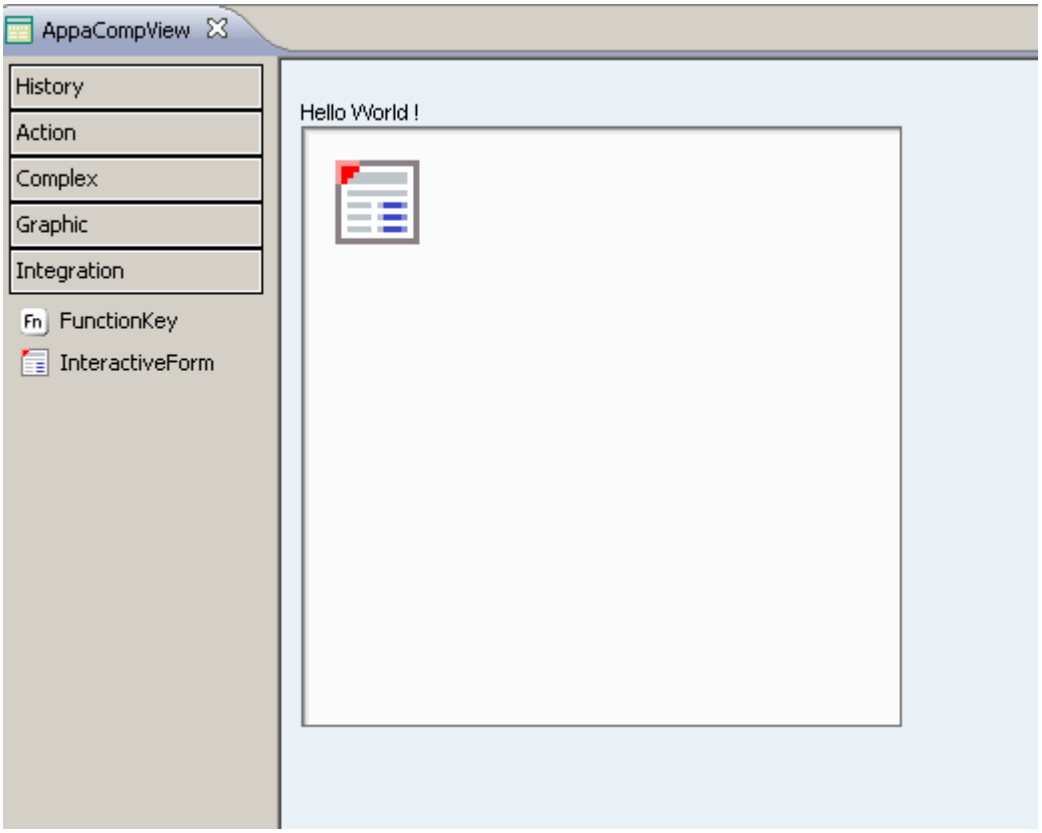
19. From the *Dependencies* tab, Choose Add.
20. From the *Adding Dependencies* wizard, expand the MyComponent node.



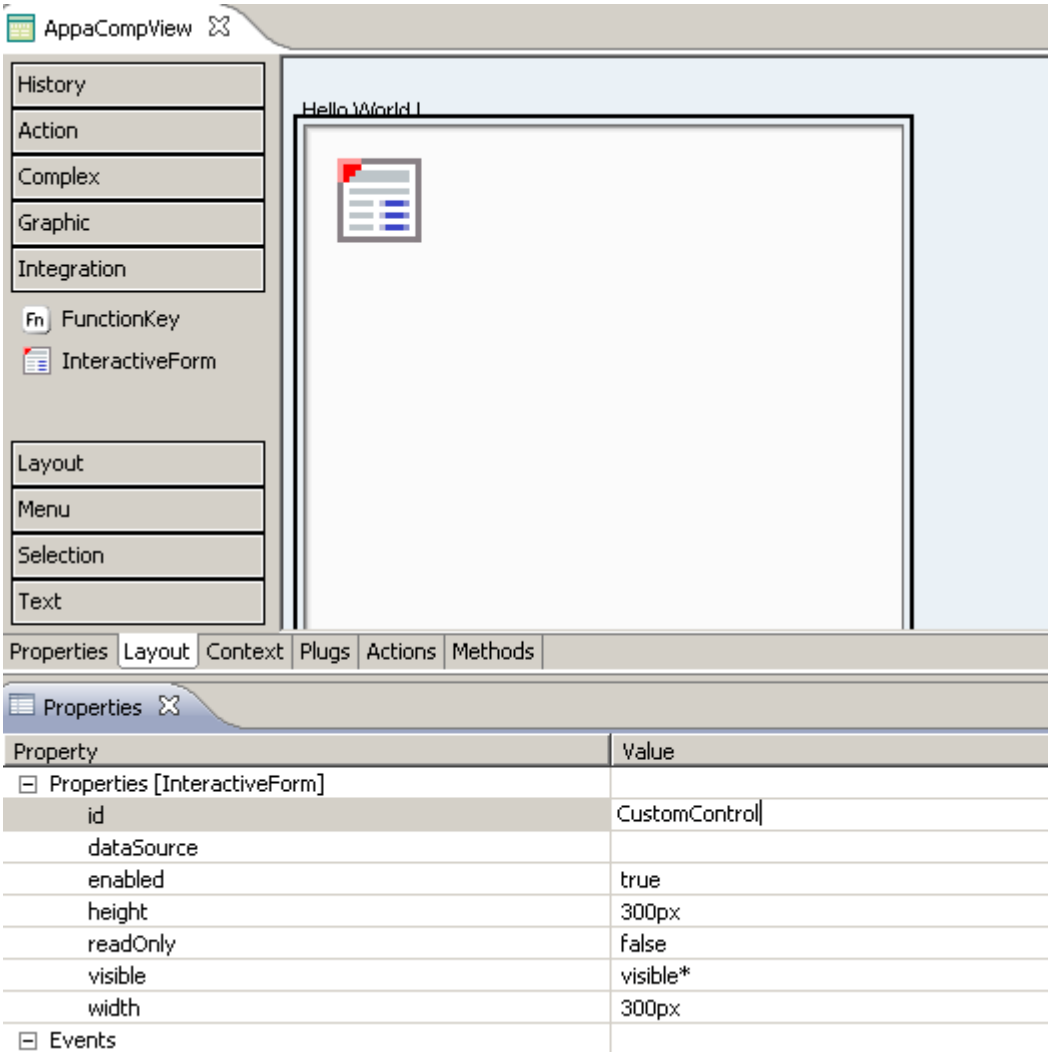
21. Select the custom control to be added as a dependency.
22. Create a new Mobile UI component and an application.
23. Expand the Components->View node and select the corresponding *View*.
24. In the Layout tab page, choose the Integration menu.



25. Drag and drop the *InteractiveForm* control into the layout.



26. In the properties View of the “InteractiveForm” UI element, change the *id* to *CustomControl*.

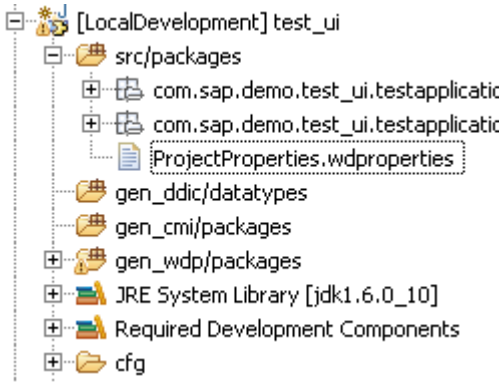


27. Choose File->Save.
28. Open the Java Editor of the View and add the following code in `wdDoModifyView()` method:

```
view.getCustomControlViewItem().setUIControlSubType("Test");
```

The parameter passed to the setUIControlSubType should be same as the Wdlite-custom-control-factory-ids-1 key’s value specified in the custom_controls.properties. More details are given in the below steps.

29. In the Java Perspective, navigate to the `/src/packages` folder.



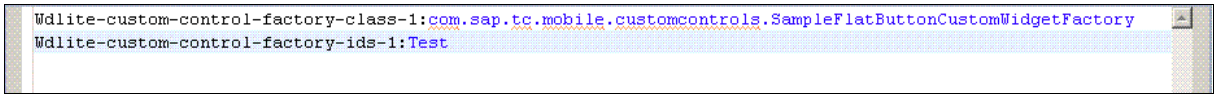
30. Create a file called custom_controls.properties and copy the below lines into it. Copy the file into the src/packages folder of the ui component.

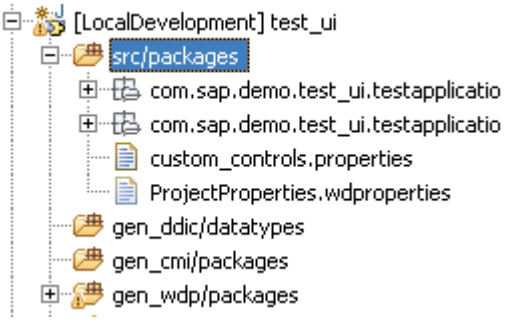
    Wdlite-custom-control-factory-class-1:
    com.sap.tc.mobile.customcontrols.SampleFlatButtonCustomWidgetFactory

    Wdlite-custom-control-factory-ids-1: Test

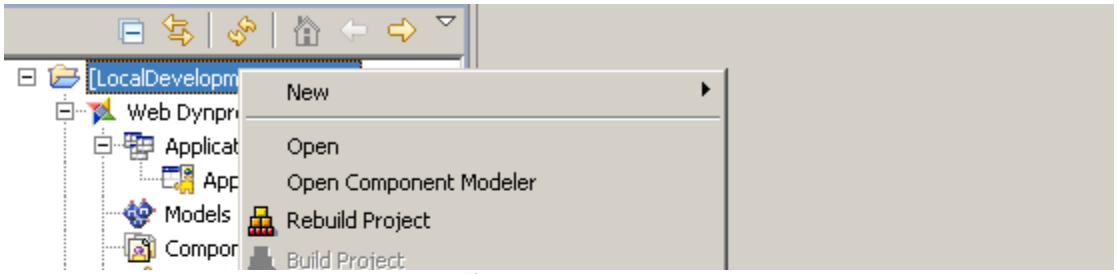
    The custom_controls.properties file contains two keys one is

    - **Wdlite-custom-control-factory-class-1 ->** This key's value is the class name of custom control which implements **IUIElementFactory** interface. In this example, this corresponds to **com.sap.tc.mobile.customcontrols.SampleFlatButtonCustomWidgetFactory.** This class is used for creating the custom controls.
    - **Wdlite-custom-control-factory-ids-1 ->** This key's value corresponds to the id of the custom Control.



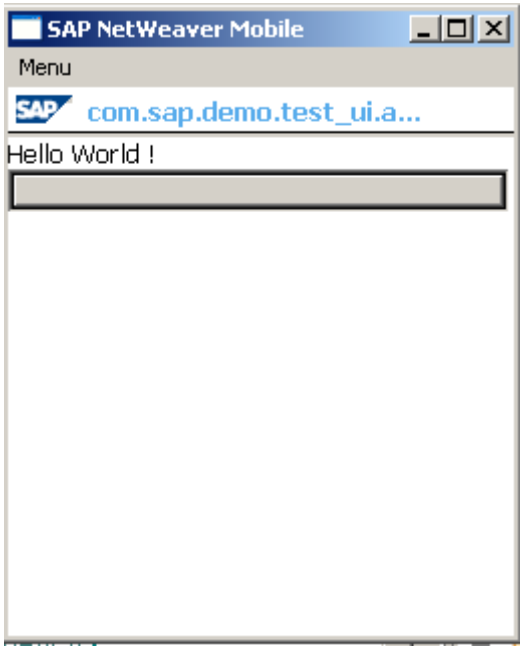


31. In the *Mobile Applications for Handhelds* Perspective, navigate to the *Handhelds UI Explorer* tab page.
32. In the context menu, choose *Deployment descriptors->Edit mcd.xml*.



33.
```
        <property_type>RUNTIME</property_type>
        <property_value>WDLITE</property_value>
    </property>
    <property>
        <property_type>COMPONENT_FILE</property_type>
        <property_value>demo.sap.com~test_ui~implementation.sda</property_value>
    </property>
    <property>
        <property_type>VERSION</property_type>
        <property_value>1.0</property_value>
    </property>
  </mcd_properties>
  <authorization_dependencies>
     <dependency />
  </authorization_dependencies>
     <mcd_dependencies>
        <dependency>
            <dependency_name>appcomp</dependency_name>
            <dependency_version>1.0</dependency_version>
        </dependency>
<!--@@begin generation mcd_dependencies (do not edit)-->
<!--@@end generation mcd_dependencies-->
    </mcd_dependencies>
<!--@@begin generation mbo_dependencies (do not edit)-->
<!--@@end generation  mbo_dependencies-->
 </mobile component>
```

34. Build the Mobile UI component.
35. Deploy the UI and application Component and launch it.



Implementing the event handling code for custom control

The custom control developer has to handle all events of the custom control manually. This means writing the usual SWT code in the implementation of UiElementMethodCallAdapter of the custom control. Events which need to be forwarded to the UI must be additionally wrapped to a UIEvent and forwarded to the doHandleCustomControlEvent() method of the particular UIGenericElement. Just have a look at the examples on how to do this.

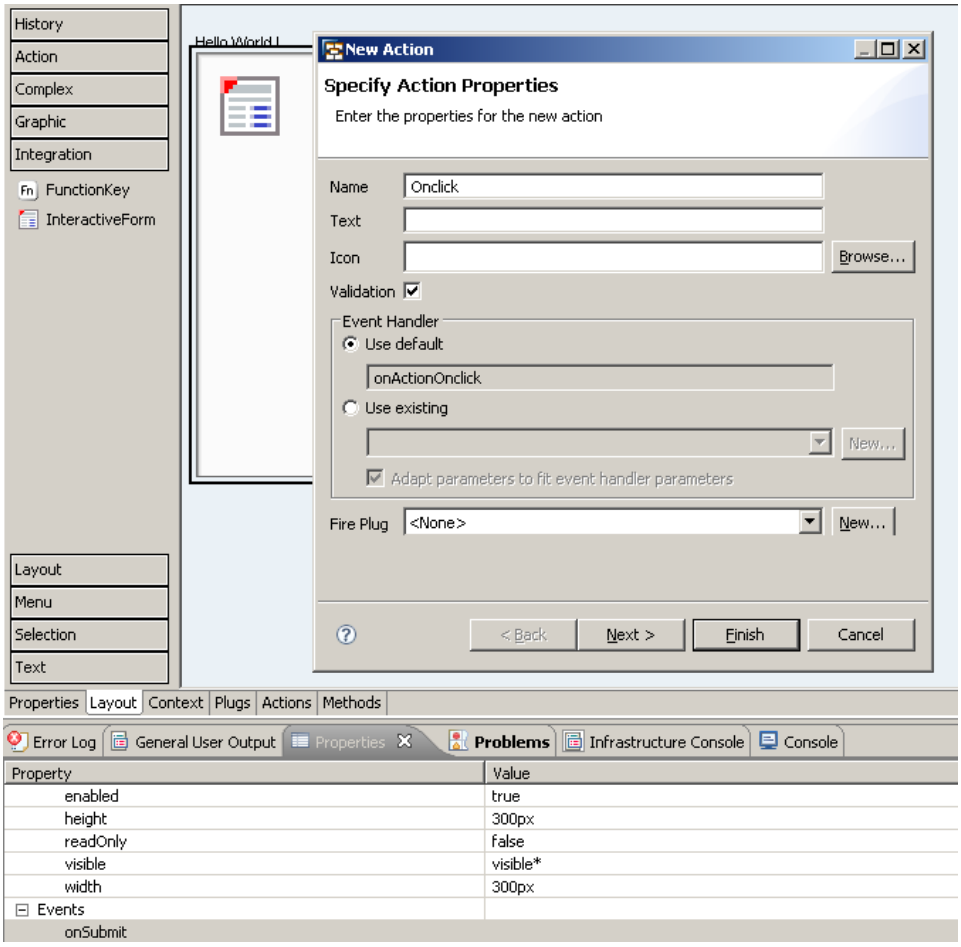
```
/**
 * SWT Event Listener: Translate the native SWT event into a wdlite event and forward
it. Note you have to implement the empty IEventSource interface to do this.
 */
public void widgetSelected(SelectionEvent arg0) {
                UIEvent wdliteUiEvent = new UIEvent(this, IEvent.EVENT_ONACTION);
                //add any data to the event, can be accessed by application developer
                wdliteUiEvent.setEventData("someEventData", "yourcustomData");
                //allows to distinguish between multiple events of a custom control
                wdliteUiEvent.setEventData("eventType", "btnClick");
                wdliteUiGenericElement.doHandleCustomControlEvent(wdliteUiEvent);
                wdliteUiGenericElement.updateProperty("textProp", buttonProb +
"updated");
}
```

Handling the event in the application

Go to the View where custom control has been integrated, go the properties of *Interactive form*, create a method for *onSubmit* Event and this method handles the click event of the custom control

```
/**
 * Event handler [onActionOnclick].
 */
public void onActionOnclick (Event wdEvent) {
//@@begin onActionOnclick(ServerEvent)
   if(wdEvent.getEventData("someEventData").toString()=="yourcustomData")
          {
                //write the logic for on click event
          }
```

```
    //@@end
}
```



Setting the properties of custom control

To set the properties of the custom control, use the following code in the *wdDoModifyView* method of the view

```
view.get<viewItemName>().setPropertyValue("<property name>", "<value>”);
```

The parameters for the method are self explanatory.

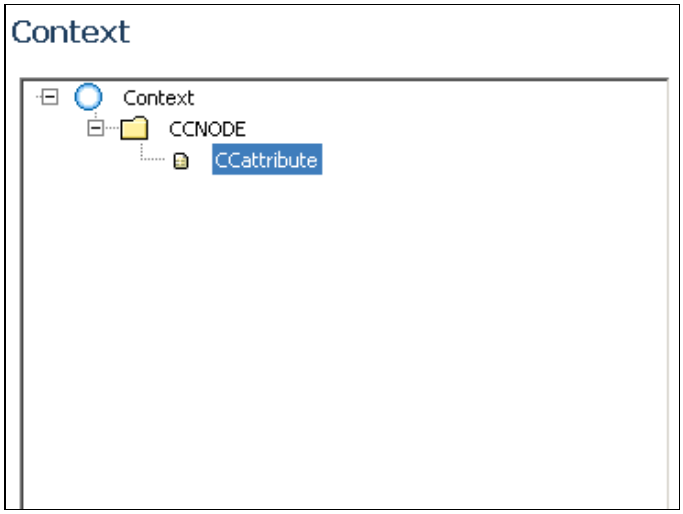
The below given code sets the text property of the custom control button.

```
view.getCustomControlViewItem().setPropertyValue("text", "custom button");
```

Context binding

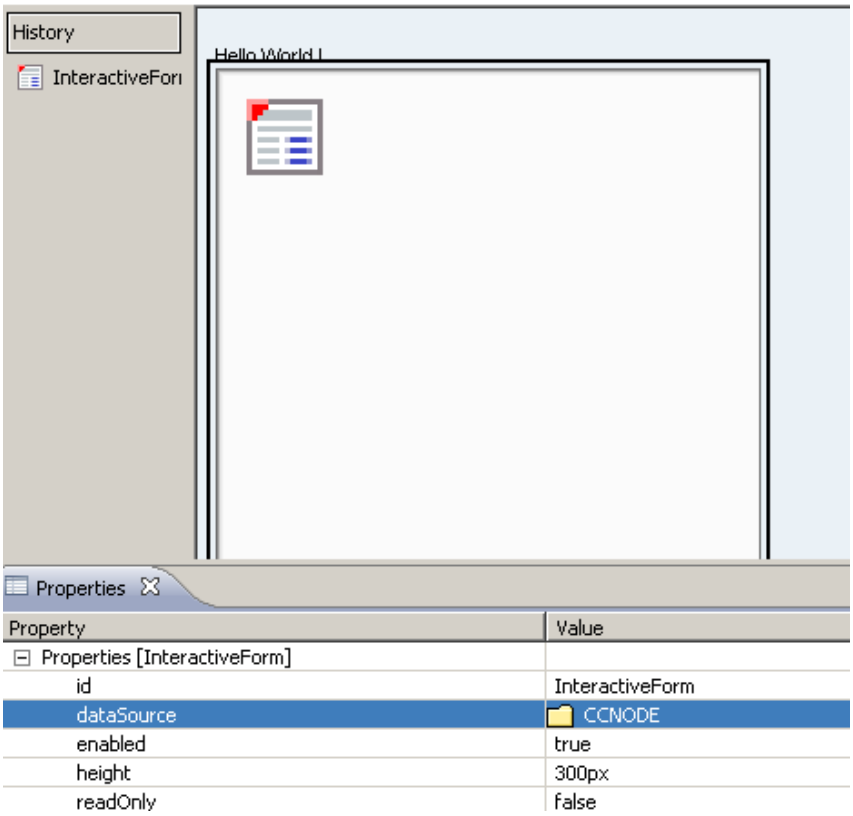
Custom controls can be bound to a single context node. There is no restriction on the number of attributes in the context node.

1. Navigate to the View of the UI Component and choose the Context tab.
2. From the context menu, choose *New->Node*.
3. Create a context node *Manually* and enter CCNODE as the *Name*.
4. From the context menu, choose *New->Attribute*.
5. Enter the CCattribute as the *Name* of the attribute.



6. Navigate to the Layout tab.
7. Drag and drop the InterativeForm UI Element in the View.
8. Bind the Interactive Form to the Context Node

   For the *dataSource* property, enter the value CCNODE.



9. Implement the `UiElementMethodCallAdapter.setContextNode(IWDNode node)` method in the `Adapter` class. You need to store the reference, add listeners and work with the context as usual. (Refer step 2.22)
10. If the collection cardinality and selection cardinality is not set , you need to create the element of the context node in *wdDoInit*() of the view class

```
wdContext.nodeCCNODE().createAndAddCCNODEElement().setCCattribute("CCBUTTON
");
        wdContext.nodeCCNODE().setLeadSelection(0);
```

- Bind the attribute to context node to the custom control text property in the *wdDoModifyView* Method of the view

```
view.getCustomControlViewItem ().bindProperty("text",
wdContext.nodeCCNODE(), "CCattribute");
```

11. Set collection cardinality and selection cardinality of the node to 1..1.In this case add the following code in *wdDoModifyView* Method of the view
    - Select the first element of context node

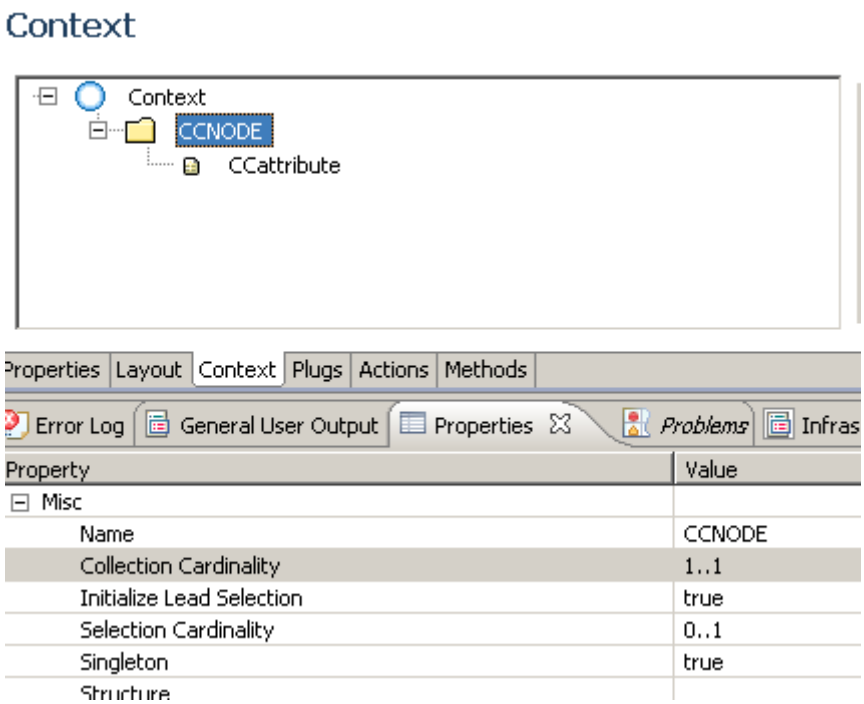
```
wdContext.nodeCCNODE().setLeadSelection(0);
```

    - Update the attribute value

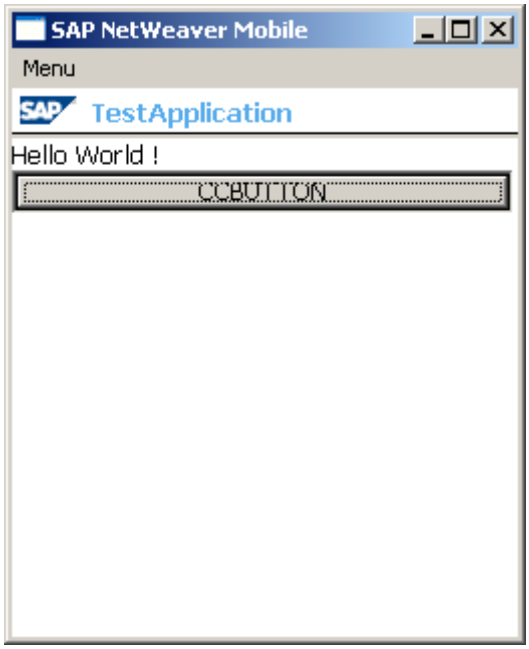
```
wdContext.nodeCCNODE().getCurrentElement().setAttributeValue("CCattribute",
"CCBUTTON");
```

    - Bind the attribute to context node to the custom control text property

```
view.getCustomControlViewItem ().bindProperty("text",
wdContext.nodeCCNODE(), "CCattribute");
```



12. Deploy the UI and application component to the runtime and launch the application.



Related Content

For more information, visit the Mobile homepage.

Copyright