Adobe Flash Islands for Web Dynpro ABAP – Tutorial #1

Applies to:

SAP NetWeaver 7.0 Enhancement Package 1 and higher – Web Dynpro ABAP

Summary

This is the first in a series of tutorials on the topic of Adobe Flash Islands for Web Dynpro ABAP. We will see how to create a simple Flex Component in Adobe Flex Builder. Then we will look at how you adjust this component to make it available as an Island. Finally we will see how to integrate this Island Component into Web Dynpro ABAP.

Author: Thomas Jung

Company: SAP Labs, LLC. – SAP NetWeaver Product Management

Created on: October 31, 2008

Author Bio

Thomas Jung is an SAP NetWeaver Product Manager focusing on Custom Development - particularly in the areas of ABAP, MDM APIs, UI Strategy, and NetWeaver Voice. Before joining SAP Labs in 2006, Thomas was an applications developer for a SAP customer company. He was involved in SAP implementations at this customer as an ABAP Developer for nearly 10 years. He is also the co-author of the SAP Press Books, Advanced BSP Programming and Next Generation ABAP Development.

Table of Contents

Adobe Flash Islands for Web Dynpro ABAP - Estimated Time: 30 minutes
For this we will learn how easy it can be to integrate Adobe Flash or Flex content into our Web Dynpro ABAP application. We have a simple charting control that has been built with Adobe Flex Builder and designed to run within the Web Dynpro Islands framework.

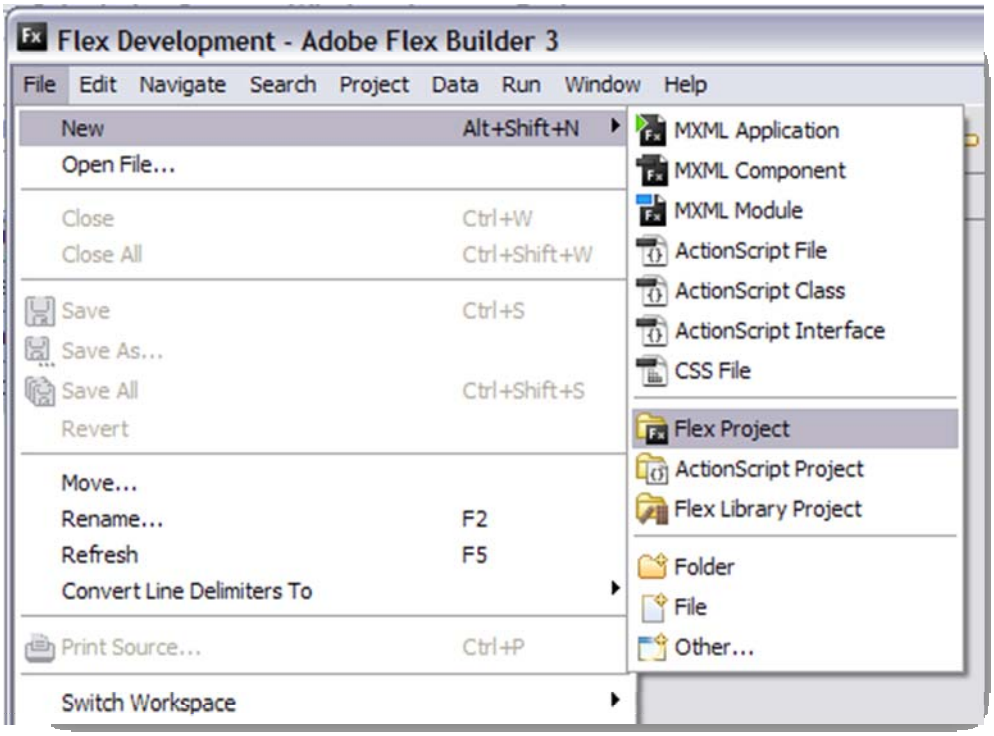
Exercise - Solution

Web Dynpro Islands

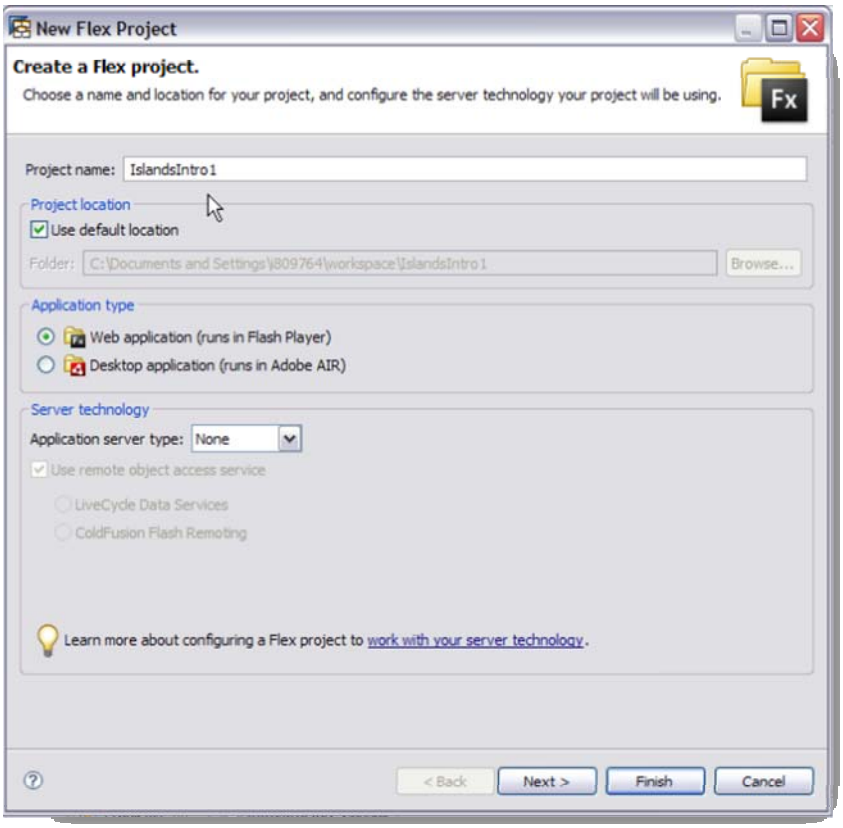
For this we will learn how easy it can be to integrate Adobe Flash or Flex content into our Web Dynpro ABAP application. We have a simple charting control that has been built with Adobe Flex Builder and designed to run within the Web Dynpro Islands framework.

Start by creating a new Flex Project in Flex Builder.

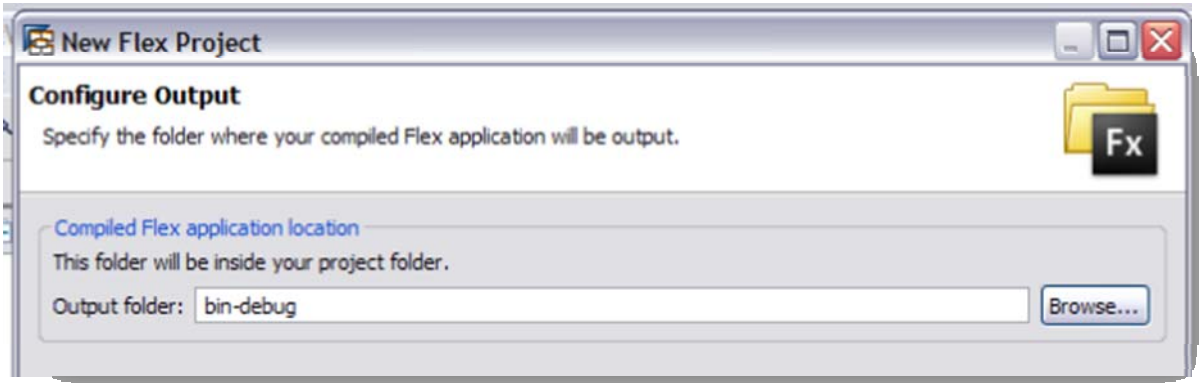
1. Open Adobe Flex Builder
2. Choose New->Flex Project.



3. Name your project IslandsIntro1. You can choose the location to store the project, just be sure to remember the location as we will need to access it later in the tutorial to upload the finished Flex Component to the ABAP Server. Be sure to leave the application type to *Web application (runs in Flash Player)*.

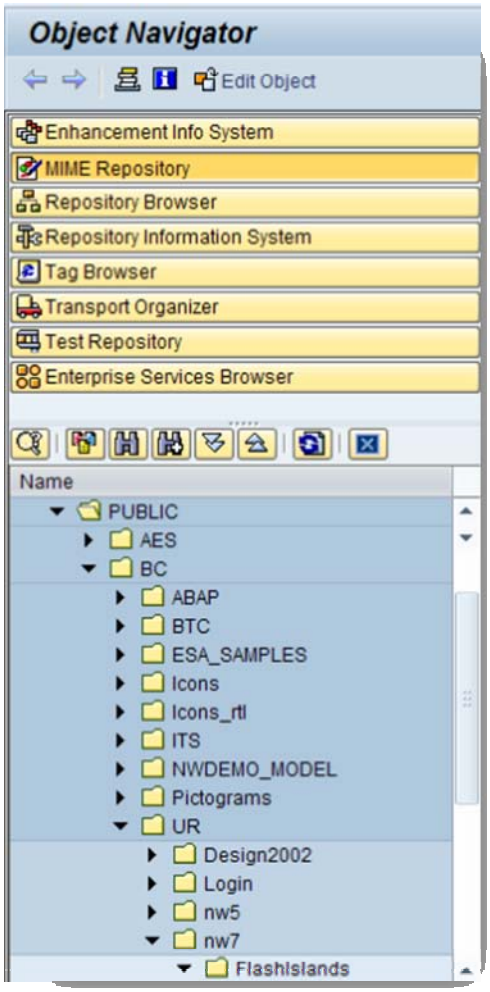


4. Click *Next*. On the next screen, leave the default output folder of bin-debug. Click *Next* again.

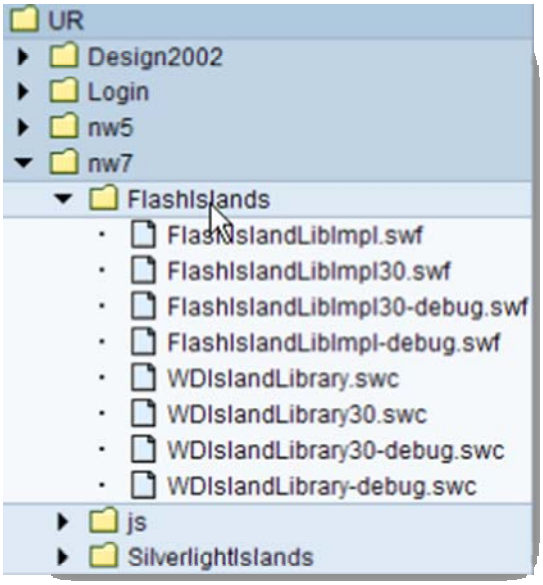


5. At this point we need to add the SAP Supplied FlashIslands Component into our project. It is this SAP supplied library that does most of the work to perform the data transformations and provides the functions for eventing.

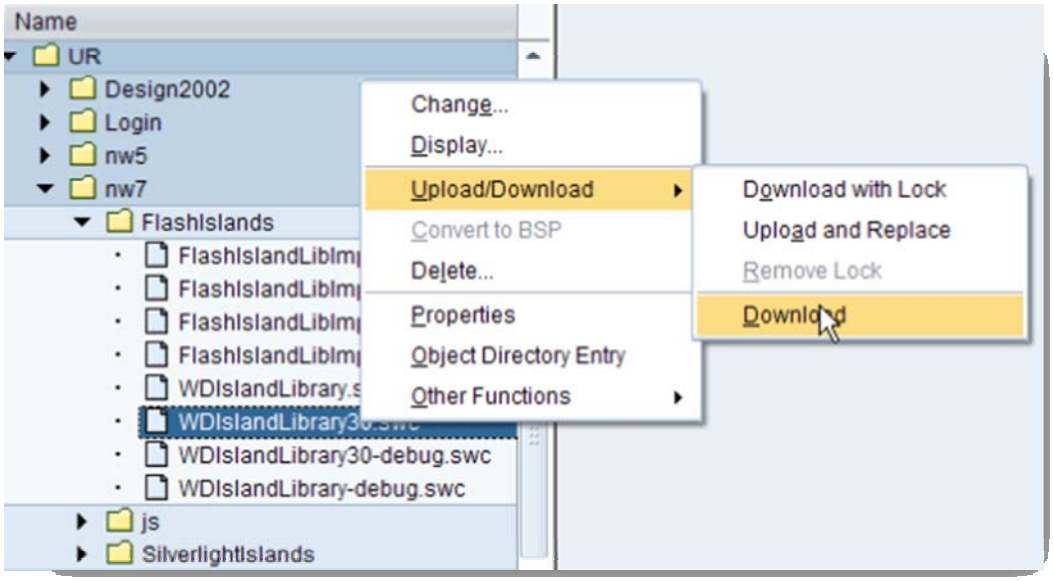
The necessary library is stored on the ABAP Server in the MIME Repository. Go to SE80 and choose the MIME Repository View. Then navigate through the folders: PUBLIC->BC->UR->nw7->FlashIslands



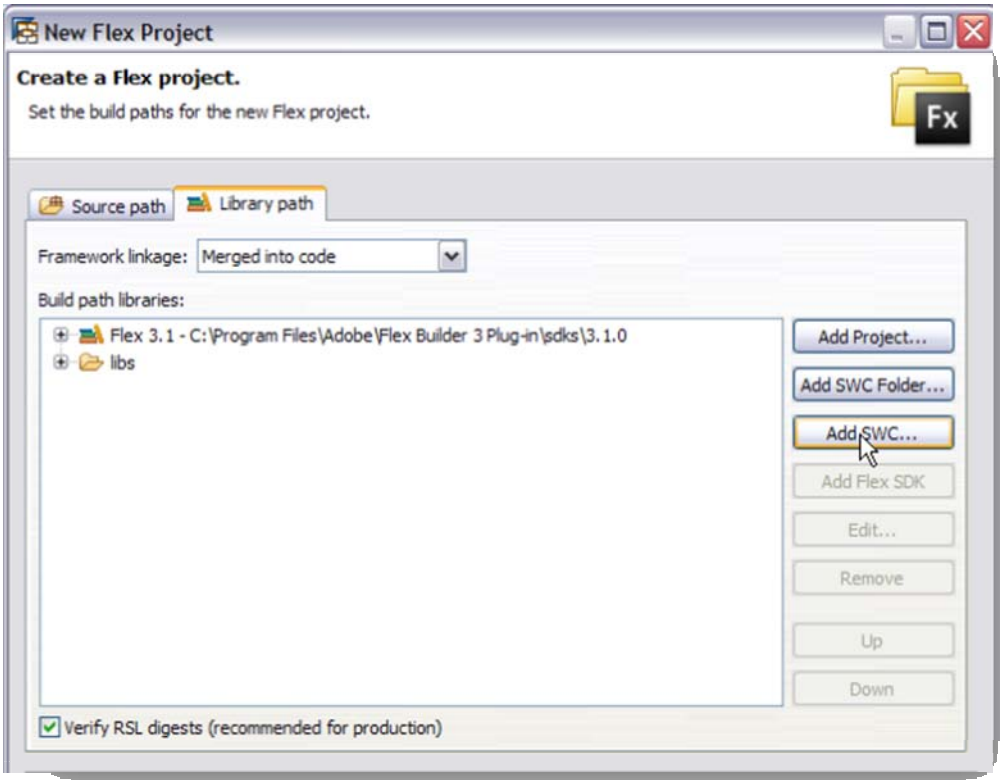
a. There are multiple files in this folder. The *Impl.swf files are linked at runtime to the compiled Flex components. These contain the majority of the Islands functionality. By linking at runtime we reduce the size of the Flex Component (saving bandwidth) but it also makes it easier for SAP to patch the library by delivering a new version in the MIME repository via support packages. You then get fixes and new functionality without having to recompile your Flex Components.
The library that we are interested in for adding to our Flex Component Project is the WDIslandsLibrary*.swc. The files without a numeric suffix are the Flex 2.0 versions. The ones with the 30 suffix are for Flex 3.0. There are also debug versions of each library. (Please note that Ramp-up customers with 7.0 SP2 or lower will not see the 3.0 version of the libraries and will have to use the 2.0 versions).



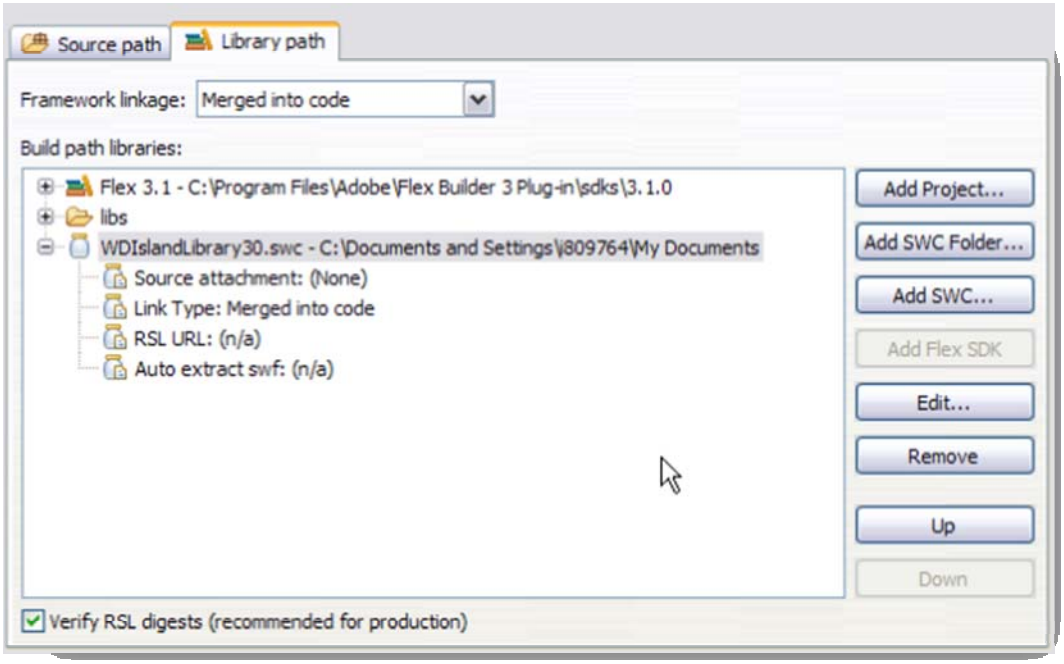
b. For our purposes we will use the 3.0 version of the non-debug library (although if you don't have the 3.0 version in your system it is fine to use the 2.0 version. You will simply need an alternative step later in the tutorial). Right mouse click on the WDIslandsLibrary30.swc and choose Upload/Download->Download. Save the file in a convenient location as well will need to access it in the next step.



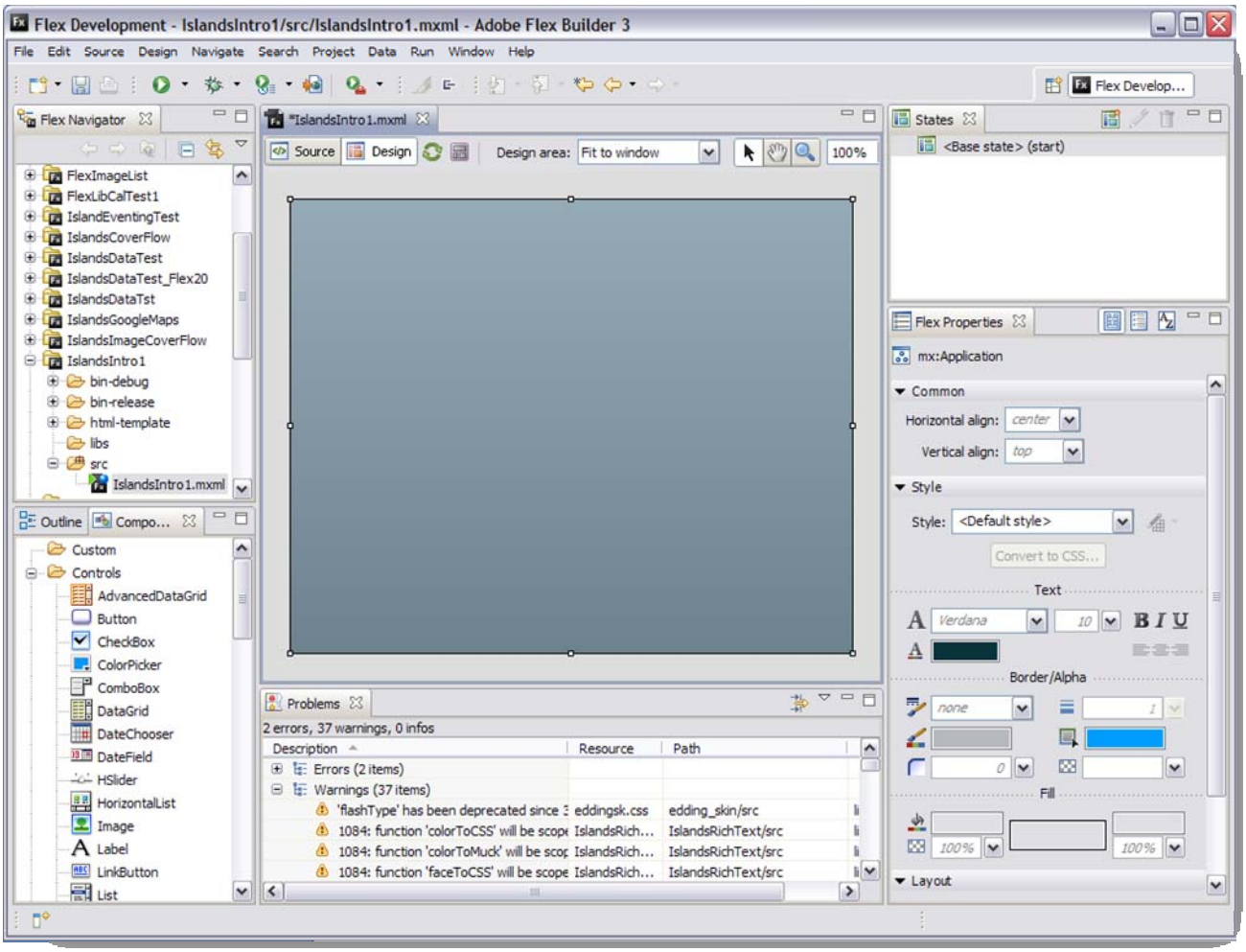
Returning to the Flex Builder we are still in the Create a Flex Project wizard. Choose the *Library Path* tab and click on the *Add SWC button*.



Choose the WDIslandsLibrary30.swc that we downloaded from the ABAP MIME repository in the previous steps and then click the *Finish* button.

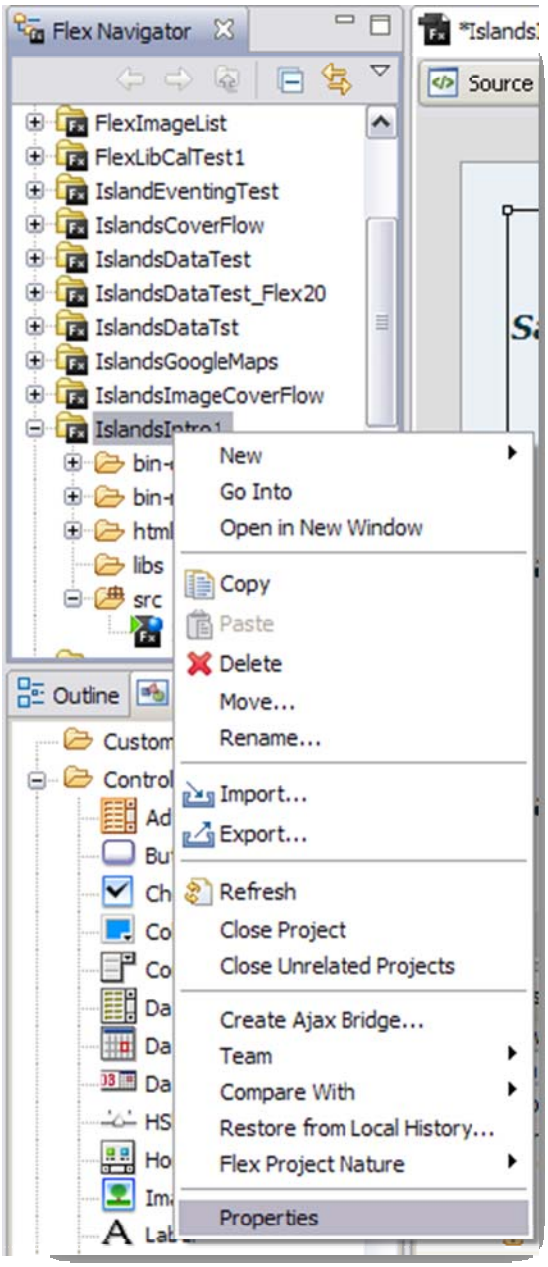


You will then have the empty shell of a Flex Component available in the Flex Builder:

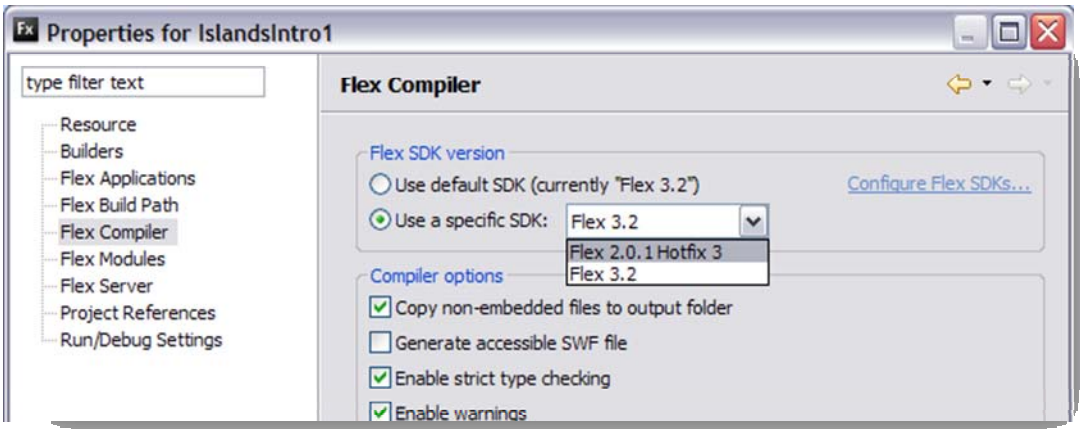


Note: Optional Step – If you have to use the Flex 2.0 version of the WDIslandLibrary.swc file then you need to also perform this optional step to also force your Flex Component to compile as version 2.0. This step is not needed if you use the Flex 3.0 version of the library (WDIslandLibrary30.swc).

Right mouse click on your project folder in the Flex Navigator and choose Properties.



c. In the *Properties* dialog set the left hand navigation to *Flex Compiler*. Then for the *Flex SDK Version* choose **Use a specific SDK**. From the drop down choose **Flex 2.x** (actual value dependent upon the version and patch level of the SDK that you have installed on your system).



Coding the Flex Component

1. You can now switch to the Source tab for the IslandsIntro1.mxml editor. We will walk through each section of the coding to explain what it is doing; but at the end of this chapter of the tutorial we have the complete code listing for easy cut-and-paste. Although this isn't intended to be a general tutorial on Flex development (check the Adobe website for such tutorials) a little explanation is in order.
   Flex development is a mixture of MXML tags and ActionScript. MXML tags allow you define the layout of the UI elements on the screen. You can also make the arrangement via the WYSIWYG editor on the Design tab and the MXML will be generated for you.
   ActionScript is the scripting language of Flex Components. This is where you place event handlers, dynamic manipulation of the UI and other such coding blocks.
   For our Flash Island component we will use a mixture of MXML and ActionScript.

2. We start by altering some of the properties of the **mx:Application** tag. We will change the layout to vertical and set a *backgroundColor* and *backgroundGradientAlphas* value that matches up to the background settings in Web Dynpro. We also set the *width* and *height* to 100%. This allows the Flash Island component to fill the entire container set aside for it on the screen by Web Dynpro. We will control the actual size of the component later when we create the definition of the Island within Web Dynpro ABAP.
   Finally we come to the most important change the **mx:Application** tag for the usage of the Islands framework. We need to set an ActionScript function to be fired upon initialization (the *initialize* property) of the component. It is the coding within this function that will register this component with the Flash Islands framework.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
    width="100%" height="100%" initialize="initApp()" backgroundColor="#eaf1f6"
    backgroundGradientAlphas="1.0,1.0">
```

3. Next we have the scripting block for the component. In this scripting block we 3 different sections.
   a. First we have the import statements. These declare the libraries we want to use in our application. Notice the import of sap.FlashIslands. This gives us access to the functions that are exposed by the WDIslandsLibrary30.swc that we added to our project earlier.

```
<mx:Script>
    <![CDATA[
        import mx.rpc.soap.SOAPFault;
        import sap.FlashIsland;
        import mx.collections.ArrayCollection;
        import mx.graphics.IFill;
        import mx.graphics.SolidColor;
```

   b. Next we have the variables section. Data binding between the Island and the surrounding Web Dynpro application is really quite easy. Any public variables or public GET/SET methods are immediately exposed to the Island Framework. The SAP provided FlashIslands library will do all the translation between the data types for us. From the Flex Developers' standpoint we only need to make these variables public. So for our Chart we want to get the data from Web Dynpro.
   In Web Dynpro ABAP this data is an internal table bound to a 0:n context node. The corresponding object ot such a data object in Flex is an Array Collection. However Array Collections do not work exactly like ABAP internal tables in that their "columns" are not statically typed at design time. Therefore it will adapt to whatever columns are passed through from Web Dynpro. However we won't want to tie our Flex code to the column names that are used in ABAP. Therefore we can declare "alias" fields to point to the actual column names. These are the next three public variables – MonthLbl, otd, and rty. Later when we do the name mapping in Web Dynpro you will see how we specific what Web Dynpro context attribute name will get filled into these alias variables. This makes it possible to rename objects in Web Dynpro without breaking the connected Flex Coding
   Finally the otdFill and rtyFill variables really don't have anything to do with the Islands interface. They are local variables to define the colors we want to use on our Bar Chart.

```
[Bindable]
public var dataSource:ArrayCollection;
[Bindable]
public var MonthLbl:String;
[Bindable]
public var otd:String;
[Bindable]
public var rty:String;

public var otdFill:IFill = new SolidColor(0xbcd0df, 1.0);
public var rtyFill:IFill = new SolidColor(0xFFD965, 1.0);
```

   c. Fine final script section is the initApp function. Remember this is the function that will be called when the component is initialized. In this function we only need to call the register function of the SAP supplied FlashIsland library. This registers the entire component with the Flash Islands framework.

```
        public function initApp():void
        {
            FlashIsland.register(this);
        }
    ]]>
</mx:Script>
```

4. In the final code block we the remain MXML tags to define the animations for our chart, the bar chart itself with its inner Axis and Series definitions as well as a Legend.

```
    <mx:SeriesSlide id="slideIn" duration="1000" direction="right" />
    <mx:SeriesZoom id="zoomIn" duration="1000" relativeTo="chart" horizontalFocus="left" />

    <mx:BarChart id="barChart" type="clustered" width="100%" fontSize="16"
        fontWeight="bold" fontStyle="italic" showDataTips="true" >
        <mx:verticalAxis>
            <mx:CategoryAxis categoryField="{MonthLbl}" dataProvider="{dataSource}" />
        </mx:verticalAxis>

        <mx:series>
            <mx:BarSeries xField="{otd}" displayName="OTD" dataProvider="{dataSource}"
                fill="{otdFill}" showDataEffect="{slideIn}" />
            <mx:BarSeries xField="{rty}" displayName="RTY" dataProvider="{dataSource}"
                fill="{rtyFill}" showDataEffect="{zoomIn}" />
        </mx:series>

    </mx:BarChart>
    <mx:Legend dataProvider="{dataSource}" direction="horizontal" />
</mx:Application>
```

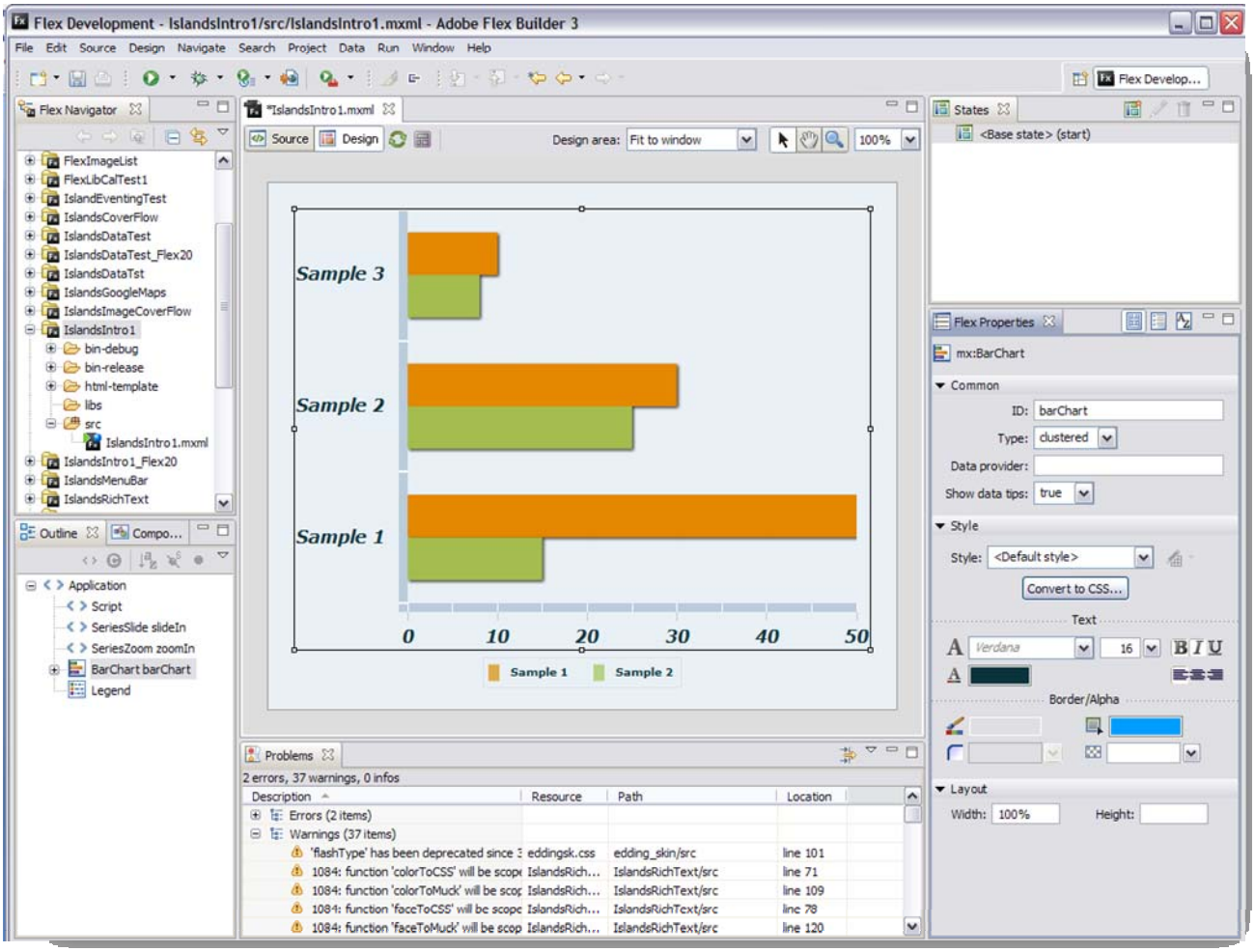
5. Complete Source:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
   width="100%" height="100%" initialize="initApp()"
      backgroundColor="#eaf1f6"
   backgroundGradientAlphas="1.0,1.0">
   <mx:Script>
         <![CDATA[
               import mx.rpc.soap.SOAPFault;
               import sap.FlashIsland;
               import mx.collections.ArrayCollection;
               import mx.graphics.IFill;
               import mx.graphics.SolidColor;

               [Bindable]
               public var dataSource:ArrayCollection;
               [Bindable]
               public var MonthLbl:String;
               [Bindable]
               public var otd:String;
               [Bindable]
               public var rty:String;

               public var otdFill:IFill = new SolidColor(0xbcd0df, 1.0);
               public var rtyFill:IFill = new SolidColor(0xFFD965, 1.0);

               public function initApp():void
               {
                     FlashIsland.register(this);
               }
         ]]>
   </mx:Script>
    <mx:SeriesSlide id="slideIn" duration="1000" direction="right" />
    <mx:SeriesZoom id="zoomIn" duration="1000" relativeTo="chart"
                  horizontalFocus="left" />
```

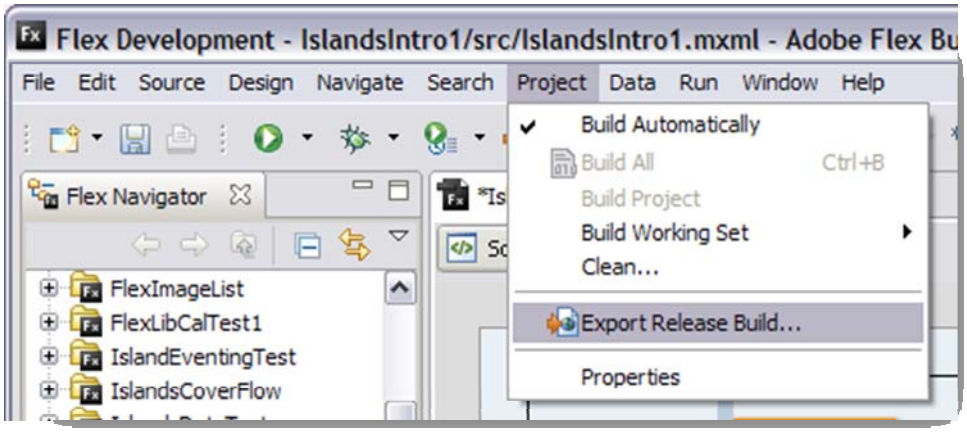
```
  <mx:BarChart id="barChart" type="clustered" width="100%" fontSize="16"
       fontWeight="bold" fontStyle="italic" showDataTips="true" >
       <mx:verticalAxis>
             <mx:CategoryAxis categoryField="{MonthLbl}"
                         dataProvider="{dataSource}" />
       </mx:verticalAxis>

       <mx:series>
             <mx:BarSeries xField="{otd}" displayName="OTD"
                       dataProvider="{dataSource}"
                   fill="{otdFill}" showDataEffect="{slideIn}" />
             <mx:BarSeries xField="{rty}" displayName="RTY"
                       dataProvider="{dataSource}"
                   fill="{rtyFill}" showDataEffect="{zoomIn}" />
       </mx:series>

  </mx:BarChart>
  <mx:Legend dataProvider="{dataSource}" direction="horizontal" />
</mx:Application>
```

Building the Flex Component

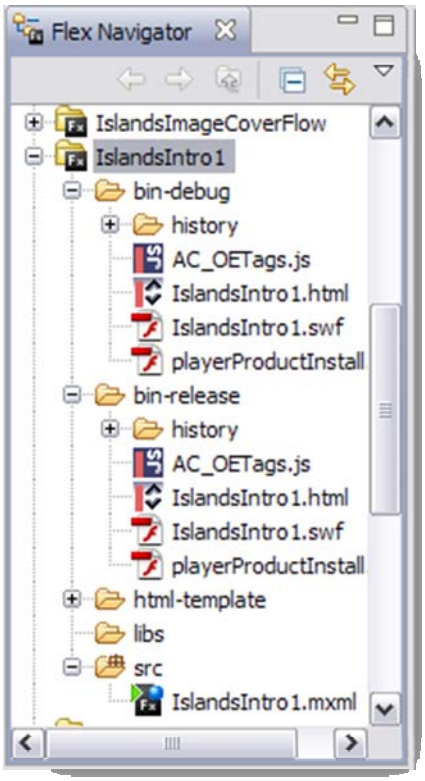
1. The Flex Coding is now complete. You can view the preview of what the final component will look like in the Design view



2. Now if you want to use a debug build of the component that has actually been being built ever time we saved the application. However we are pretty sure that this application is going to work the first time without needing any debugging so we will produce a final build For this choose Project->Export Release build.

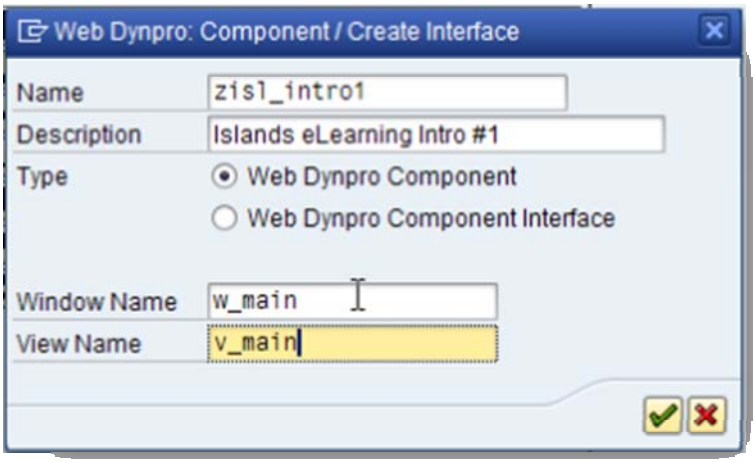


This will create a new folder within your project workspace called bin-release with the compiled runtime version of your component named IslandsIntro1.swf. Remember this location as we will need to upload this file into our Web Dynpro application later in the tutorial.

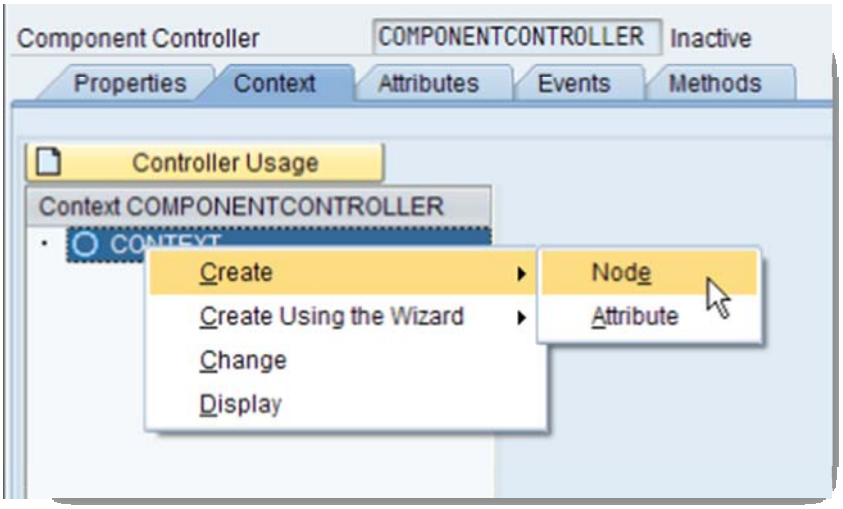


Create the New Web Dynpro Component

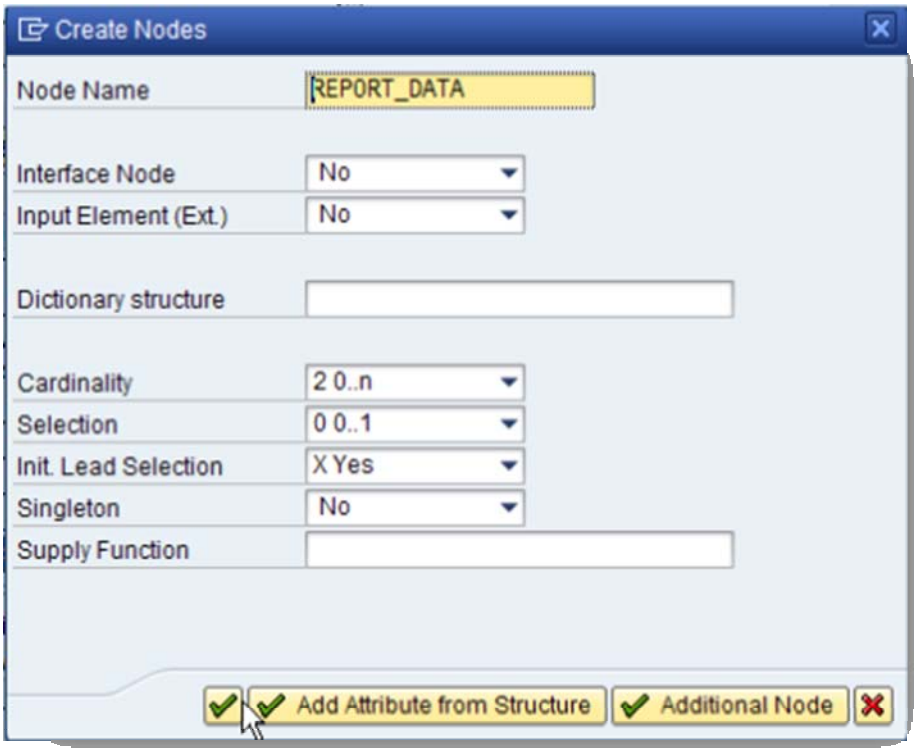
1. We will start this process by creating a normal Web Dynpro Component. Name the Component **ZISL_INTRO1**. Also create the initial Window (**W_MAIN**) and initial View (**V_MAIN**).



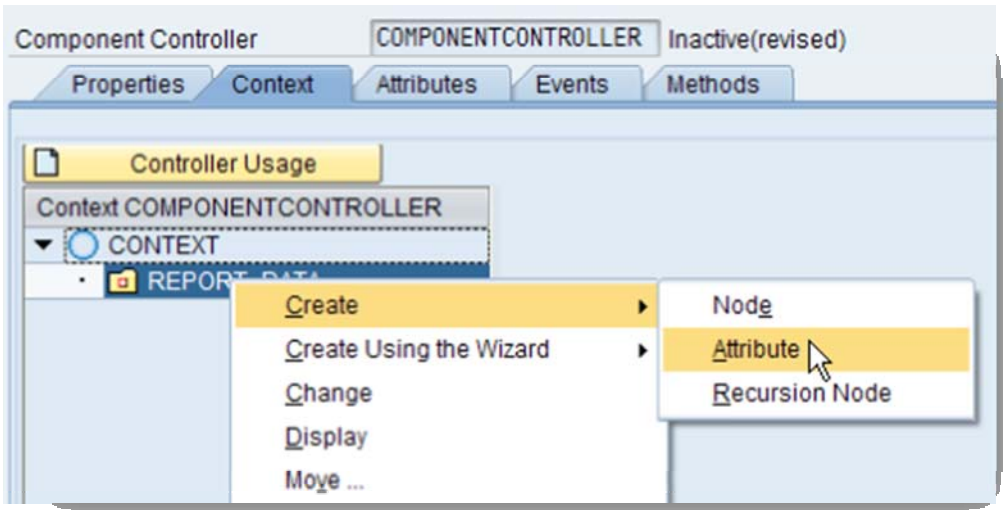
2. We now need to model that data that will be displayed both in the Web Dynpro table UI element and the Flash Island Bar Chart. For this we will go to the context of the Component Controller.
   a. Right mouse click on the Context and choose to create a new Node



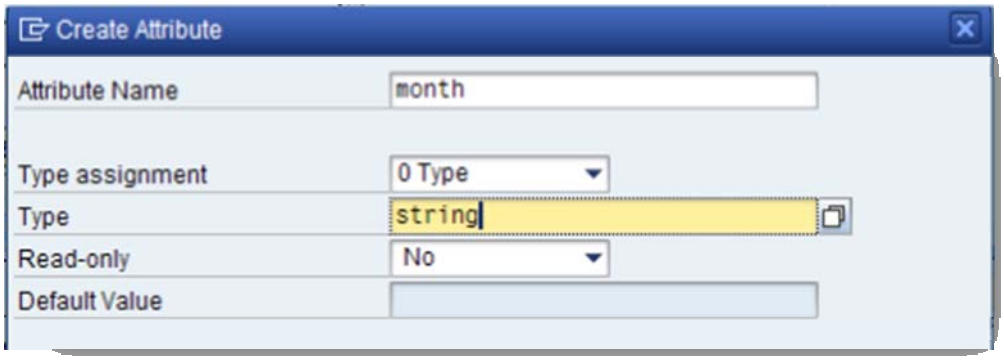
b. Create a Node named **REPORT_DATA** with a cardinality of 0:n.



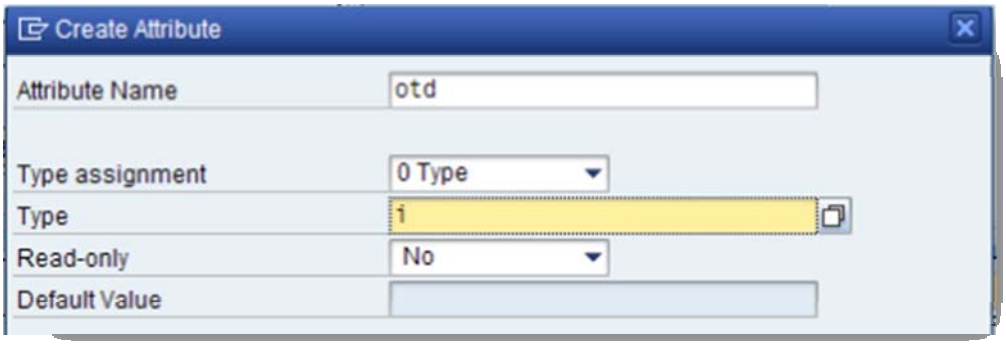
c. Now we need add the attributes to our **REPORT_DATA** context node. There are three attributes that we will add – the Month Description, the On Time Delivery value, and the Roll Throughput Yield value.



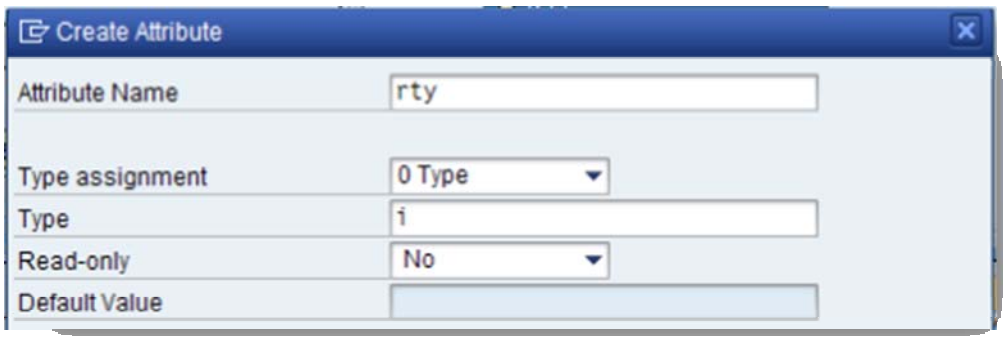
i. Month Description: type STRING



ii. On Time Delivery: type I (integer)



iii. Roll Throughput Yield: type I (integer)



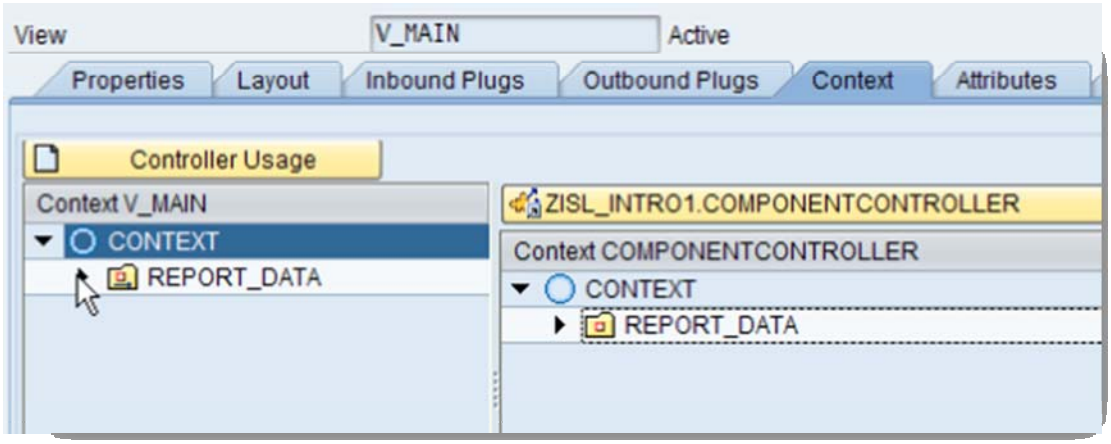
3. We now need some code to populate the **REPORT_DATA** context node. Since this is just a very simple example application we will actually hard code these values. We will use the **WDDOINIT** method of the component controller to do this population.

```
METHOD wddoinit .
```

```
  DATA lo_nd_report_data TYPE REF TO if_wd_context_node.
  DATA lt_report_data TYPE wd_this->elements_report_data.
  lo_nd_report_data = wd_context->get_child_node(
     name = wd_this->wdctx_report_data ).

  FIELD-SYMBOLS <wa_data> LIKE LINE OF lt_report_data.
  APPEND INITIAL LINE TO lt_report_data ASSIGNING <wa_data>.
  <wa_data>-month = '07/2008'.
  <wa_data>-otd   = 90.
  <wa_data>-rty   = 85.

  APPEND INITIAL LINE TO lt_report_data ASSIGNING <wa_data>.
  <wa_data>-month = '08/2008'.
  <wa_data>-otd   = 73.
  <wa_data>-rty   = 78.

  APPEND INITIAL LINE TO lt_report_data ASSIGNING <wa_data>.
  <wa_data>-month = '09/2008'.
  <wa_data>-otd   = 94.
  <wa_data>-rty   = 56.
*
  lo_nd_report_data->bind_table(
     new_items = lt_report_data set_initial_elements = abap_true ).
ENDMETHOD.
```

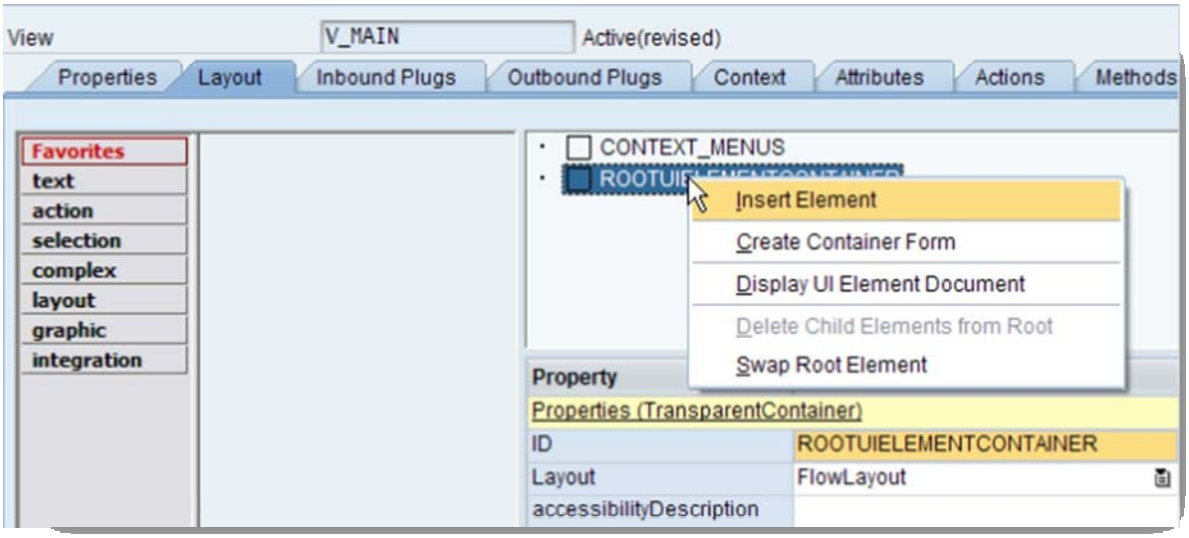
Building the V_MAIN view layout.

1. We can now go to the **V_MAIN** view and map the Context Node from the Component Controller to the View Controller Context via Drag and Drop.



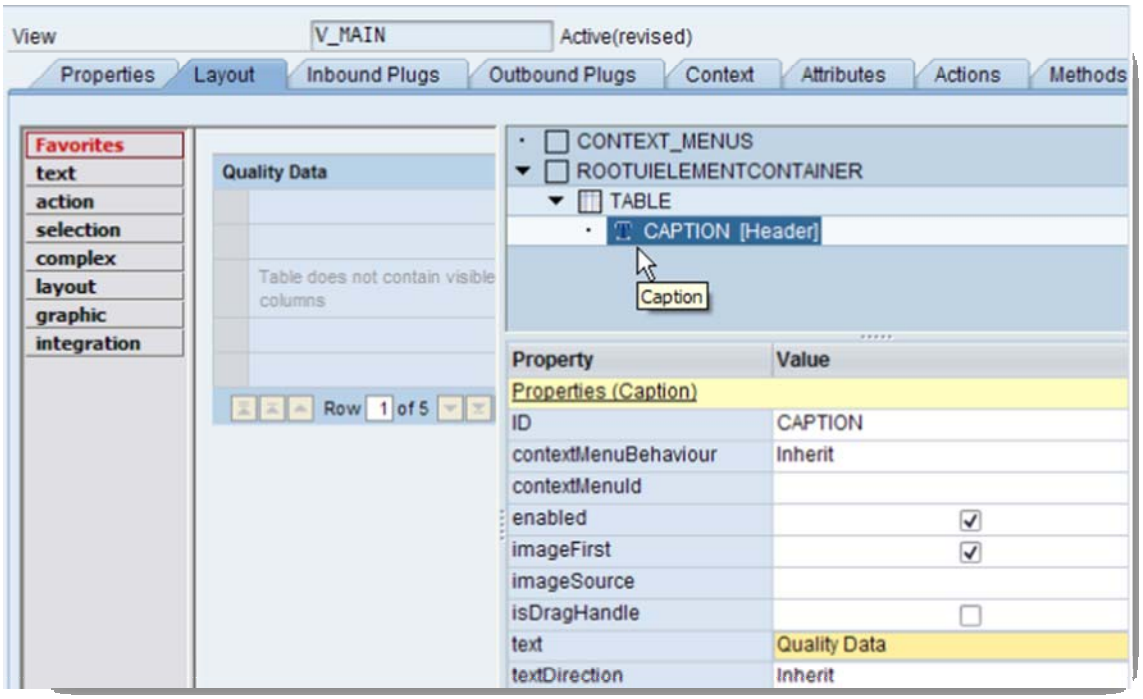
2. Now return to the Layout tab of the **V_MAIN** view. We want to create a Table UI element.

Right Mouse Click on the **ROOTUIELEMENTCONTAINER** and choose *Insert Element*.

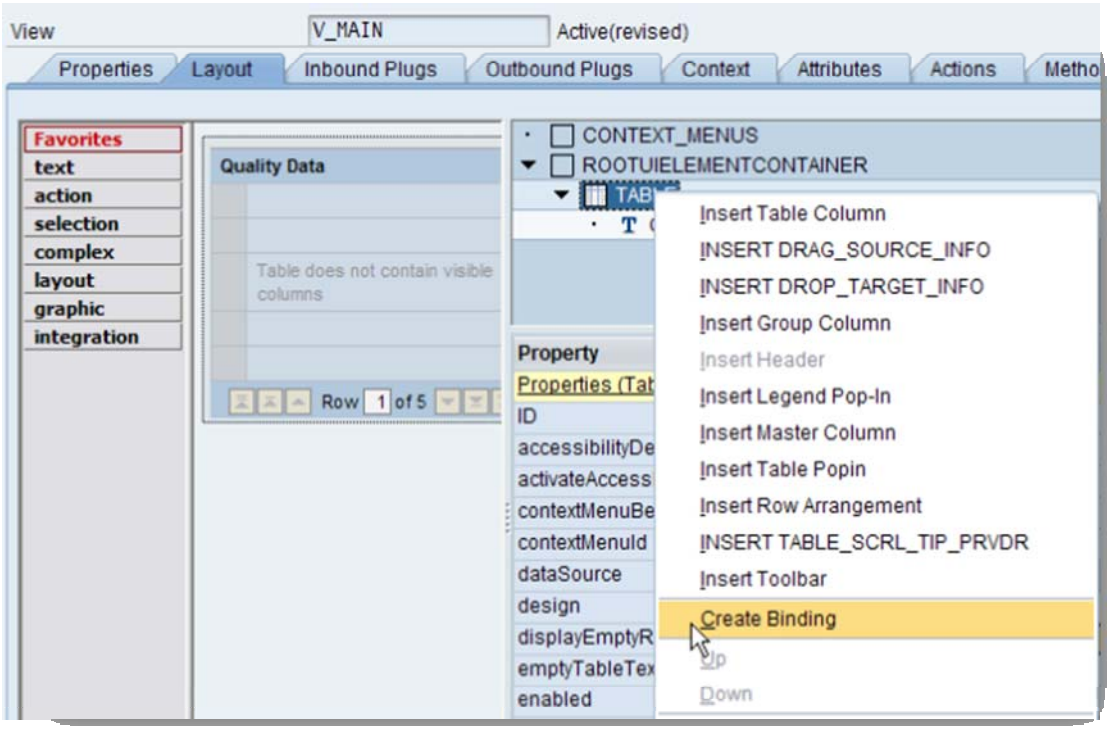


In the Create element dialog, choose a UI element named *table* of type **Table**.

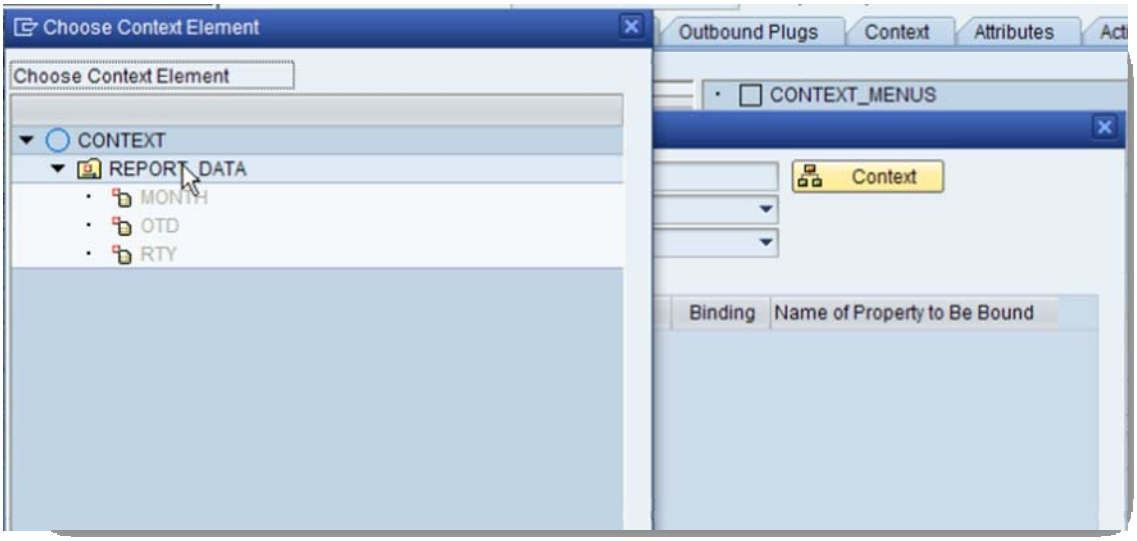
Add a caption called *Quality Data* to the table


View
V_MAIN
Active(revised)
Properties
Layout
Inbound Plugs
Outbound Plugs
Context
Attributes
Actions
Methods
Favorites
text
action
selection
complex
layout
graphic
integration
Quality Data
Table does not contain visible columns
Row 1 of 5
CONTEXT_MENUS
ROOTUIELEMENTCONTAINER
TABLE
CAPTION [Header]
Caption
Property
Value
Properties (Caption)
ID
CAPTION
contextMenuBehaviour
Inherit
contextMenuId
enabled
imageFirst
imageSource
isDragHandle
text
Quality Data
textDirection
Inherit

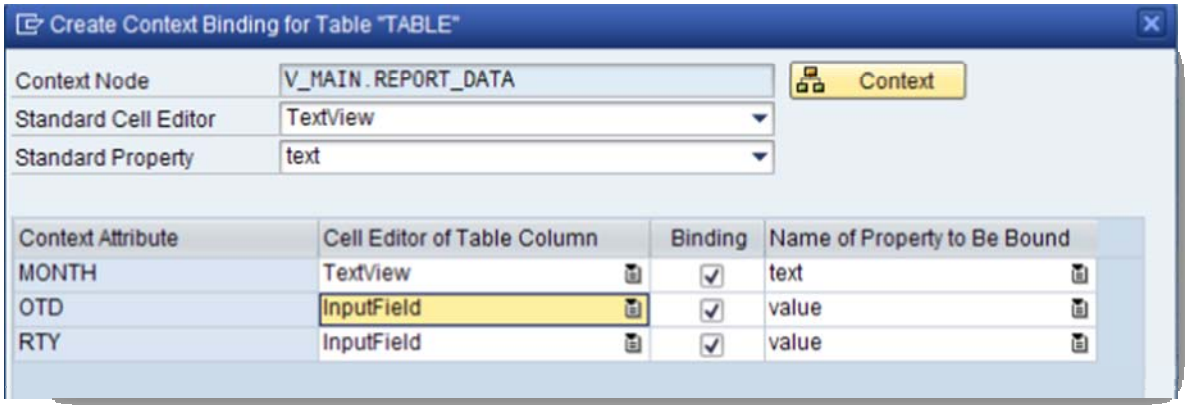
a. Right Mouse Click on the **Table** UI element and choose *Create Binding* to start the Binding Wizard.



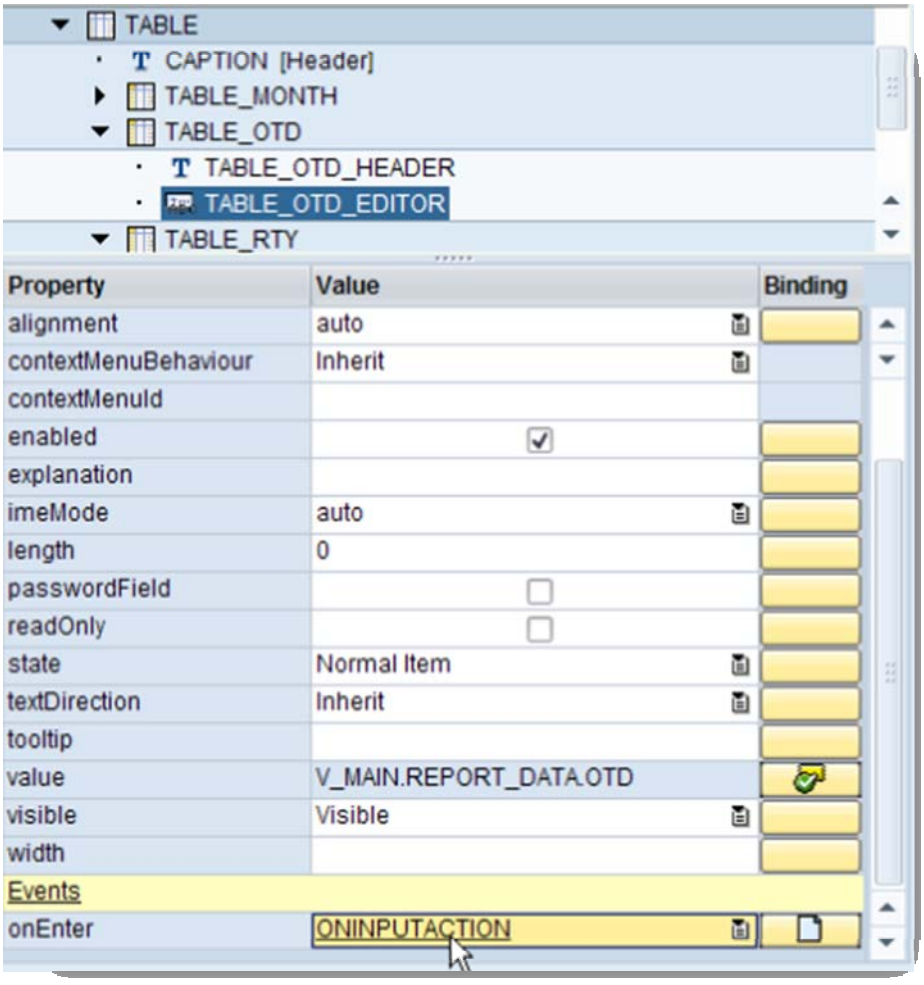
b. Choose REPORT_DATA as the context node to bind the table to



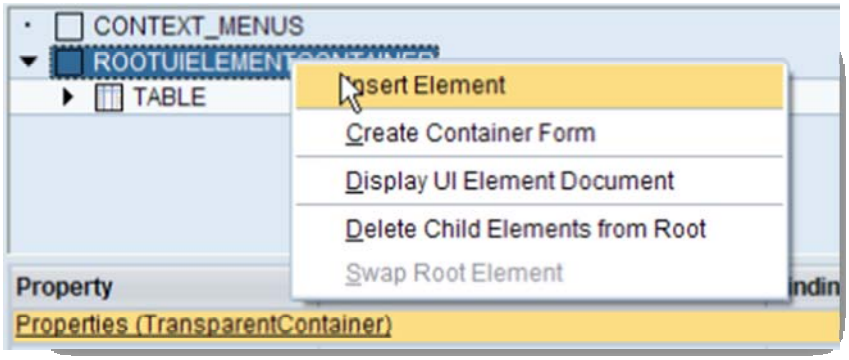
c. Map the MONTH attribute to a **TextView** and the OTD and RTY attributes to **InputField**. This way users of the application can change the OTD and RTY values and view the update to the Bar Chart immediately.



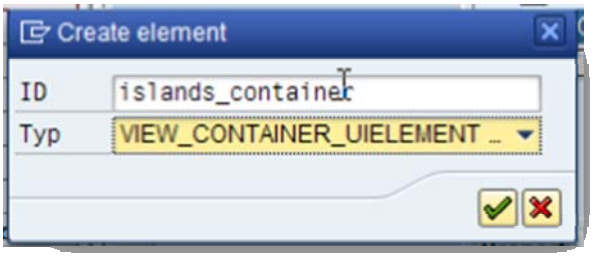
d. In order to Trigger the update of the Flash Island Bar Chart, place an onEnter Action on the OTD and RTY InputField. There doesn't actually have to be any coding in the Event Handler for this Action. Just the act of triggering the event will update the shared context between Web Dynpro and the Flex Component.



3. The **FlashIsland** UI element cannot be placed directly within the UI element Hierarchy. It must always be a root node UI element of a View. Therefore within our **V_MAIN** we can only reserve a place holder for the Flash Islands View using a **VIEWCONTAINERUIELEMENT**.
    a. Right Mouse Click on the **ROOTUIELEMENTCONTAINER** and choose *Insert Element*.

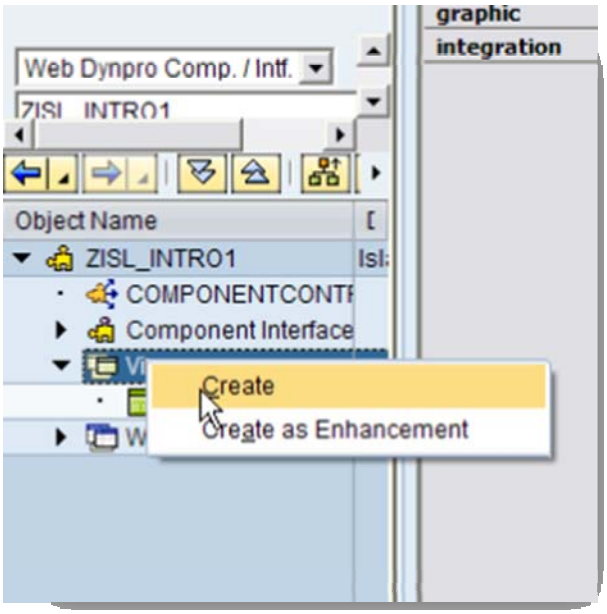


    b. In the *Create element* dialog create a new UI element named **ISLANDS_CONTAINER** of type **VIEW_CONTAINER_UIELEMENT**.

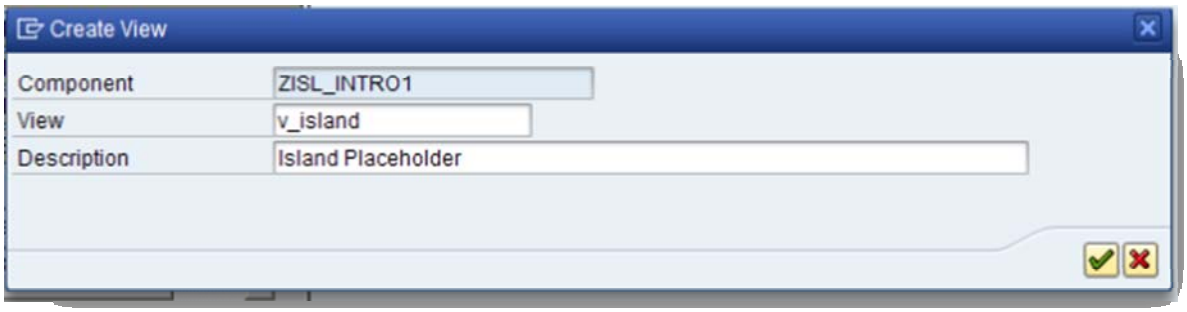


Creating the Flash Islands View

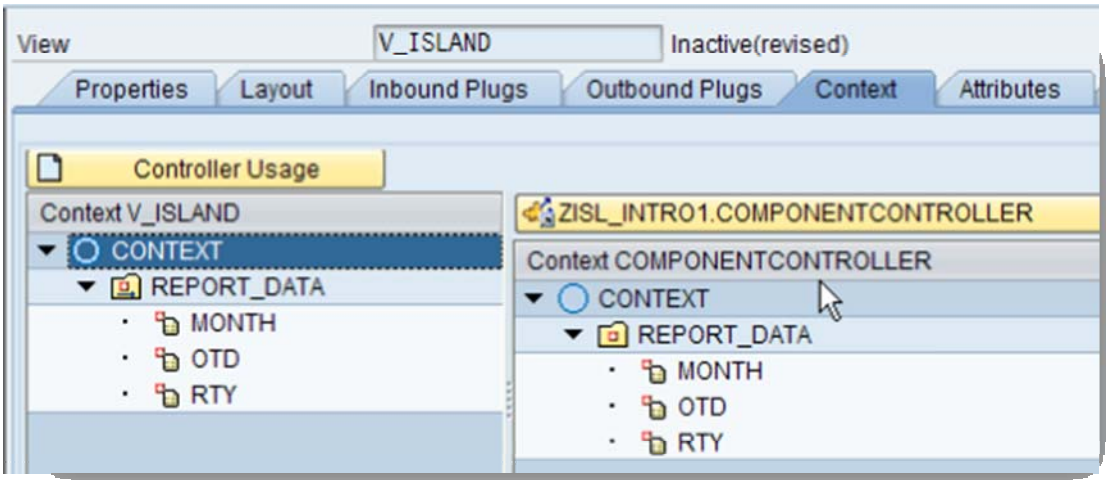
1. Create a new View named V_ISLAND
   a. Right mouse click on the Views node of the Component and choose *Create*.



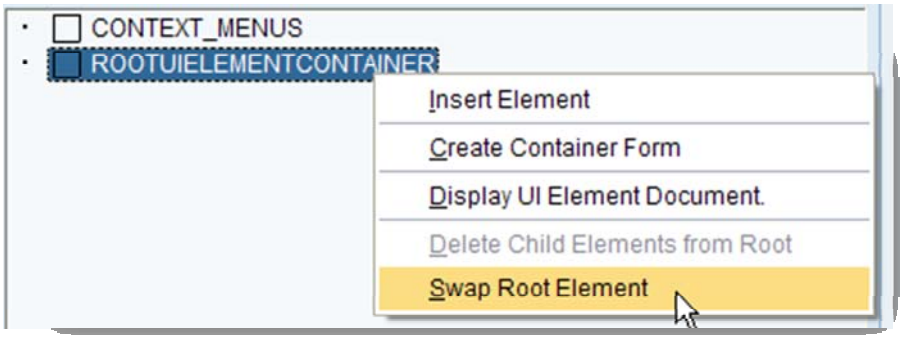
   b. Name the new View – **V_ISLAND**.



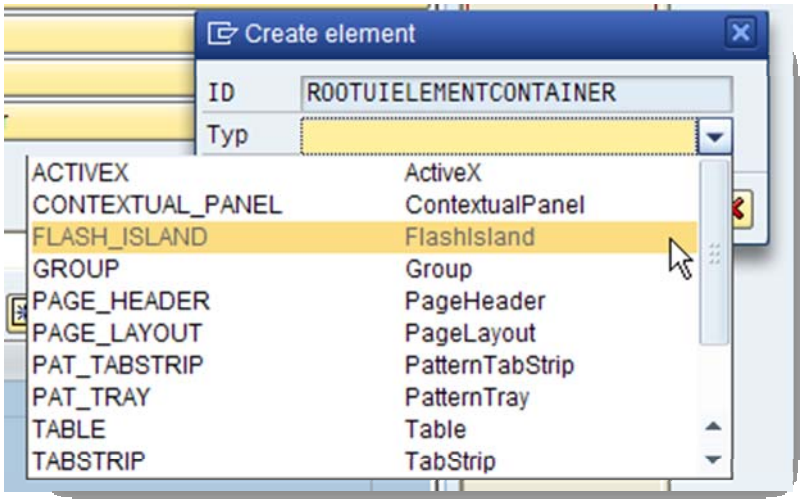
2. Map the Component Controller Context Node **REPORT_DATA** to the View Context.



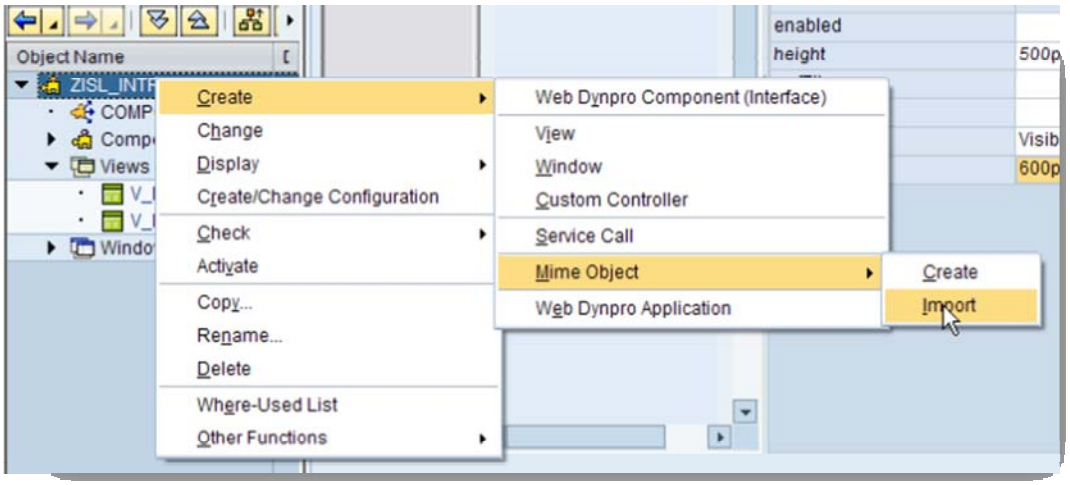
3. Add the ISLAND UI elements to the View and Map to the Flex Component names
   a. The FlashIslands UI element must be the ROOT UI Element. Therefore, Web Dynpro has a new feature that lets you swap the UI Element of the **ROOTUIELEMENTCONTAINER**. Right Mouse click on the **ROOTUIELEMENTCONTAINER** in the UI element Hierarchy and choose *Swap Root Element*.



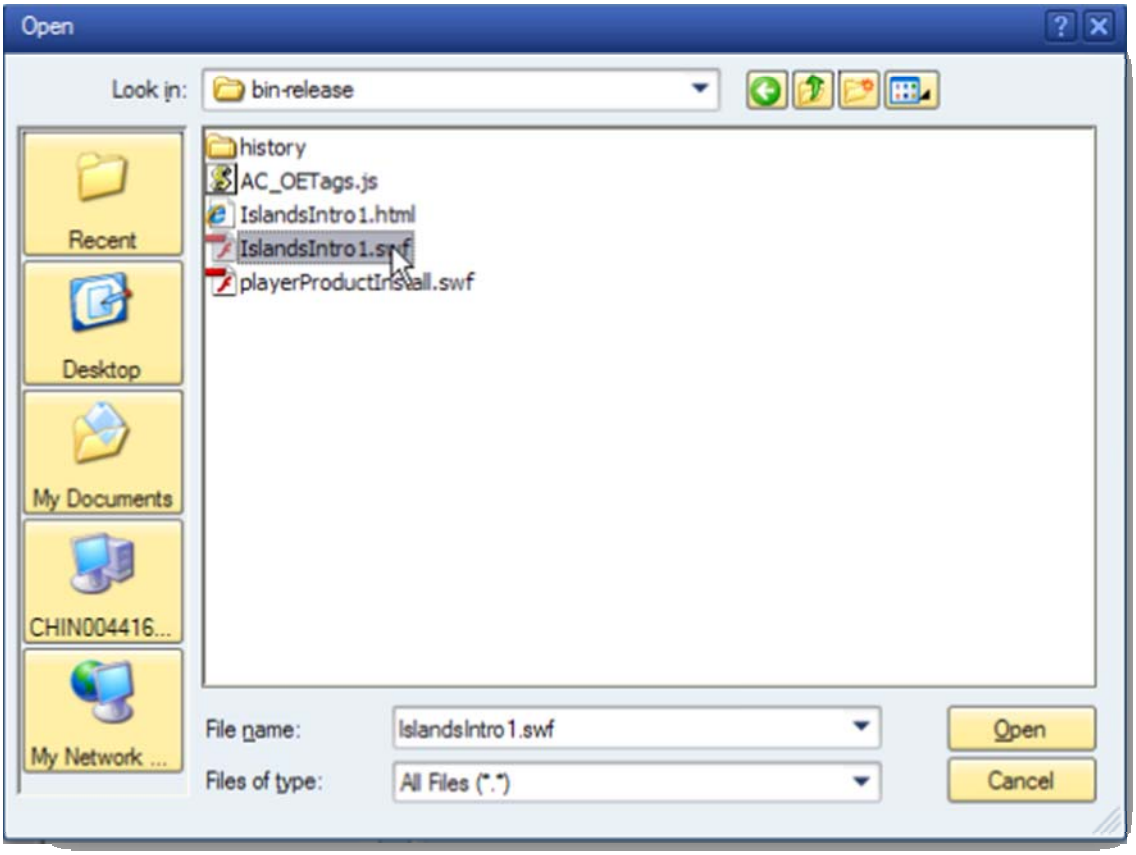
Choose FlashIsland from the list of possible UI elements.



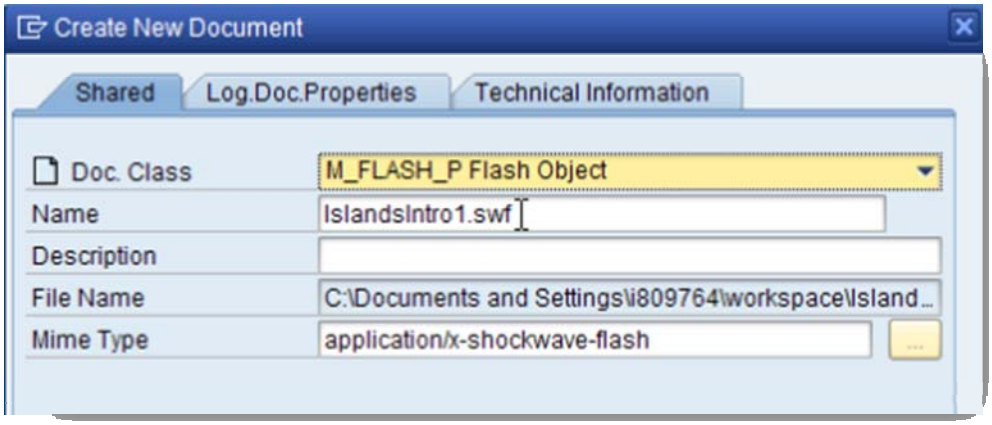
b. We now need to upload the compiled Flex Component file to the ABAP application server. Right Mouse Click on the Component and choose *Create->Mime Object->Import*.



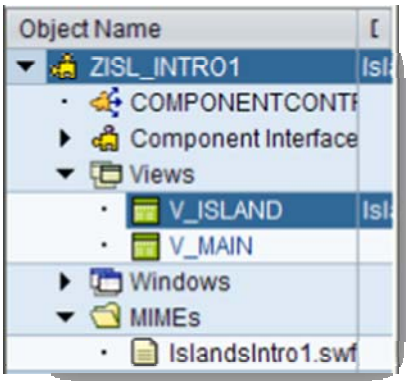
c. Choose the completed Flex Component file from the Flex Builder project workspace.



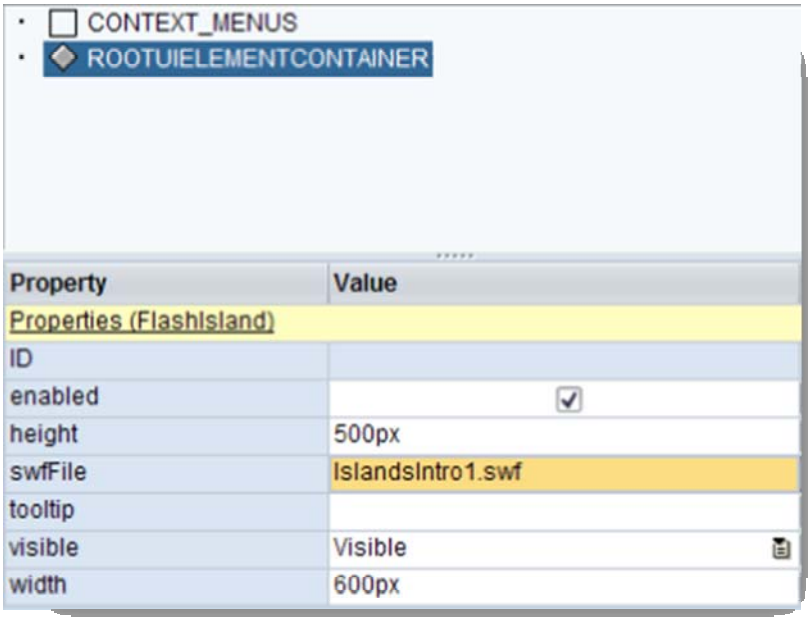
d. Save the Mime Object with the default Document Settings



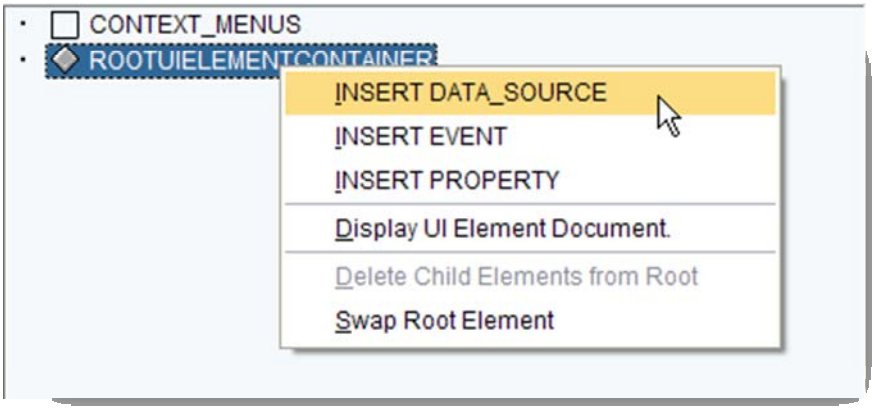
e. The Flex Component is now displayed within the **MIMEs** folder of the Web Dynpro Component and transported with the Web Dynpro development object.



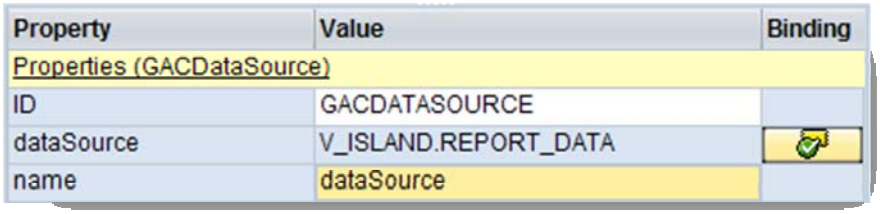
f. Click on the ROOTUIELEMENTCONTAINER and you will see via its Properties that it now is a FlashIsland. The main property is the swfFile – which tells Web Dynpro which Adobe Flash/Flex component should be loaded in this space. Supply the following properties:



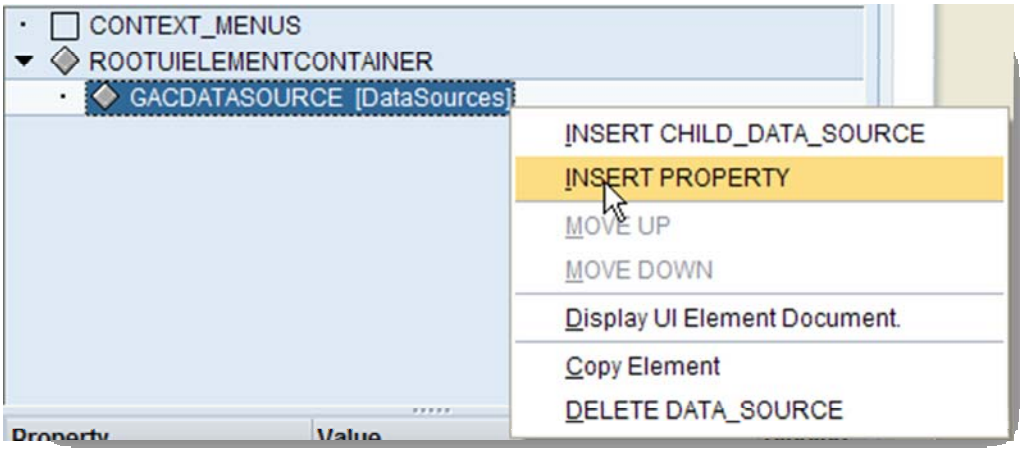
g. Now we need to map our Context Node **REPORT_DATA** to the Flex Component DataSource named **dataSource**. To do this add a child element of the **ROOTUIELEMENTCONTAINER** of type **DataSources**.



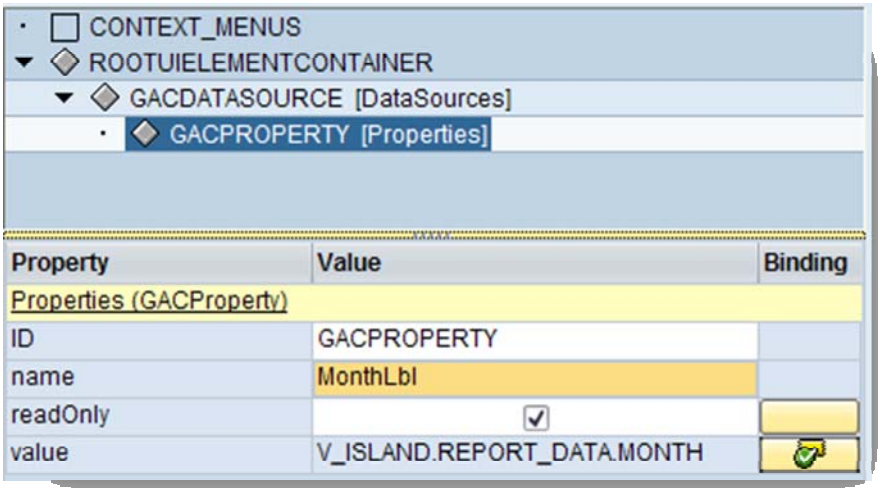
h. Then context bind the **dataSource** property and the supply the Flex Component name in the **name** property.



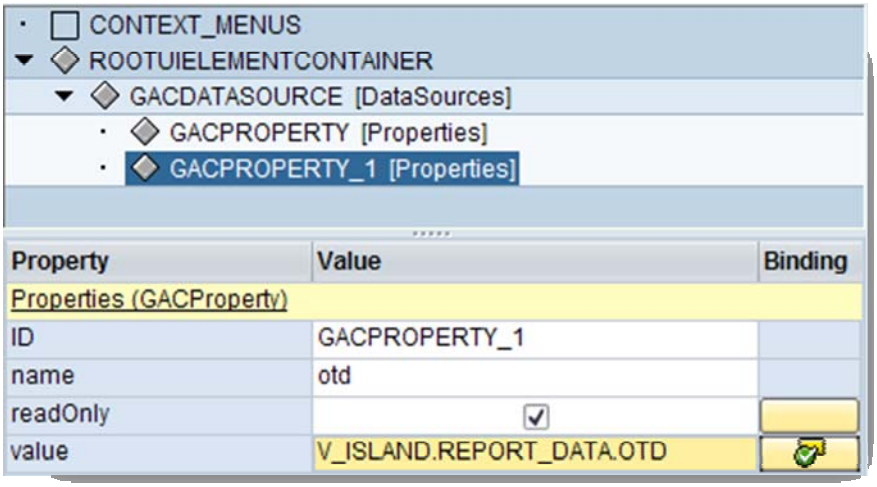
i. Just like we mapped the name of the Context Node to the Flex dataSource, we must also map the names of the Context Attributes to Flex property names. Right mouse click on the **GACDataSource** in the UI element hierarchy and choose *Insert Property*.



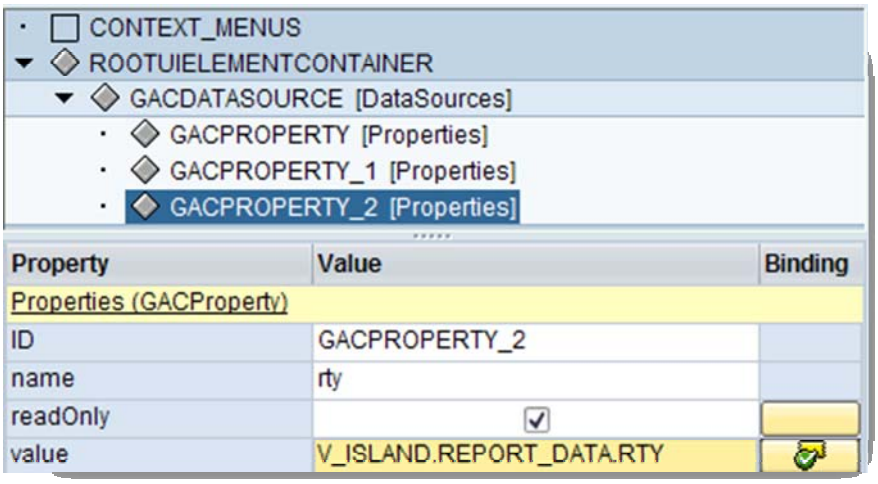
j. The first property is for MONTH = MonthLbl.



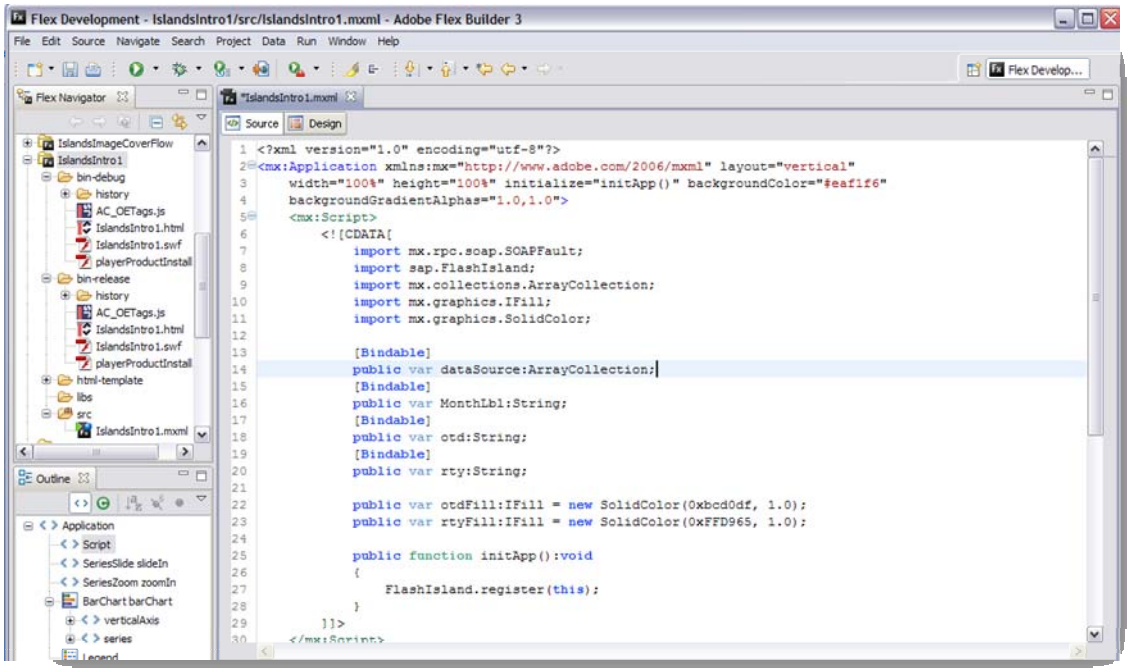
k. The second property is for OTD = otd



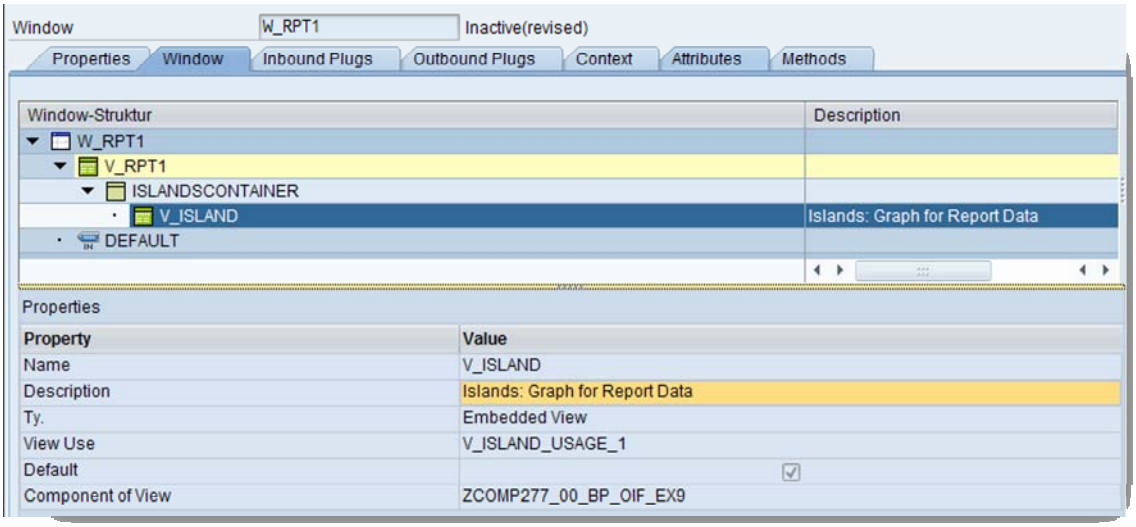
l. The final property is for RTY = rty



m. You might be wondering how we knew the Flex property and dataSet names. Well if you didn't have access to the Flex source code, you would have to ask the original developer. Here see the source code within the Adobe Flex Builder.

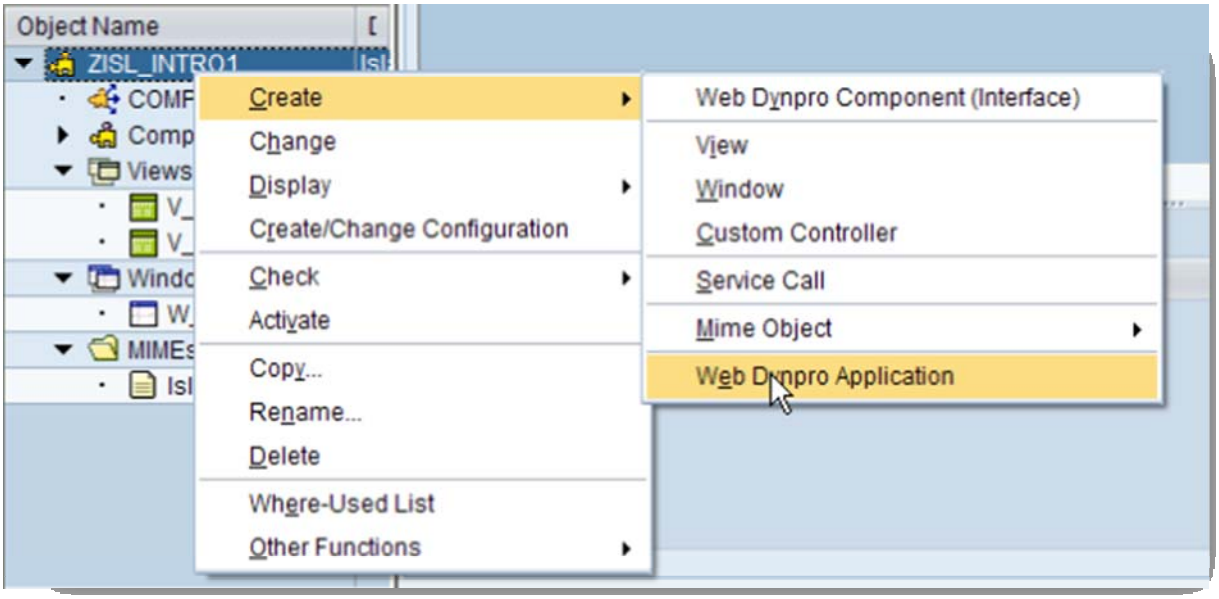


4. Embed the Islands View within the Window/View Container
   a. Drag and Drop the V_ISLAND View to the ISLANDSCONTAINER within V_RPT1.
   b. The final association should look like this.

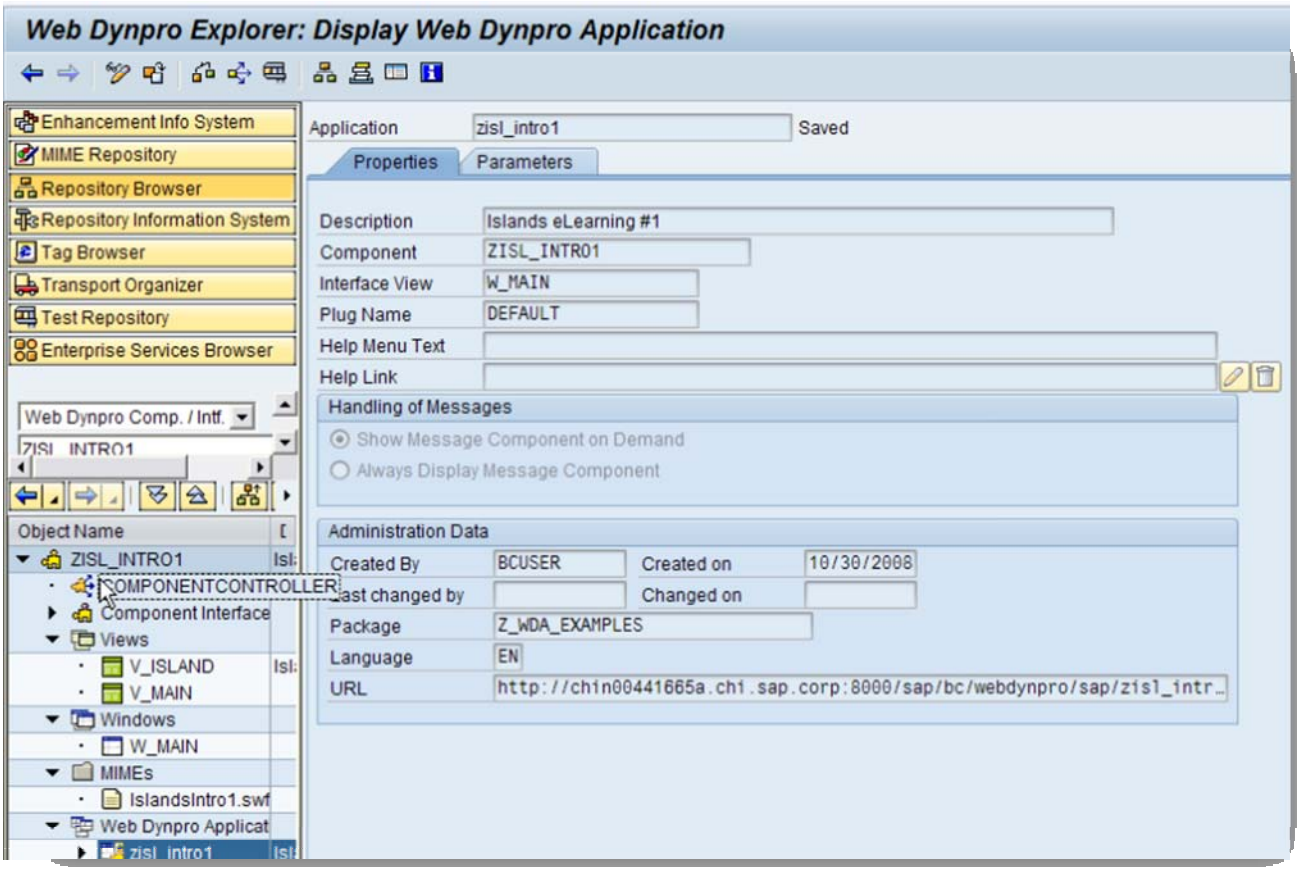


Completing and Running the Web Dynpro Application

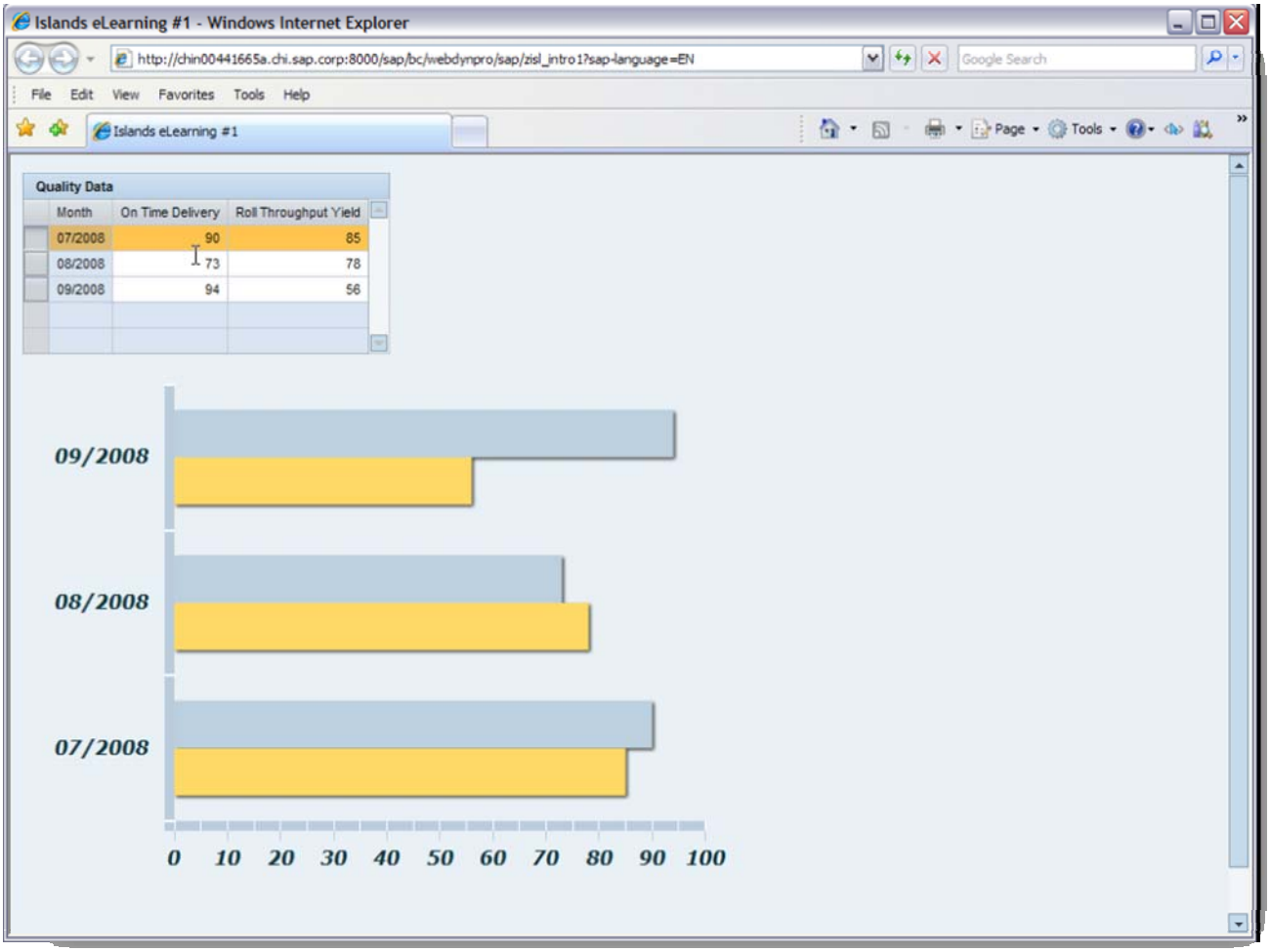
1. Right Mouse Click on the Web Dynpro Component and choose Create->Web Dynpro Application.



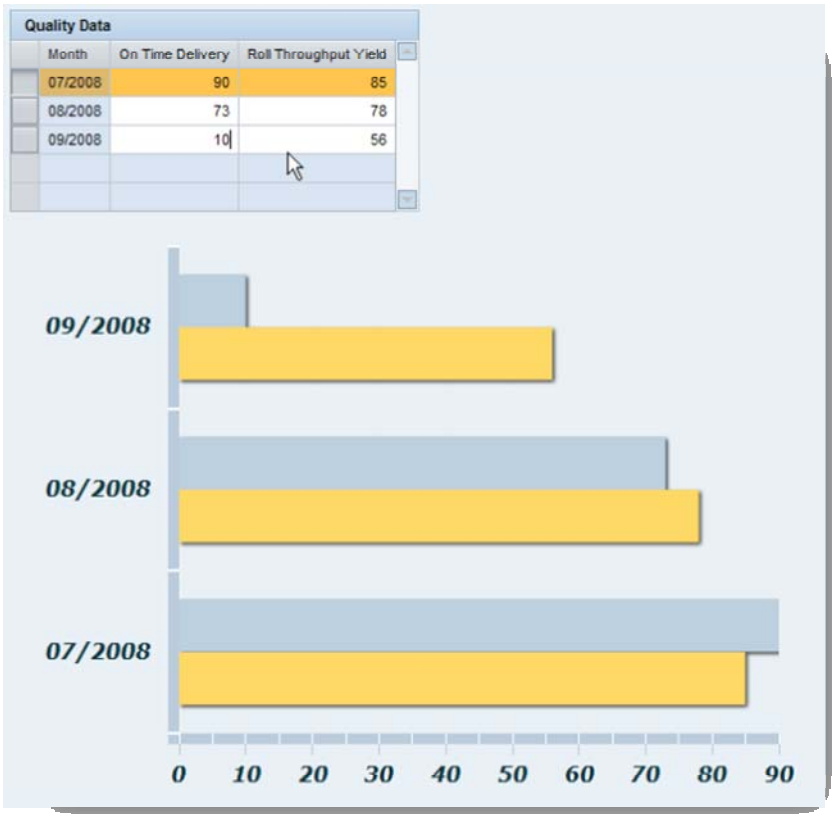
2. Creating the application will generate a URL from which we can test our application.



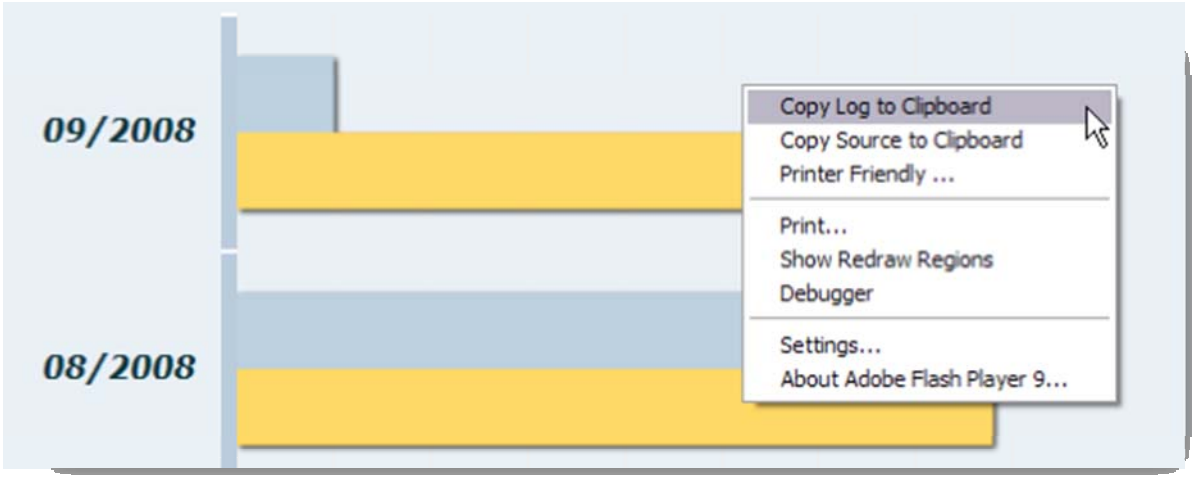
3. Hit F8 or press the Text/Execute button to launch the application in your default Web Browser. You should see the data in the Table and in the Flex Component Bar Chart



4. Try changing some of the data in the Table and see the changes reflected in the chart.



5. If you need to do any trouble shooting trying using the Right Mouse Clicks within the FlashIsland area and choose either Copy Log to Clipboard or Copy Source to Clipboard. This will allow you to view the FlashIsland processing log or the xBCML data feed that passes between Web Dynpro ABAP and the FlashIslands Component.



Related Content

Adobe Flash Islands for Web Dynpro WIKI Page

Web Dynpro ABAP Page on SDN

Web Dynpro ABAP WIKI

Copyright