Accessing ABAP Functions in Web Dynpro Java

Applies to:

Web Dynpro Java in SAP NetWeaver 7.0 (2004s)

Summary

This tutorial shows how to use the Adaptive RFC model to connect to a SAP backend.

- Level of complexity: Intermediate
- Time required for completion: 60 min.

Author: SAP NetWeaver Product Management

Company: SAP AG

Created on: 08 December 2007

Table of Contents

Sample Projects

To follow the tutorial step by step, download the initial project here:

- WDJ_Tut_AdaptiveRFC_70_Init.zip

To check the ready-to-use tutorial, download this project:

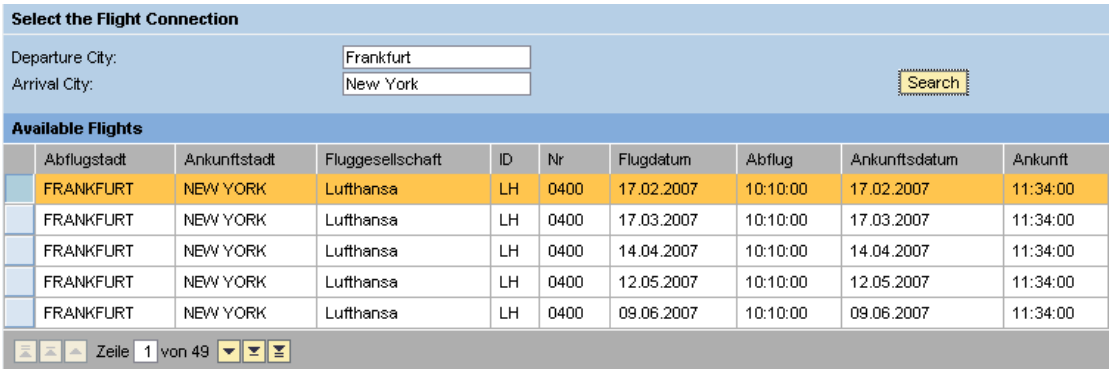
- WDJ_Tut_AdaptiveRFC_70.zip (some configuration steps have to be performed nevertheless)

Creating a Web Dynpro Application Accessing ABAP Functions

The following tutorial shows you how to design, implement, deploy, and run a basic Web Dynpro application that accesses persistent data from a remote SAP system.

In the Web Dynpro application, you will connect to the remote SAP system, the backend, using the **Adaptive RFC model**. To access database tables, you can make use of existing functions in the form of RFC function modules. For each function module that you need, the system generates a corresponding Java proxy class. All the generated proxy classes and interface are bundled together in the RFC model and treated as part of your Web Dynpro project.

For the purposes of this tutorial, you design a simple, structured Web application, which will display flight connections between a given departure and destination airport. The user interface for this Web application will consist of two views. In the first view, the user should be able to enter the departure and destination airports in the appropriate input fields and trigger display of the flight data using a *Search* button. As a result of this query, all the available flight data will be displayed as a table in the next view.



Objectives

By the end of this tutorial, you will be able to:

- Create a model that is used to connect to the SAP backend from the Web Dynpro project
- Implement access to remote function modules in an SAP system
- Implement custom controllers for specific tasks that cannot be assigned to a single view
- Create contexts for the custom controller and bind them to the model
- Create view contexts and map them to the custom controller context
- Bind UI controls to view context elements

Prerequisites

Systems and Authorizations

- The SAP NetWeaver Developer Studio is installed on your computer.
- You have access to the SAP Web Application Server
- You have access to a remote SAP back-end system. To test this example application successfully, you must also make sure that this SAP system contains the function module BAPI_FLIGHT_GETLIST and the appropriate data from the flight data model. To access functions within a SAP system, a user must provide this system with valid credentials by means of the logon process. Since you are using the Web Dynpro Adaptive RFC Layer, the user ID defined in the Web Dynpro Content Administrator of your SAP NetWeaver Application Server will take care of the connection automatically.
- The SAP System Landscape Directory (SLD) and the SLD bridge are configured and running. The SLD contains component information about all SAP software modules, the system landscape description and the name reservation service. The SLD bridge is used to transform the system data to the SLD server compliant format.
- For more information, see also the Post-Installation Guide for the System Landscape Directory of SAP NetWeaver 7.0 or have a look at the documentation regarding SAP System Landscape Directory (requires SMP Login).

In order to connect the logical systems defined in the Adaptive RFC model with a physical SAP system you need to perform two configuration tasks.

Setting Up the SLD Connection

To establish a connection to the SLD server you have to specify the HTTP connection parameters. You will carry out this task in the *Visual Administrator* while the SAP NetWeaver Application Server is running.

Proceed as follows:

1. Start the SAP NetWeaver Application Server if you have not already done so.
2. Start the Visual Administrator using the path <Drive>:\usr\sap\<System ID>\JC00\j2ee\admin\go.bat.
3. Choose Connect and log on to the SAP NetWeaver Application Server. You need administrator rights for the login process.
4. After login, choose the Cluster tab on the left, expand the node Server, Services, and choose SLD Data Supplier.
5. Choose CIM Client Generation Settings and specify the necessary HTTP connection parameters to establish a connection to the SLD.
6. Save the data by pressing the Save button
7. Choose the CIM Client Test button to check the CIM client connectivity.
8. If the test was successful, you can close the Visual Administrator console. Otherwise correct your connection parameters.

   Now you have defined the connection parameters for SLD server has to be used to run your application.

Maintaining the JCO Destinations in the Web Dynpro Content Administrator

The logical system names used in the model declaration need to be associated with an actual SAP system defined in the SLD before this application can be executed. You will carry out this task after deploying your example application in the Web Dynpro *Content Administrator*.

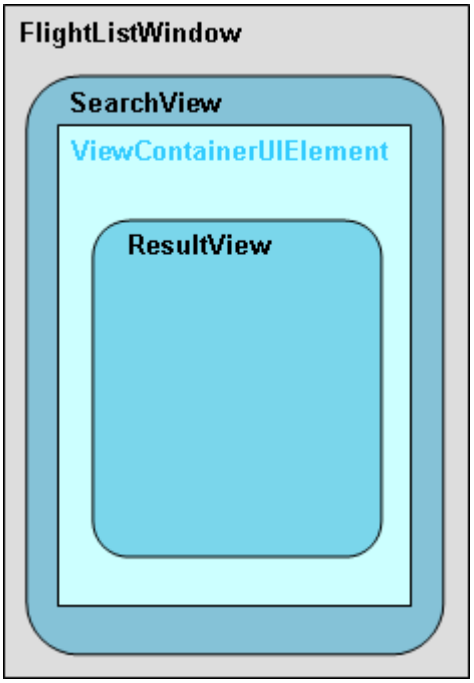
If you do not perform this step, then you will only be able to build and deploy the application, but you will be unable to run it.

Importing the Project Template into the SAP NetWeaver Developer Studio

1. Download the ZIP file *WDJ_Tut_AdaptiveRFC_70_Init.zip*, which contains the initial Web Dynpro project *TutWD_FlightList_Init* and save it in a local directory or directly in the workspace of the SAP NetWeaver Developer Studio.
2. Unzip the contents of the ZIP file *WDJ_Tut_AdaptiveRFC_70_Init.zip* into the workspace of the SAP NetWeaver Developer Studio or in local directory.
3. Call the SAP NetWeaver Developer Studio.
4. Import the Web Dynpro project *TutWD_FlightList_Init.*

   The Web Dynpro project *TutWD_FlightList_Init* then appears in the Web Dynpro Explorer for further processing and editing in the context of this tutorial. The warnings triggered by the Web Dynpro project *TutWD_FlightList_Init* can be ignored at this time, since we will extend the Web Dynpro project during the remainder of this tutorial and will thus remove the displayed warnings. Without the additions we will make, the initial Web Dynpro project template *TutWD_FlightList_Init* cannot be executed.

In the Navigation Modeler, the initial Web Dynpro project looks as follows: The *FlightListWindow* contains a *SearchView*, which inherits a *ViewContainerUIElement* embedding the *ResultView*.



Developing the Application – Steps

The following description of how to extend the example application with access to ABAP functions is split into two sections:

1. Creating a model and context elements and bind them:
   a. Add a new *FlightModel* that is based on the Adaptive RFC.
   b. In the custom controller *FlightListCust*, create an appropriate context structure and bind it to the model *FlightModel*.
   c. Create appropriate context structures in the views *SearchView* and *ResultView* and map them to the context structures of the custom controller *FlightListCust*.
2. Creating an action, declaring a method, and adding the implementation of the back-end connection:
   a. Define an action *Search* in the *SearchView* that will trigger the query to the back-end system. By declaring a method in the custom controller *FlightListCust* the query is sent to the back-end system.
   b. You bind the UI elements to the context structures or the action.
   c. As a last step before building, deploying, and executing, you must implement the back-end call.

Creating a Model

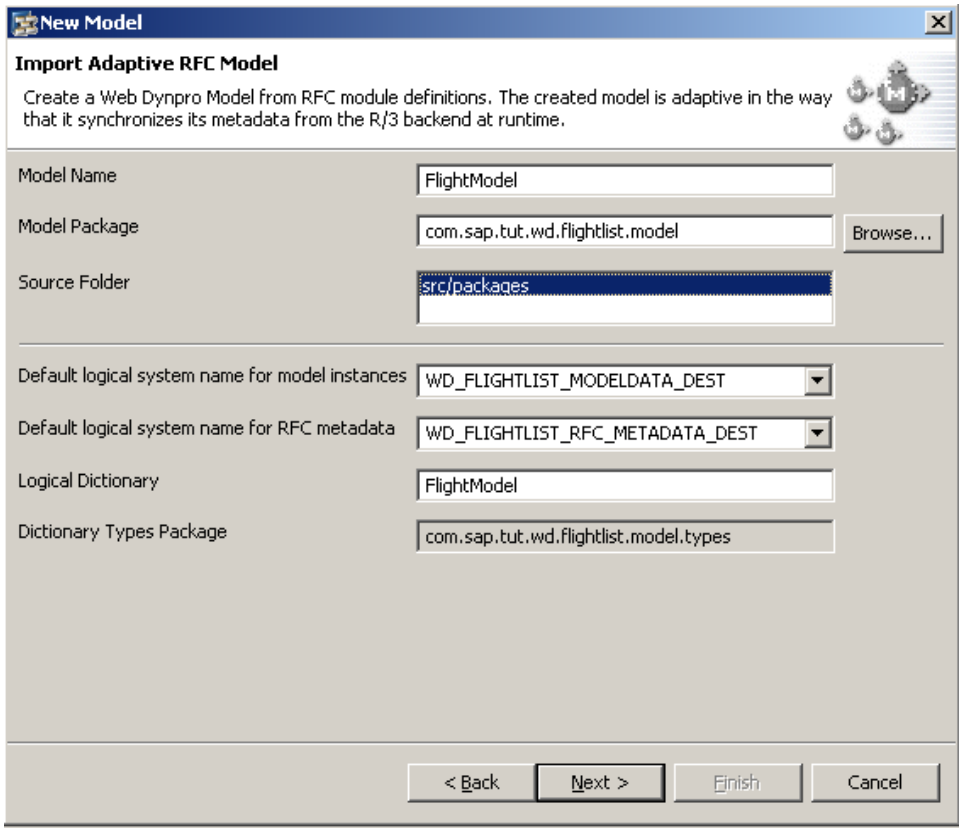
You use a model to specify where the data for your Web Dynpro application is to come from. Since you are using a remote SAP system to retrieve the flight data, you can make use of an existing model. You will import an adaptive RFC model into your project for this purpose. This step also includes in particular generating Java proxies, which represent the corresponding RFC function modules in your Web Dynpro project.

1. In the project structure, expand the node *Web Dynpro* → *Models*.
2. From the context menu, choose  *Create Model*.
   The appropriate wizard appears.
3. Choose the *Import Adaptive RFC Model* option, followed by *Next*.
4. Enter the model name **`FlightModel`** and the package name **`com.sap.tut.wd.flightlist.model`**.

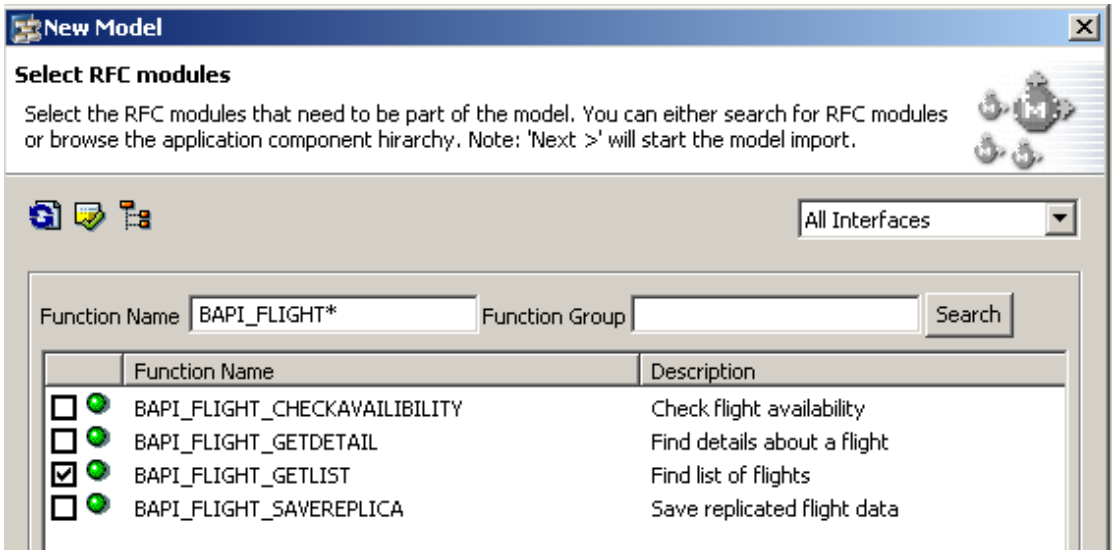
   When importing an adaptive RFC Model you have to specify the logical system names for model instances and RFC metadata:

   - Change the default logical system name for model instances `WD_MODELDATA_DEST` to **`WD_FLIGHTLIST_MODELDATA_DEST`**
   - Change the default logical system name for RFC metadata `WD_RFC_METADATA_DEST` to **`WD_FLIGHTLIST_RFC_METADATA_DEST`**

   The logical systems will later be separately configured using the JCO Connections screen of the Web Dynpro Content Administrator (see section Building, Deploying, Configuring, and Running Your Application).

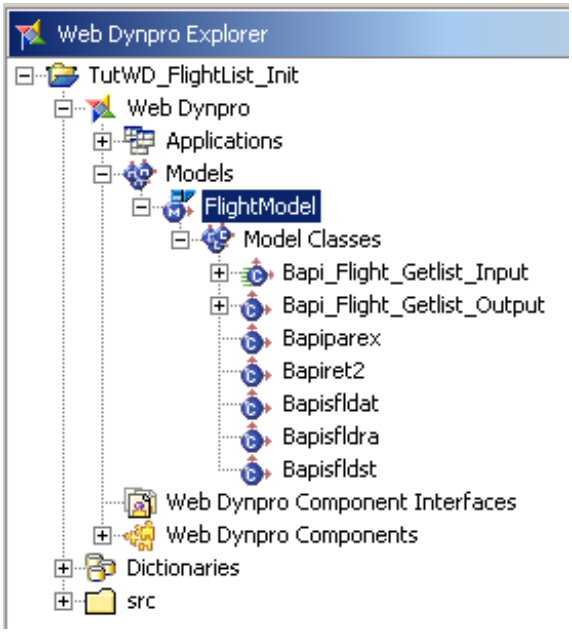


5. Accept the suggested values and choose *Next*.
6. Enter the appropriate data for logging on to the SAP system and choose *Next*.
7. Enter either the complete name of the function module **`BAPI_FLIGHT_GETLIST`** in the appropriate field or enter the start of the name followed by an asterisk (*****). Then choose *Search.*
8. Select the function module **`BAPI_FLIGHT_GETLIST`** from the list that appears.



9. Choose *Next*. By doing so, you automatically trigger the generation process. The import process is logged by a detailed description, which you can see in the next dialogue. To confirm choose *Finish.*

10. Now the java proxies are generated and a new model node *FlightModel* is inserted into project structure.



Creating the Context and Binding it to the Model

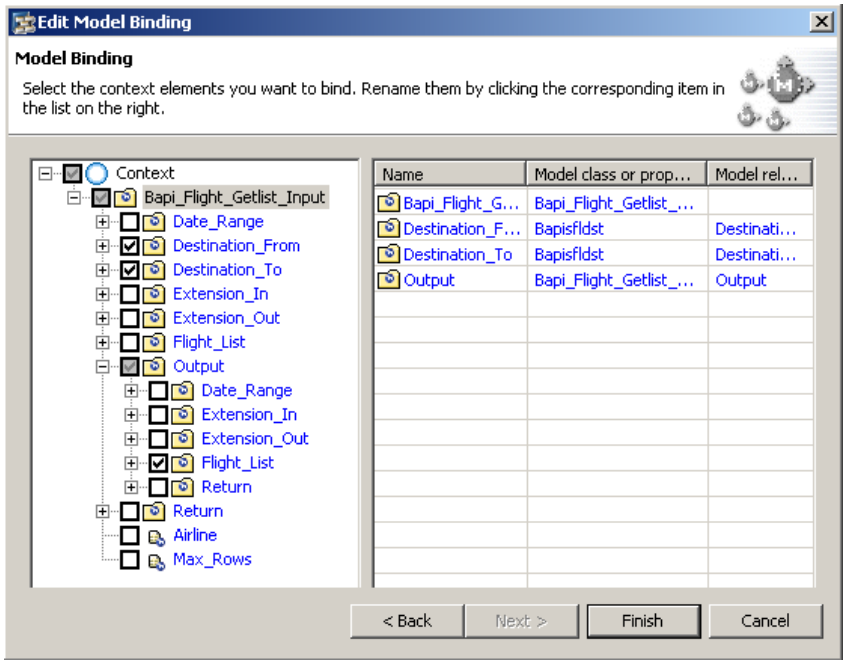
The custom controller *FlightListCust* is responsible for retrieving flight data from an SAP system, so it needs to be able to map the corresponding input and output for the flight model. To establish this correspondence between the custom controller and the model, you will create an appropriate controller context and then bind the context nodes to the model structure. In this way, you can ensure that the model data is stored and manipulated in a central location.

Adding a model to the Web Dynpro component

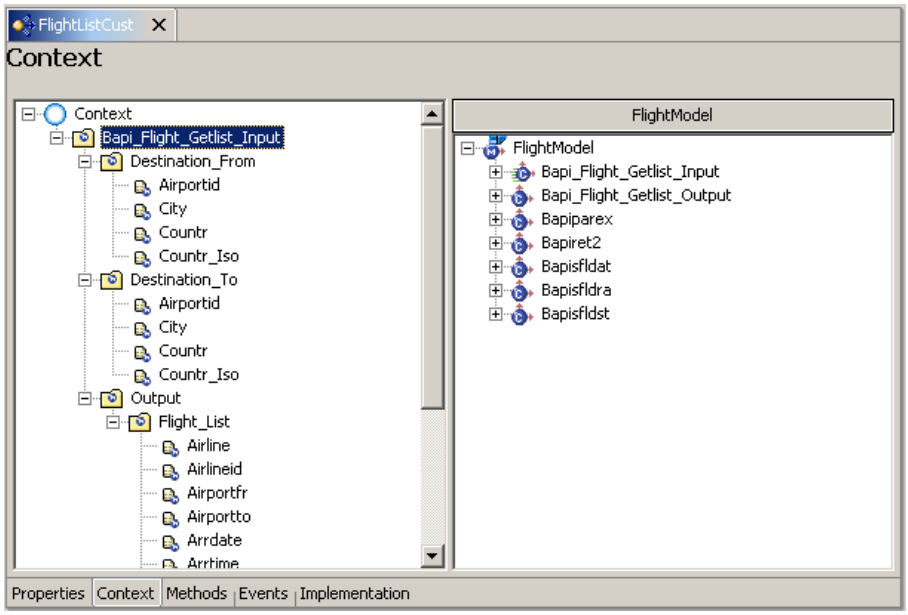
1. In the project structure, expand the tree up to the node *Web Dynpro* → *Web Dynpro Components* → *FlightListComp*.
2. Select the node *Used Models,* open the context menu and choose *Add*.
3. In the list that appears, select the model **`FlightModel`** and confirm by choosing *OK*. By doing so, you specify that all views and controllers of *FlightListComp* have a dependency relationship with the *FlightModel*.

Creating a Context for the Custom Controller

1. In the project structure, double-click the name of the custom controller (in this case, **`FlightListCust`**).
2. Choose the *Context* tab if necessary.
3. Open the context menu for the root node *Context* and choose the option *New* → *Model Node*.
4. Enter the name **`Bapi_Flight_Getlist_Input`** for the model node and choose *Finish*.
5. From the context menu for the model node that you have just created, choose *Edit Model Binding*…
6. Choose the model class **`Bapi_Flight_Getlist_Input`**, followed by *Next*.
7. Activate the following entries: **`Destination_From`** and **`Destination_To`**. Expand **`Output`** and activate **`Flight_List`**. Choose *Finish*.



The Developer Studio refreshes the context tree appropriately.



In this way, you have completed specification of all the context nodes for the input and output data. You will need this context later for the *SearchView* and the *ResultView*.

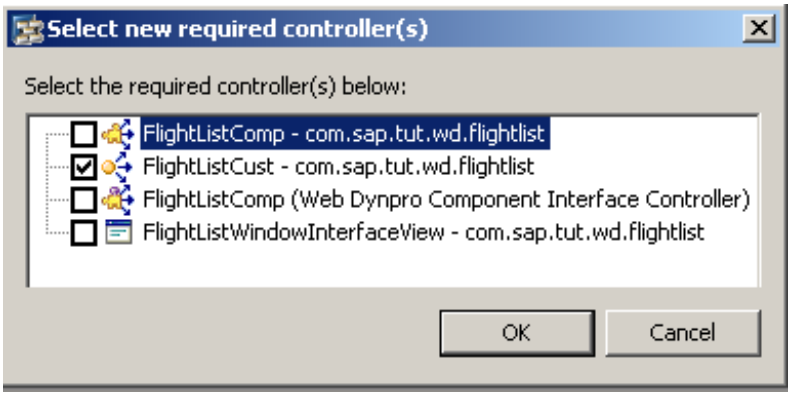
8. Save your work by choosing the icon (*Save All Metadata)* from the toolbar.

Mapping Custom Context Elements to View Context Elements

In this step you will learn how to map context elements of the custom controller *FlightListCust* to the appropriate context elements of the views *SearchView* and *ResultView.*

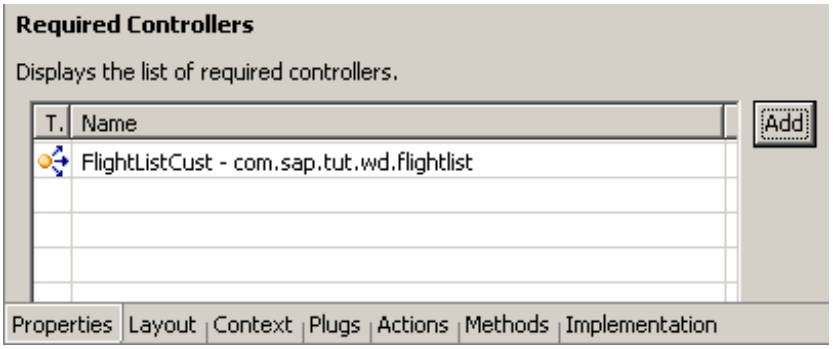
Adding Dependencies to the Views

1. Navigate to **`SearchView`** (*Web Dynpro* → *Web Dynpro Components* → *FlightListComp* → *Views* → *SearchView*) and open the *ViewDesigner* with secondary mouse button.
2. Choose the *Properties* tab.
3. Under *Required Controllers*, choose *Add*.
4. In the list that appears, choose the **`FlightListCust`** component.



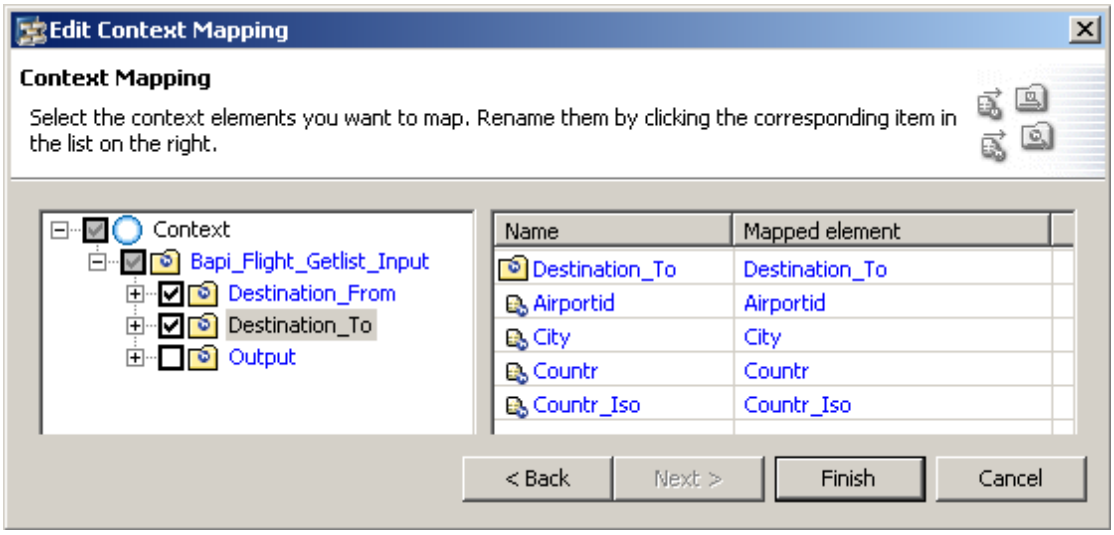
5. Confirm by choosing *OK*.
6. Repeat steps 1-5 as appropriate for the **`ResultView`** view.

   You have now entered the appropriate dependency on the custom controller **`FlightListCust`** for each view.

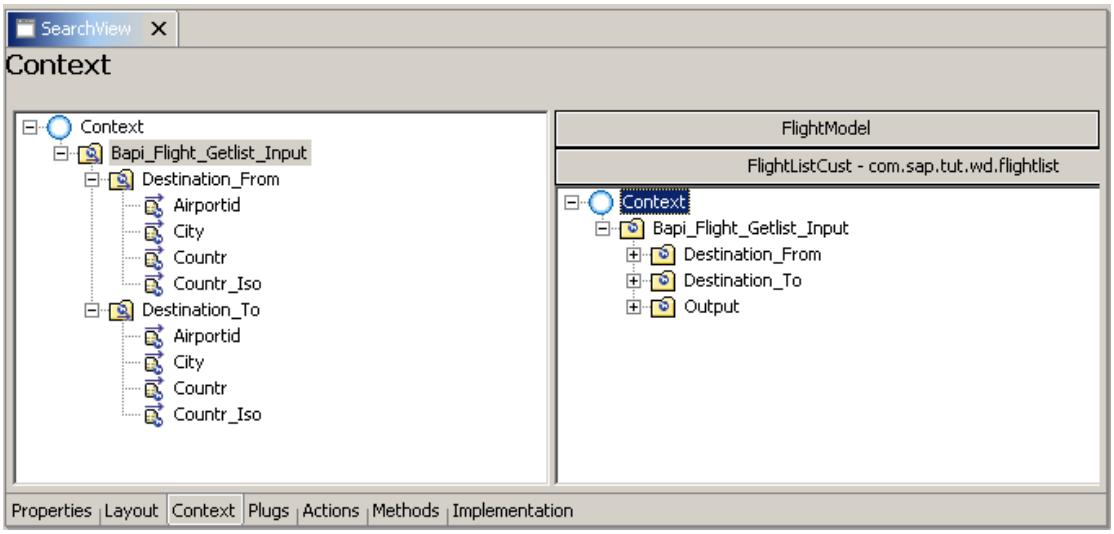


Creating a Context for the SearchView

1. Open the View Designer for the *SearchView* again and choose the *Context* tab.
2. Open the context menu for the root node Context and choose the option *New* → *Model Node.*
3. Enter the name **Bapi_Flight_Getlist_Input** for the model node and choose *Finish*.
4. From the context menu of the model node that you have created, choose *Edit Context Mapping…*
5. Choose the custom context node **Bapi_Flight_Getlist_Input,** followed by *Next*.
6. Activate only **Destination_From** and **Destination_To.** Choose *Finish*.



The Developer Studio refreshes the context tree appropriately.

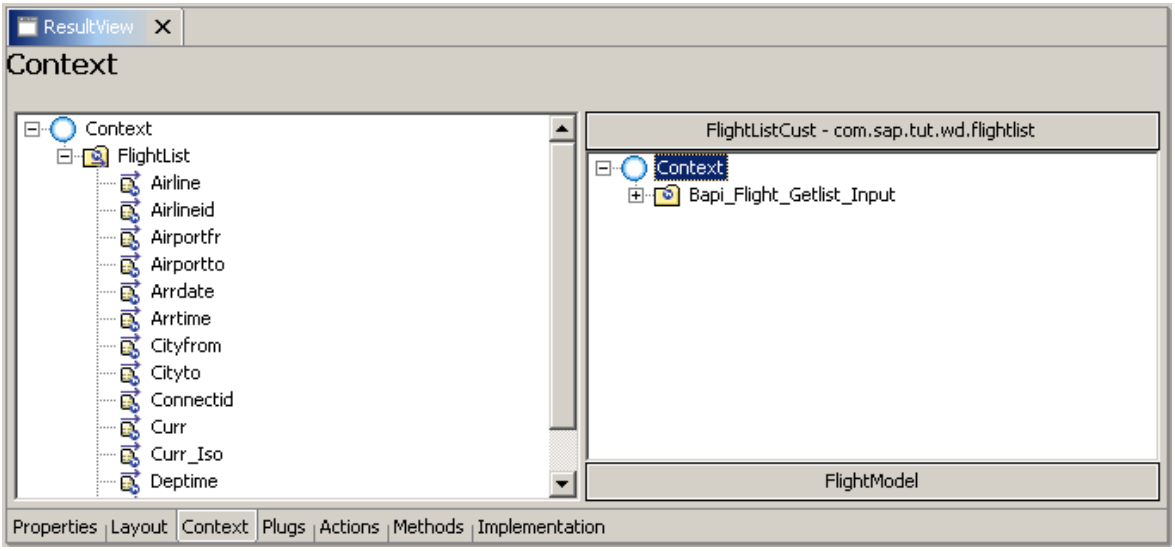


Creating a Context for the ResultView

1. Open the *View Designer* for the **`ResultView`** again.
2. Choose the *Context* tab.
3. Open the context menu for the root node *Context* and choose the option *New* → *Model Node*.
4. Enter the name **`FlightList`** for the model node and choose *Finish*.
5. From the context menu of the model node that you have created, choose *Edit Context Mapping*…
6. Expand the custom context structure **`Bapi_Flight_Getlist_Input`** → **`Output`**, then choose *Next*.

7. Choose the node **`Flight_List`**, then choose *Finish*.

   The Developer Studio refreshes the view context tree appropriately.



8. Save your work by choosing the icon (*Save All Metadata)* from the toolbar

Creating Actions and Declaring Methods

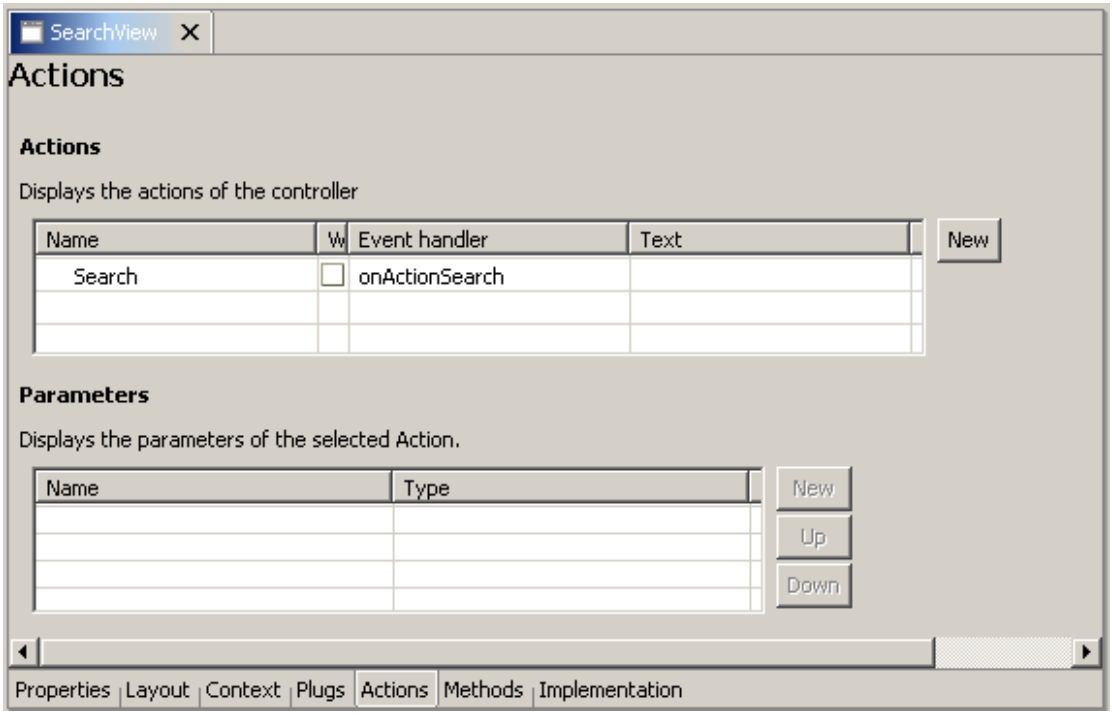
To trigger display of the flight data from the SAP system, in the *SearchView*, you need to create an action that can be bound to a UI element, such as a button. You can then implement the event handler, which reacts to this action.

Creating the Search Action

1. Open the *View Designer* for the **`SearchView`**.
2. Choose the *Actions* tab.
3. Choose the *New* pushbutton.

   You can create a new action in the wizard that appears.
4. Enter the name **`Search`** for this new action, leave the other settings unchanged, and choose *Finish.*

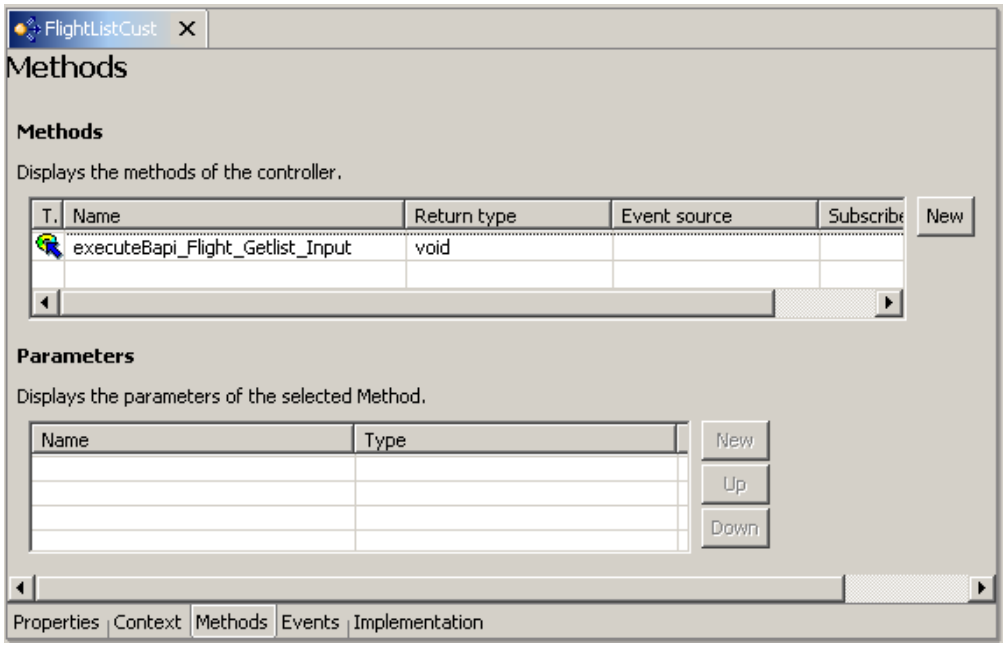
   An event handler, **`onActionSearch`**, is automatically created for this new action.



Declaring the executeBapi_Flight_Getlist_Input() method

1. Open the editor for the custom controller **`FlightListCust`** again.
2. Choose the *Methods* tab.
3. Choose *New.*
4. Select the *Method* option and choose *Next*.
5. In the wizard screen that appears, enter the name **`executeBapi_Flight_Getlist_Input`** for this new method and assign it the return type **`void`**. Choose *Finish*.

   The method **`executeBapi_Flight_Getlist_Input`** is added to the custom controller.



6. Save the new metadata by choosing the icon (*Save All Metadata)* from the toolbar.

Editing the UI Elements

Editing UI Elements for the Search view

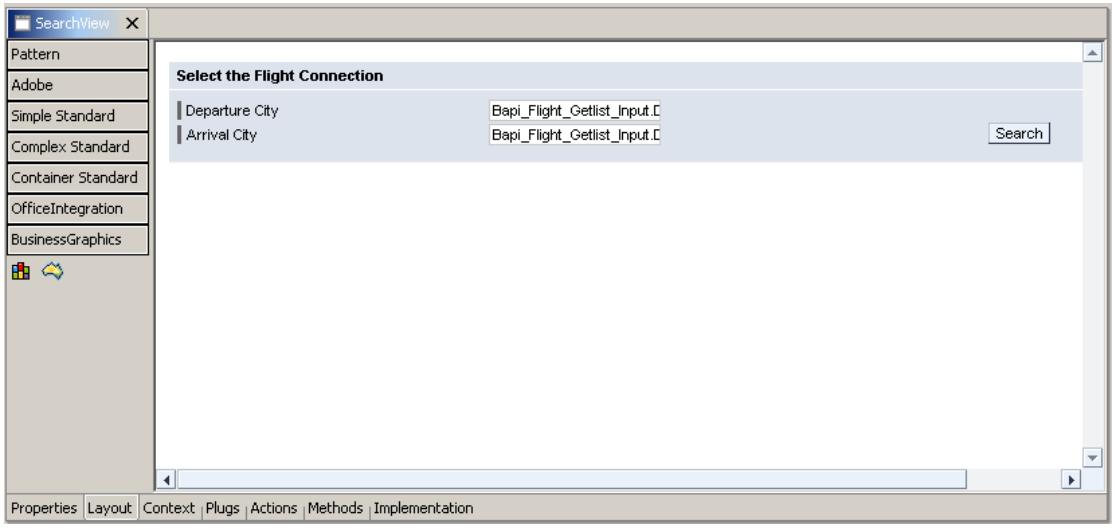
1. Open the **`SearchView`** in the *View Designer* by clicking the *Layout* tab of the View Editor.

   The *View Designer* displays a predefined default text. Simultaneously, the *Outline* view displays a list of the UI elements included. If you select an element in the *Outline* view or on the *Layout* tab, its associated element properties are shown in the *Properties* view.

2. Choose following UI elements that have been included in the project template and give them following properties:

Property	Value
For **`FromCityInputField`**	
Value	Bapi_Flight_Getlist_Input.Destination_From.City
For **`ToCityInputField`**	
Value	Bapi_Flight_Getlist_Input.Destination_To.City
For **`SearchButton`**	
Event > onAction	Search

   The View Designer displays the following layout for the SearchView:



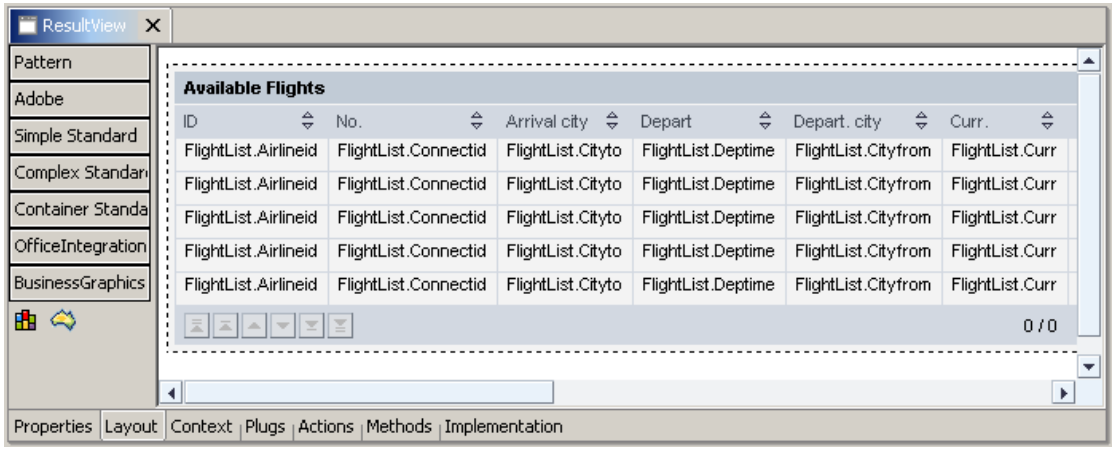
3. Save the new metadata by choosing the icon (*Save All Metadata)* from the toolbar.

Editing UI Elements for the ResultView

1. Open the **`ResultsView`** in the *View Designer* by clicking the *Layout* tab.
2. Select the **`Table`** on the *Outline* tab and choose *Create Binding* from the context menu.
3. In the wizard that appears, you can specify table columns that will be displayed in the TableView. Select the *FlightList* model node and press *Next*.
4. On the next screen leave all the other values unchanged and choose *Finish.*

   The value *FlightList* is automatically assigned to the *dataSource* property of the table UI element and all selected table columns are created for the TableView.

   The *View Designer* displays the following layout for the **`ResultView.`**



5. Save the new metadata by choosing the icon (*Save All Metadata)* from the toolbar.

Adding the Implementation of the Backend Connection

Implementing the Action Event Handler

1. In the *View Designer,* choose the *Implementation* tab for the **`SearchView`**.

   The Developer Studio runs several generation routines, and then displays the updated source code for the implementation of the view controller.

2. Insert the following line of code in the **`onActionSearch()`** method:

```
/** declared validating event handler */
public void
onActionSearch(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
  //@@begin onActionSearch(ServerEvent)
  // The event handler onActionSearch triggers the remote function call

wdThis.wdGetFlightListCustController().executeBapi_Flight_Getlist_Input();
  //@@end
}
```

Adding the Implementation for the Custom Controller FlightListCust

1. Open the Controller Editor for the custom controller **`FlightListCust`** again.
2. Choose the *Implementation* tab.
3. In the standard method **`wdDoInit()`**, between `//@@begin wdDoInit()` and `//@@end`, add the following lines of code:

```
/** Hook method called to initialize controller. */
public void wdDoInit()
{
  //@@begin wdDoInit()
  // Create a new element in the Bapi_Flight_Getlist_Input node
  Bapi_Flight_Getlist_Input input = new Bapi_Flight_Getlist_Input();
  wdContext.nodeBapi_Flight_Getlist_Input().bind(input);
  // Create new elements in the Destination_From and Destination_To
nodes
  input.setDestination_From(new Bapisfldst());
  input.setDestination_To(new Bapisfldst());
  //@@end
}
```

4. In the method **`executeBapi_Flight_Getlist_Input()`**, between `//@@begin executeBapi_Flight_Getlist_Input()` and `//@@end`, add the following lines of code:

```
/** declared method */
public void executeBapi_Flight_Getlist_Input( ) {
//@@begin executeBapi_Getlist_Input()
  try
  {
   // Calls remote function module BAPI_FLIGHT_GETLIST
wdContext.currentBapi_Flight_Getlist_InputElement().modelObject().execut
e();
  // Synchronise the data in the context with the data in the model
  wdContext.nodeOutput().invalidate();
  }
  catch (Exception ex)
  {
  // If an exception is thrown, then the stack trace will be printed
   ex.printStackTrace();
  }
  //@@end
}
```

5. To add the import statements, position the cursor anywhere in the Java Editor and choose *Source* → *Organize Imports* from the context menu.

The following `import` statements are added to the source code, without your having to do anything more:

```
//@@begin imports
import com.sap.tut.wd.flightlist.model.Bapi_Flight_Getlist_Input;
import com.sap.tut.wd.flightlist.model.Bapisfldst;
import com.sap.tut.wd.flightlist.wdp.IPrivateFlightListCust;
//@@end
```

6. Save the new metadata by choosing the icon  (*Save All Metadata)* from the toolbar.

Building, Deploying, Configuring and Running Your Application

You have now reached the last stage in the development of you example application. However, some preparation is still essential before you can deploy and run the application successfully on the SAP NetWeaver Application Server. Go through each of the following prerequisites carefully.

Prerequisites

- You have made sure that the relevant ABAP system, which you will be accessing remotely to retrieve the flight data, is currently available and contains flight data.
- You have made sure that the SAP NetWeaver Application Server has been launched.
- You have checked that the configuration settings for the SAP NetWeaver Application Server and for the SDM server are entered correctly in the Developer Studio.
- To check the server settings, choose the menu path *Window* → *Preferences* → *SAP J2EE Engine.*
- The connection parameters for the used SLD are defined in the Visual Administrator.

Building the Project

1. If you have not already done so, save the metadata for your project in its current state.
2. In the *Web Dynpro Explorer*, from the context menu of the project node **`TutWD_FlightList_Init`**, choose  *Rebuild Project*.

Deploying the Project

3. In the *Web Dynpro Explorer*, select the project node **`TutWD_FlightList_Init`** and choose  *Create Archive* from the context menu.
4. Choose  *Deploy Project* from the context menu of the project node.

Creating the JCO Connections in the Content Administrator

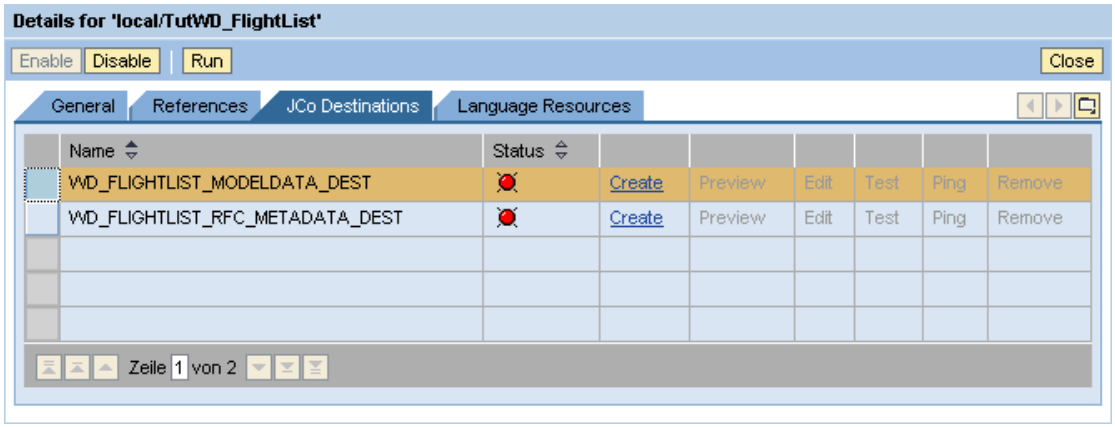
The logical system names used in the model declaration must be associated with an actual SAP system defined in the SLD before this application can be executed. For this reason you need to create JCO connections in the Web Dynpro *Content Administrator*. To use an adaptive RFC model in your Web Dynpro application you will define two connections:

- A connection to get the needed (dictionary) meta data information
- A connection to get the read the application data

To create the new JCO connections proceed as follows:

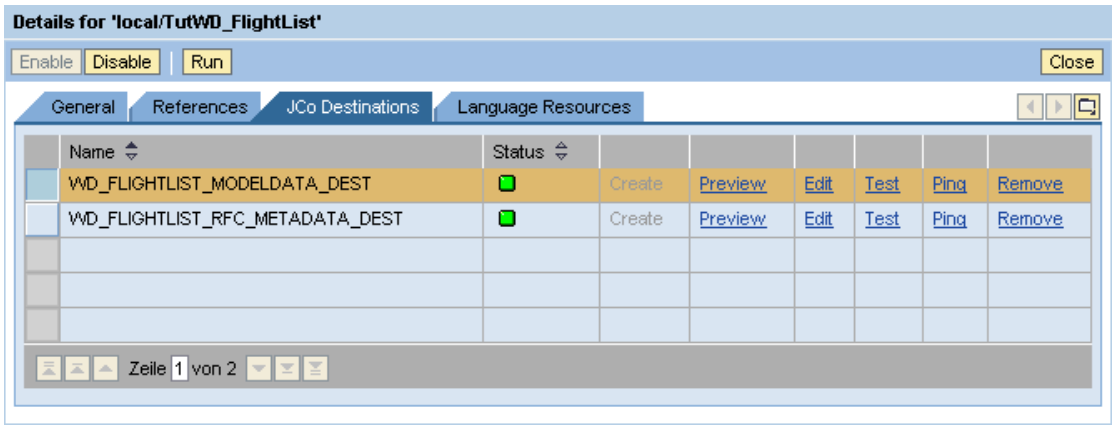
1. Open the Web Dynpro *Content Administrator* on your SAP NetWeaver Application Server using the URL: `http://<host>:<port>/webdynpro/welcome`.
2. If you are using the *Content Administrator* for the first time, you need to perform the self registration.
3. Select the *Browse* tab and navigate to the application node *local*→ *TutWD_FlightList_Init*→ *Applications*→ *FlightListApp.*
4. Choose the *JCO Connections* tab.

   Two destinations named **`WD_FLIGHTLIST_MODELDATA_DEST`** and **`WD_FLIGHTLIST_RFC_METADATA_DEST`** are shown in this view and both of them have the status red. This means that the JCO connection is not yet maintained in assigned SLD.



5. Select **`WD_FLIGHTLIST_MODELDATA_DEST`** and choose *Create* to configure a new JCO connection.
   a. Go though the steps of the JCO connection creation wizard and specify the connection data.
   b. After making the required entries, press *Finish*.
   c. Test your configuration data by pressing the *Test* button.
6. Do the same for **`WD_FLIGHTLIST_RFC_METADATA_DEST`**.

   The both destinations are ready to use.



7. If the tests were successful, you can close the Content Administrator. Otherwise correct your connection entries.

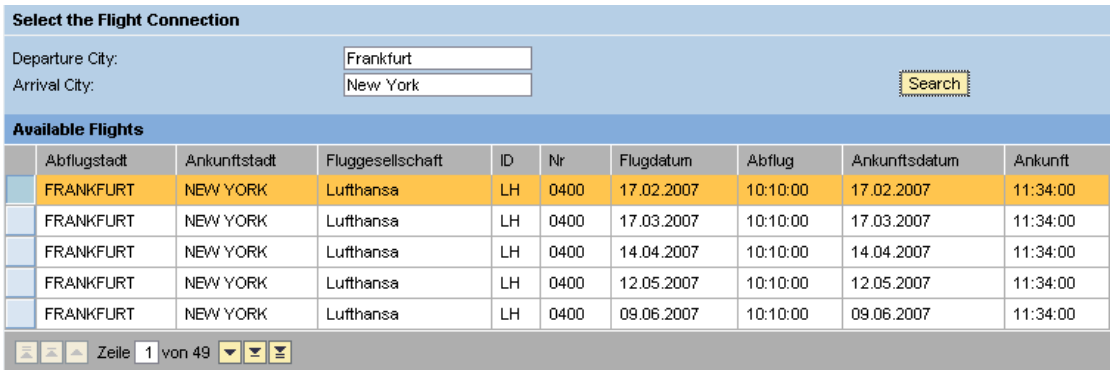
Launching the Application

8. In the Web Dynpro Explorer, open the context menu for the application object **`FlightListApp`**.
9. Choose *Run*.

Result

The Developer Studio performs the deployment process and then automatically launches your application in the Web browser.

Test your Web Dynpro application by entering a valid city name for the *Departure City* and *Arrival City* and then clicking the *Search* button. Provided the system contains the appropriate flight data, it will display it in a table.



Related Content

- Tutorials & Samples for Web Dynpro Java
- Web Dynpro Models: Backend Access
- Importing an Adaptive RFC Model

Copyright