Using an Adaptive RFC 2 Model in Web Dynpro Java

Applies to:

Web Dynpro for Java 7.11. For more information, visit the Web Dynpro Java homepage.

Summary

This tutorial is aimed at developers already familiar with versions of Web Dynpro Java earlier than 7.1 and in which you have written code that calls ABAP function modules through the Adaptive RFC interface.

The older version of the Adaptive RFC interface (known as ARFC1) has now been deprecated and replaced with a new version (known as ARFC2).

ARFC1 is still supported for reasons of backward compatibility; however, for all new Web Dynpro Java applications that need to invoke ABAP based functionality, SAP strongly recommends that the new ARFC2 interface is used instead of ARFC1.

This document describes the following topics:

- Why it was necessary to replace ARFC1
- The BAPI calls made during the execution of a simple demo application
- How to create ARFC2 destinations in the NetWeaver Administrator
- How to migrate ARFC1 models to the new structure used by ARFC2
- The coding differences between ARFC1 and ARFC2
- The coding needed to invoke ABAP function modules using the ARFC2 interface

Author: Web Dynpro Java Team

Company: SAP AG

Created on: 29June 2010

© 2010 SAP AG

Table of Contents

© 2010 SAP AG

Introduction

In this tutorial you will learn about Adaptive RFC Version 2 model objects. This type of model object is used by software outside an ABAP system to invoke remote callable ABAP function modules within the ABAP system.

The coding API is very similar to that used by ARFC1, but there are a number of restrictions that may require certain sections of coding in older Web Dynpro applications to be rewritten.

Background

The reason for a new version of the Adaptive RFC (ARFC2) model interface was to overcome the shortcomings present in the previous version (known as ARFC1). A few of these were:

- ARFC1 made use of the various version 2.x implementations of the Java Connector layer. These versions of JCo have now all been deprecated.
- ARFC1 made use of the `aii` proxy generation framework layer that in turn, also used JCo 2.x. The use of `aii` became a restriction and therefore has been removed.
- The architecture of ARFC1 did not allow any central monitoring of open JCo connections.
- No ability to rename generated model classes
- No support for the new ABAP data type of DECFLOAT.

One of the key factors here is that the ARFC1 layer made use of an intermediate proxy generation layer known as `aii`. In turn, `aii` was dependent upon the underlying JCo 2.x implementation.

The fact that the architecture of ARFC1 was tightly coupled to `aii` was a limiting factor that greatly reduced its flexibility. There were also features in ARFC1 that created a tight coupling to Web Dynpro Java: for instance, the lifespan of the JCo connection pool used by the model was tied directly to the lifespan of the Web Dynpro application (`WDModelScopeType.APPLICATION_SCOPE`).

Therefore, it was necessary to dispense both with the `aii` layer and ARFC1's tight coupling to the Web Dynpro architecture: but making this decision meant that a complete rewrite of the Adaptive RFC layer was required – hence ARFC2.

ARFC2 now interacts directly with the new implementation of JCo (in which the version number has jumped to 7.x).

© 2010 SAP AG

Prerequisites

The NetWeaver Developer Studio (Version 7.11 or later) should be installed and connected to an SAP Java Application Server of the same version or higher. This will allow you to compile and deploy the tutorial application.

Required Units of Software

Software Component: **HM-WDUIDMKTCNT**

Development Components: **tc/wd/tut/model/arfc2**

tc/wd/tut/model/arfc2/models

The exact steps required to install the Software Component are described in a separate document.

Objectives

After working through this tutorial you will be able to:

- Understand how to create an ARFC2 JCo Destination using the NetWeaver Administrator.
- Migrate ARFC1 models in existing applications to ARFC2.
- Understand the key differences between ARFC1 and ARFC2 models and the implications these differences have on older Web Dynpro applications.

The Tutorial Scenario: A Simple Demo

The tutorial application in itself is not particularly remarkable because the use of the ARFC2 interface is not visible on the screen. Therefore, to all intents and purposes, the ARFC2 demo looks exactly like the ARFC1 implementation of the same demo application.

The application allows a user to search for flight information. The functionality is straight forward:

1) The user enters some search criteria for flights departing from one city and arriving at another. The search is implemented by calling the ABAP function module BAPI_FLIGHT_GETLIST.

   For example, flights leaving New York's JFK (code JFK) airport and arriving in Frankfurt, Germany (code FRA).

2) Once a list of flights has been obtained, it is displayed as a table.

3) Selecting a table row causes the flight details to be displayed. This is implemented as a call to BAPI_FLIGHT_GETDETAIL.

The important part of this application is not its visual appearance, but its use of ARFC2 and its architectural coding style. Therefore, if you open the development component **tc/wd/tut/model/arfc2**, you will see that many comments have been added to explain why the coding has been written in that particular manner.

© 2010 SAP AG

Before running the application

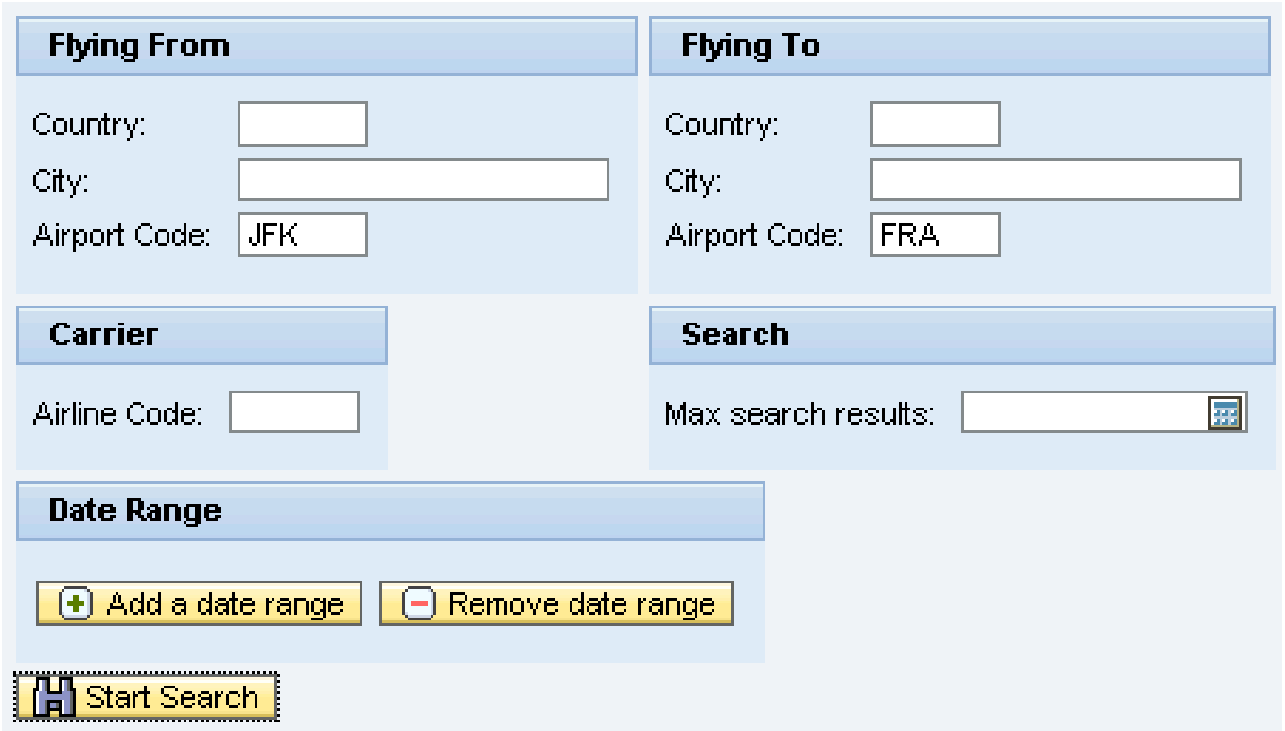
This particular Web Dynpro application requires the use of two JCo Destinations. These destinations must be created before you attempt to run the application. Having said, it's a useful exercise to run a Web Dynpro application ***before*** the JCo Destinations have been created simply to see what type of stack trace you get.

In order to avoid repeated stack traces, please follow the instruction in the chapter on Creating JCo Destinations for ARFC2 on page 7. Once you have done this the application will at least be executable.

Please choose a backend ABAP system to which you have a userid and password. You must also ensure that this system contains the ABAP function modules BAPI_FLIGHT_GETLIST and BAPI_FLIGHT_GETDETAIL. Use transaction SE37 to check for the existence of these function modules.

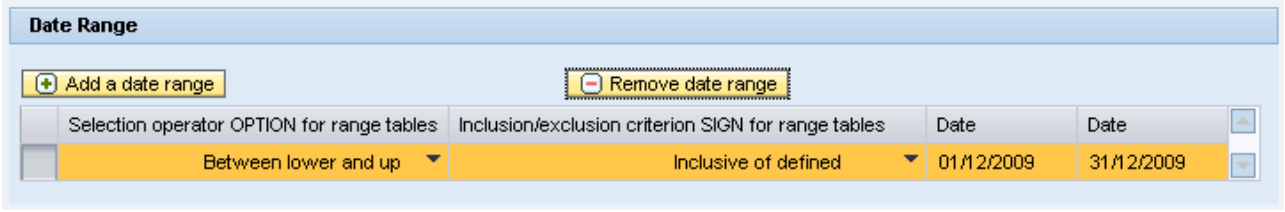
Running the application

When the application starts, you will see the initial search screen:



The screen shot shows that the airport codes "JFK" (John F. Kennedy Airport in New York) and "FRA" (Frankfurt Main Airport, Germany) have been entered.

At this point, you could simply press the "Start Search" button to look for all flights between New York and Frankfurt. However, you could also narrow the date range by searching only for flights in a certain month. If you wish to add a date range, press the "Add a date range" button and a new table will be displayed containing a single row.

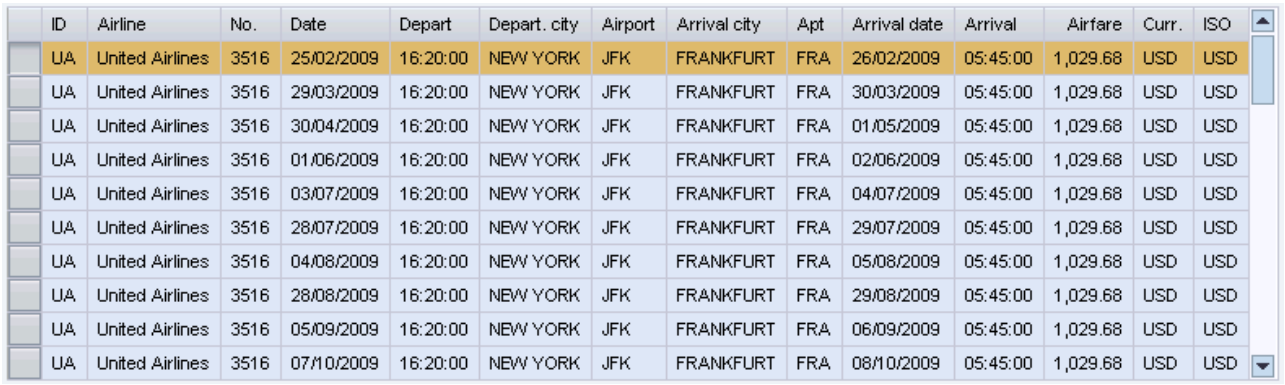


© 2010 SAP AG

In this case, the search criteria wants to find all flights in December 2009, so the Option column should be set to “Between lower and upper”, the Sign column set to “Inclusive of defined” and then the Low and High columns set to the start and end dates respectively.

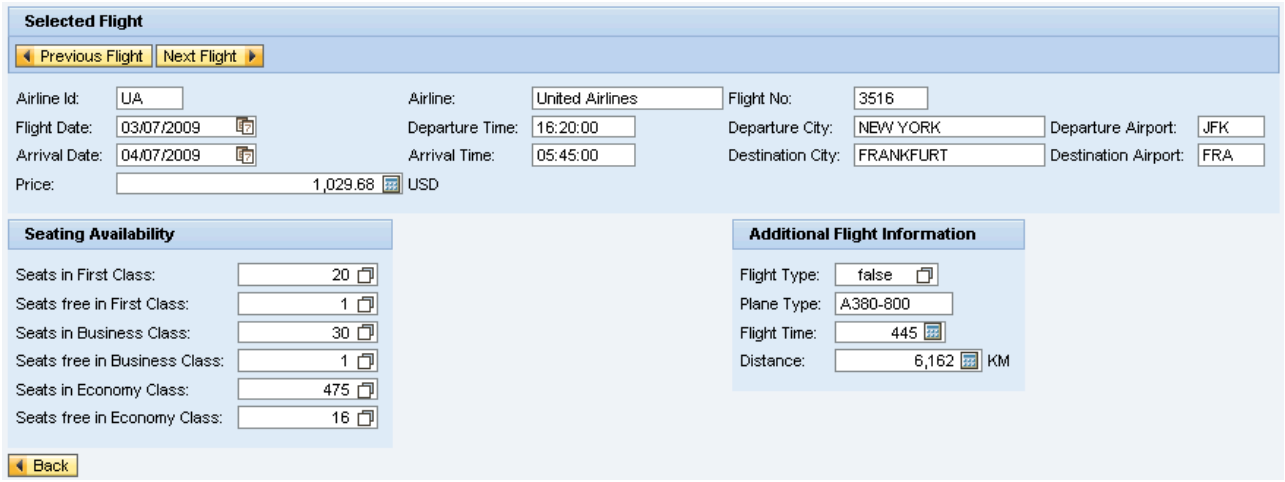
Whether or not you choose to add a date range does not affect the fact that when the “Start Search” the Web Dynpro application will make a call to BAPI_FLIGHT_GETLIST in the back end ABAP system.

Assuming you do not add a date range, you should see a table of results similar to the one show below:



ID	Airline	No.	Date	Depart	Depart. city	Airport	Arrival city	Apt	Arrival date	Arrival	Airfare	Curr.	ISO
UA	United Airlines	3516	25/02/2009	16:20:00	NEW YORK	JFK	FRANKFURT	FRA	26/02/2009	05:45:00	1,029.68	USD	USD
UA	United Airlines	3516	29/03/2009	16:20:00	NEW YORK	JFK	FRANKFURT	FRA	30/03/2009	05:45:00	1,029.68	USD	USD
UA	United Airlines	3516	30/04/2009	16:20:00	NEW YORK	JFK	FRANKFURT	FRA	01/05/2009	05:45:00	1,029.68	USD	USD
UA	United Airlines	3516	01/06/2009	16:20:00	NEW YORK	JFK	FRANKFURT	FRA	02/06/2009	05:45:00	1,029.68	USD	USD
UA	United Airlines	3516	03/07/2009	16:20:00	NEW YORK	JFK	FRANKFURT	FRA	04/07/2009	05:45:00	1,029.68	USD	USD
UA	United Airlines	3516	28/07/2009	16:20:00	NEW YORK	JFK	FRANKFURT	FRA	29/07/2009	05:45:00	1,029.68	USD	USD
UA	United Airlines	3516	04/08/2009	16:20:00	NEW YORK	JFK	FRANKFURT	FRA	05/08/2009	05:45:00	1,029.68	USD	USD
UA	United Airlines	3516	28/08/2009	16:20:00	NEW YORK	JFK	FRANKFURT	FRA	29/08/2009	05:45:00	1,029.68	USD	USD
UA	United Airlines	3516	05/09/2009	16:20:00	NEW YORK	JFK	FRANKFURT	FRA	06/09/2009	05:45:00	1,029.68	USD	USD
UA	United Airlines	3516	07/10/2009	16:20:00	NEW YORK	JFK	FRANKFURT	FRA	08/10/2009	05:45:00	1,029.68	USD	USD

The details of a particular flight can be seen by selecting a row of the search results table, as follows:



These details are obtained by the Web Dynpro application taking the airline code, the flight number and departure date of the selected flight and using these values as the input parameters for a call to BAPI_FLIGHT_GETDETAIL.

The details of the next and previous flights in the search results can be accessed by clicking on the “Previous Flight” and “Next Flight” buttons. This functionality does not wrap. That is, you cannot select a flight is off either the start or end of the list.

© 2010 SAP AG

Creating JCo Destinations for ARFC2

Caveat Confector![1]

ARFC1 destinations were created using the Web Dynpro Content Administrator tool. This tool is still available for compatibility reasons, but the destinations it creates are ***not*** used by ARFC2 models!

Instead, you should use the "Destinations" tool found on the Configuration Management screen of the NetWeaver Administrator.

The creation of JCo destinations does not need to be performed before the business application is written; however, it does need to be done in order to make the application executable.

Also, whatever JCo Destination names you choose during the model import process, must also be used in the steps shown below. After models have been imported, it is possible to change the destination names without needing to re-import the model.

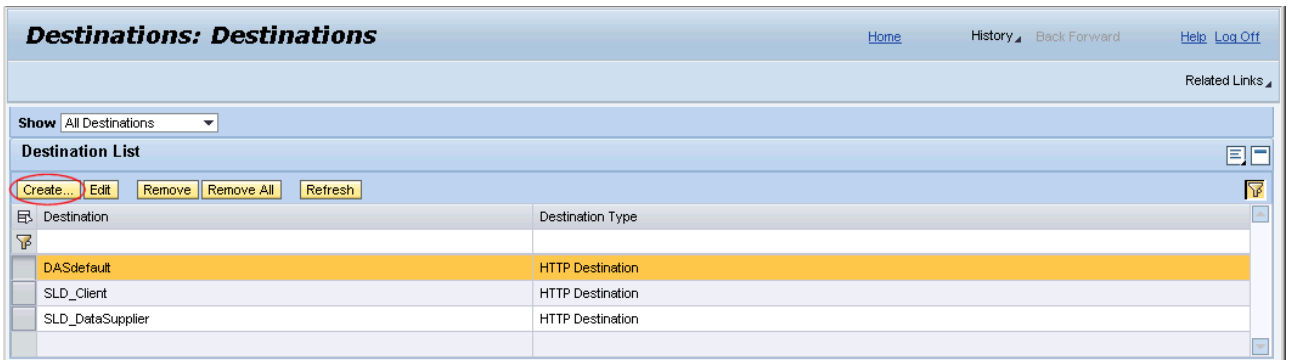
Irrespective of whether you are using version 1 or 2 of the Adaptive RFC model, you will always need to create at least 2 JCo Destinations. One connection is used by the Adaptive RFC layer itself to fill the metadata cache with dictionary information. This connection is shared by all models that connect to the same back end ABAP system. The other connection is used to invoke the remote callable ABAP function module.

Any number of applications can make use of the same model pointing to the same destination. At runtime, the JCo layer will supply each application instance with a JCo connection from a pool of connections.

In both versions of ARFC, a JCo destination can be thought of as a template. At runtime, a pool of JCo connections is created using the dimensions defined in the JCo destination.

For ARFC 2 models, JCo Destinations are created using the NetWeaver Administrator as follows:

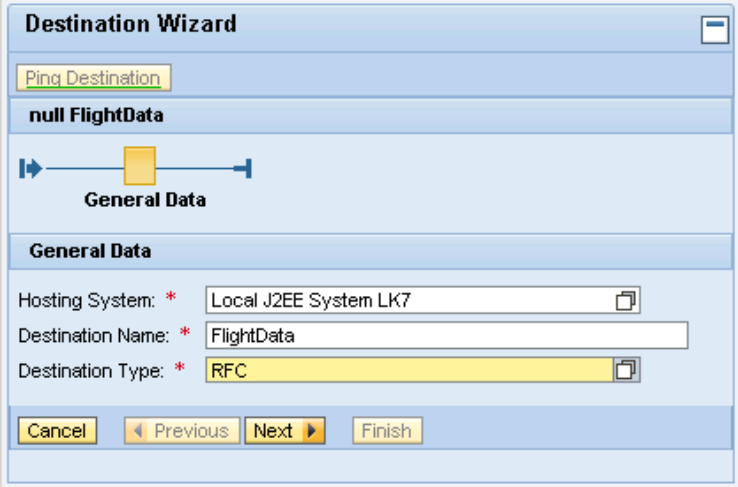
1. Point your browser to the start page of the SAP NetWeaver Application Server Java and log on to the NetWeaver Administrator.
2. From the top level menu bar, select "Configuration Management"
3. Select "Destinations" and then press "Create"



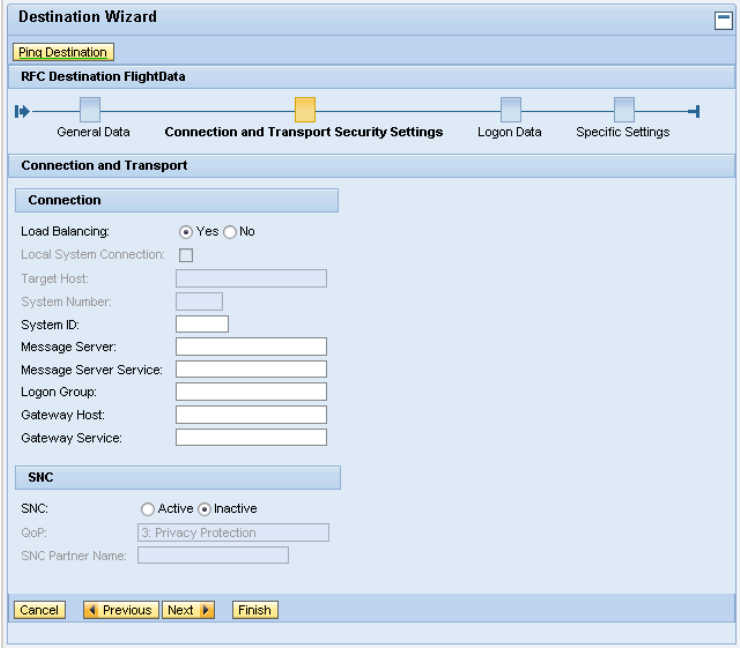
[1] Latin "developer beware"

© 2010 SAP AG

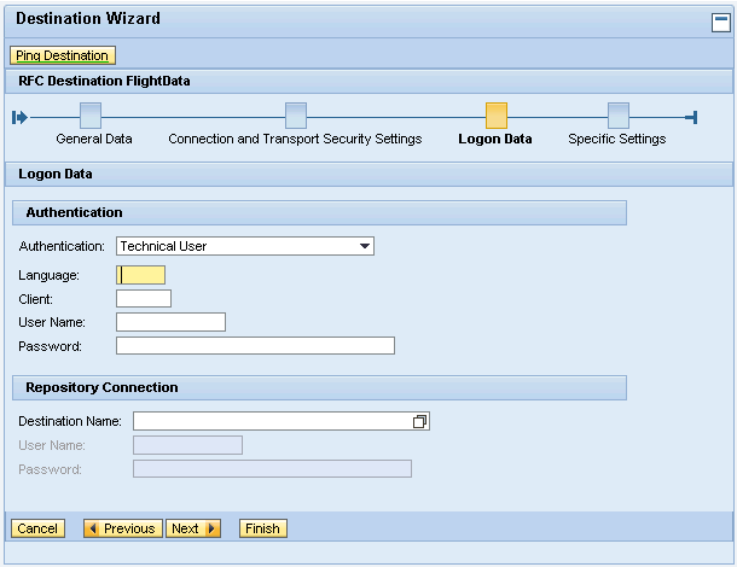
4. Enter the names of the system on which this destination is being created (typically, this will be the local system), the destination name and the destination type. Press “Next”.



5. Enter the details of the system to which this JCo destination will connect and press “Next”.



6. Now enter the logon details for how the JCo connection will be made and press “Next”.

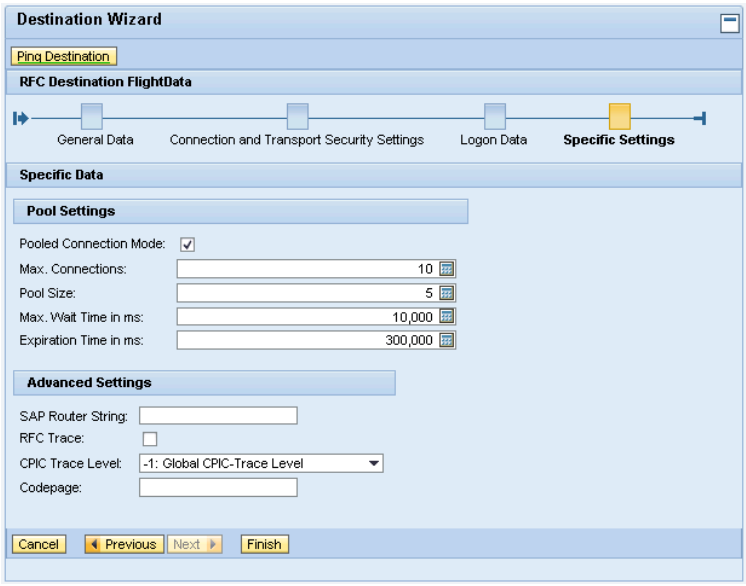


© 2010 SAP AG

7. Finally, the JCo Pool dimensions need to be entered.

   It is important that the meanings of these 4 pool dimension parameters are understood; otherwise it is possible to introduce potentially serious runtime inefficiencies.

   A brief description of these parameter values is given below.



Maximum Connections: Determines the maximum number of connections the pool may contain. This defines the pool's upper connection limit.

Pool Size: How many connections can remain open after they have been returned to the pool?

Max Wait Time: For how long will the JCo Pool Manager wait before deciding that the backend ABAP system cannot supply the required connection?

Expiration Time: For how long will an unused connection remain open after it has been returned to the pool? In a subsequent request for a JCo connection is not received within this expiry period, the connection is closed.

After pressing "Finish", the JCo destination will be created.

8. Repeat the above steps to create the JCo destination needed for the metadata connection.

© 2010 SAP AG

Migrating an ARFC1 Model to ARFC2 in the NWDS

Prerequisites

If you are not using a source control system, then you must manually check that all the source code files and sub-folders under **`${dc}`**`\_comp\src` are set to "Read-only" ***except for*** the folder(s) containing model specific files.

If you are using a source control system, then once you check in your change list, all the relevant files will be set to "read only".

It is very important to check that the model's migration to ARFC2 is taking place against the ***same*** back end ABAP system from which the original ARFC1 model was created.

Procedure

1) From your existing Web Dynpro project, select the ARFC1 model that needs to be migrated and right click on it.

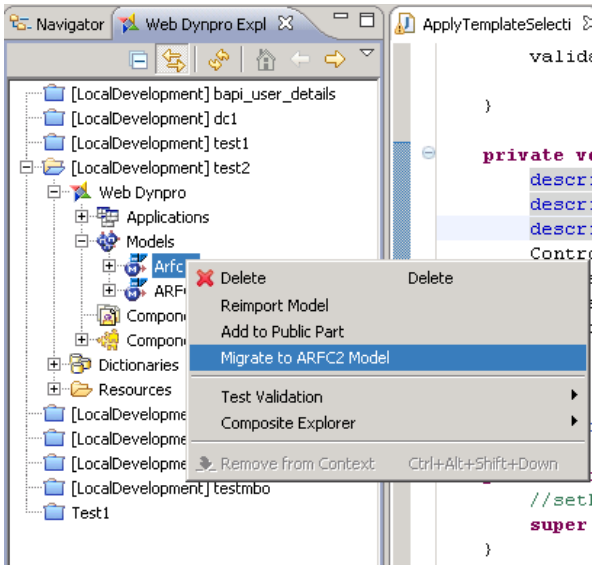
   Select "Migrate to ARFC2".

2) A pop-up will appear asking for confirmation. Click yes.

3) The ARFC 2 migration tool will then run in the background. After the migration is complete, the migration log page displays a list of the changes that were made.

4) If the migration is successful, an information dialog will appear.

5) After migration is complete, the entire development component must be reloaded into the NWDS.



6) Next, right-click on the DC project and choose "Repair" -> "Project Structure and Classpath". This is necessary because the migration process alters the structure of the generated model classes.

© 2010 SAP AG

Summary of Differences between ARFC1 and ARFC2

Instantiation of Model objects

In ARFC1, model objects could be instantiated by directly using the model class' null constructor. For instance:

```
Bapi_Flight_Getlist getFlightList = new Bapi_Flight_Getlist();
```

However, ARFC2 model objects are always created with reference to the model to which they belong. Therefore, the above coding must be modified slightly:

```
ARFC2FlightModel flightModel      = new ARFC2FlightModel();
Bapi_Flight_Getlist getFlightList = new Bapi_Flight_Getlist(flightModel);
```

Connection Management

In ARFC1, you had direct access to the `JCO.Client` object. This was the actual connection object to the backend ABAP system and using it, you could directly control the lifespan and parameters of the connection. For instance, it was perfectly possible to create your own `JCO.Client` object and assign it to a model using the `setJcoClient()` method.

IMPORTANT! In ARFC2, these coding possibilities no longer exist!

There are many reasons for why this is so, but in terms of monitoring, once the ARFC1 layer had created the `JCO.Client` object, it handed it over to the business application and consequently lost any ability to know how that connection was being used. This architecture removed any possibility of providing a tool for central monitoring of JCo connections. The best the ARFC1 layer could do was to keep track of the total number of connection objects it had created.

Now, the JCo connection is entirely managed by the ARFC2 layer and the business application no longer has access to the JCo connection.

The `JCO.Client` object no longer exists.

Explicit Connection Assignment

In ARFC1, it was possible to create your own `JCO.Client` object and assign it to a model object using the `setJcoClient()` method. This is no longer possible in ARFC2.

In the event that you need to migrate an older Web Dynpro application to use ARFC2 models, any sections of coding that perform direct connection assignment must be rewritten.

Connection Sharing

In ARFC1, it was possible to share a JCo connection between different models by means of the `setConnectionProvider()` method. This is no longer possible in ARFC2.

In ARFC2, if two model objects need to share the same JCo connection, then they must exist be imported into the same model.

Scope types in ARFC2

Using the ARFC1 scope type of `WDModelScopeType.APPLICATION_SCOPE` meant that the lifespan of a JCo connection was tied to the lifespan of the Web Dynpro application. However, since

© 2010 SAP AG

ARFC2 has been decoupled from any dependencies to Web Dynpro, the technical implementation of this association has had to be changed (although the concept has not changed).

ARFC2 now uses the concepts of a "scope provider" and a "resource provider". This means that Web Dynpro and JCo now interact in the following way:

- Web Dynpro acts as the "scope provider", and
- JCo (via ARFC2) acts as the "resource provider"

Acting as the scope provider, Web Dynpro has simply become a consumer of whatever connection resources are provided by the resource provider. It is the resource provider's job to determine the lifespan of any connections it manages.

In spite of the fact that ARFC2 no longer allows direct connection management, all JCo connections in ARFC2 behave in a conceptually identical way to `APPLICATION_SCOPE` connections in ARFC1 – although the technical implementation is very different.

Use of Model Class Methods Inherited from the aii Layer

Since ARFC2 no longer uses the `aii` layer, any coding in older Web Dynpro applications that called methods belonging to this layer will have be removed.

In practical terms, this situation in unlikely to occur since the methods inherited from the `aii` layer were concerned with such tasks as converting the BAPI interface data to and from XML – tasks with which a business application should not be concerned!

API Access to JCo Destination Definitions

In ARFC1, it was possible to access the details of a JCo destination using the static class `WDSystsemLandscape`. This is no longer possible in ARFC2 since the new JCo destinations are managed as J2EE Destination Services.

© 2010 SAP AG

Calling ARFC2 Model Objects

The easiest way to invoke a model object is to call its `execute()` method directly (this is true in both ARFC1 and ARFC2 scenarios). This simplifies the coding in that, once you have created a class-wide reference to a model object, it can be executed from any method within a non-visual controller.

In ARFC1, a model object would throw a `WDDynamicRFCExecuteException`. This exception is no longer used. Instead, the `ARFC2ModelExecuteException` exception is thrown.

Shown below are the code fragments taken from a non-visual controller needed to illustrate these points:

- Creation of a class-wide model object. (Definition of the input data structures has been omitted.)
- Direct execution of the model object from within a dedicated method. (This bypasses the need to have to traverse the context in order to locate the required model object).
- In the event that the model execution throws an exception, the user is shown not only the most meaningful error message available from the exception, but also any application error messages that may be available.

```
public void wdDoInit() {
  //@@begin wdDoInit()
  // Set the class wide reference to the component's message manager
  msgMgr = wdComponentAPI.getMessageManager();
  // Create a new model
  flightModel = new FlightModel();
  // Using the model, create an executable model object for the BAPI
  bapiGetFlightList = new Bapi_Flight_Getlist_Input(flightModel);
  <snip>
}

public void call_BAPI_FLIGHT_GETLIST( ) {
  //@@begin call_BAPI_FLIGHT_GETLIST()
  // Call BAPI_FLIGHT_GETLIST by using the class wide reference to the
  // model object.  This style of coding makes the coding easier to
  // understand, and bypasses the need to traverse the context to locate
  // the node to which the model object is bound, and then finally call
  // execute() method.  It is much simpler to invoke the execute() method
  // directly on the model object
  try {
    bapiGetFlightList.execute();

    // In ARFC2, it is no longer necessary to call the invalidate()
    // method of the context node that corresponds to the output side of
    // the BAPI's interface
    checkBapiReturn(bapiGetFlightList.getOutput().getReturn());
  }
  catch(ARFC2ModelExecuteException ex) {
    // If the call to the ABAP system goes pear-shaped, then the most
    // meaningful message to show the user is usually found by calling the
    // exception object's getNestedLocalizedMessage() method
    msgMgr.reportException(ex.getNestedLocalizedMessage());
    // Also write the exception to the system log
    // Please make it a habit to add this simple line of code because it
    // makes solving CSN messages *so* much easier!
    logger.catching(ex);
  }
  //@@end
}
```


© 2010 SAP AG

```
public void checkBapiReturn( java.util.List returnStruct )  {
  //@@begin checkBapiReturn()
  // Check that we've actually got a return value here
  if (!returnStruct.isEmpty()) {
    Bapiret2 returnMsg;

    // Process all the messages found in the return structure
    for (int i = 0; i < returnStruct.size(); i++) {
      returnMsg = (Bapiret2)returnStruct.get(i);

      // What king of message was returned?
      switch(returnMsg.getType().toCharArray()[0]) {
        // Error
        case 'E':
          msgMgr.reportException(returnMsg.getMessage());
          break;
        // Warning
        case 'W':
          msgMgr.reportWarning(returnMsg.getMessage());
          break;
        // Success or Information
        case 'S':
        case 'I':
          // msgMgr.reportSuccess(returnMsg.getMessage());
          break;
        // Something else...
        default:
          msgMgr.
            reportSuccess("Received message type " + returnMsg.getType() +
                          " with text " + returnMsg.getMessage());
      }
    }
  }
  else
    msgMgr.reportWarning("Empty return structure after BAPI call");
  //@@end
}

//@@begin others
IWDMessageManager msgMgr;
FlightModel flightModel;
Bapi_Flight_Getlist_Input   bapiGetFlightList;
//@@end
```


© 2010 SAP AG

Copyright

© Copyright 2010 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, System i, System i5, System p, System p5, System x, System z, System z10, System z9, z10, z9, iSeries, pSeries, xSeries, zSeries, eServer, z/VM, z/OS, i5/OS, S/390, OS/390, OS/400, AS/400, S/390 Parallel Enterprise Server, PowerVM, Power Architecture, POWER6+, POWER6, POWER5+, POWER5, POWER, OpenPower, PowerPC, BatchPipes, BladeCenter, System Storage, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, Parallel Sysplex, MVS/ESA, AIX, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects S.A. in the United States and in other countries. Business Objects is an SAP company.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.