SAP Composite Application Framework

Using a Composite Application via a Web Dynpro UI
SAP AG
Neurottstraße 16
69190 Walldorf
Germany
T +49/18 05/34 34 24
F +49/18 05/34 34 20
www.sap.com

© Copyright 2005 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, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, and Informix are trademarks or registered trademarks of IBM Corporation 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.

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, 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 in several other countries all over the world. 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.

Disclaimer
Some components of this product are based on Java™. Any code change in these components may cause unpredictable and severe malfunctions and is therefore expressively prohibited, as is any decompilation of these components.

Any Java™ Source Code delivered with this product is only to be used by SAP’s Support Services and may not be modified or altered in any way.
## Typographic Conventions

<table>
<thead>
<tr>
<th>Type Style</th>
<th>Represents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example Text</td>
<td>Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.</td>
</tr>
<tr>
<td>Example text</td>
<td>Emphasized words or phrases in body text, graphic titles, and table titles.</td>
</tr>
<tr>
<td>EXAMPLE TEXT</td>
<td>Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.</td>
</tr>
<tr>
<td>Example text</td>
<td>Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.</td>
</tr>
<tr>
<td>Example text</td>
<td>Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.</td>
</tr>
<tr>
<td>&lt;Example text&gt;</td>
<td>Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.</td>
</tr>
</tbody>
</table>

## Icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Alert" /></td>
<td>Caution</td>
</tr>
<tr>
<td><img src="image" alt="Example" /></td>
<td>Example</td>
</tr>
<tr>
<td><img src="image" alt="Note" /></td>
<td>Note</td>
</tr>
<tr>
<td><img src="image" alt="Recommendation" /></td>
<td>Recommendation</td>
</tr>
<tr>
<td><img src="image" alt="Syntax" /></td>
<td>Syntax</td>
</tr>
</tbody>
</table>
Contents

Introduction .................................................................................................................................... 2
  Requirements and Dependencies ................................................................................................. 2
  Applicable Releases .................................................................................................................... 2
  Disclaimer .................................................................................................................................... 2

Create the Web Dynpro UI ........................................................................................................... 3
  Web Dynpro Project .................................................................................................................... 3
  Web Dynpro Layout ..................................................................................................................... 6

Create and publish proxy classes in CAF DC ................................................................................ 9

Customize NWDI settings ........................................................................................................... 10

Import proxy classes into Web Dynpro DC .................................................................................. 11

Create Model Nodes and Binding ............................................................................................... 13

Do the coding ............................................................................................................................... 19
  A simple query to get all locations ............................................................................................. 19
  Linking an Object Browser pattern to a Web Dynpro Development Component.................... 22
  Creation of a new entity service instance – create new travels ............................................... 31
  Another query – find existing travels ....................................................................................... 36
  Updating an entity service – add a new passenger ....................................................................... 38
Introduction

Requirements and Dependencies
Before you start with this tutorial you should have installed the following Software:

- SAP Web Application Server Java 7.0
- SAP NetWeaver Developer Studio 7.0

This tutorial is based on the following How-to Guide "Create an Application Service".

Additionally you should have configured CAF as described in the How-to Guide "Installation and Configuration Guide".

Applicable Releases
This tutorial is compatible with the following releases:

- SAP NetWeaver ’04s
- SAP Composite Application Framework (CAF) 7.0

Disclaimer
Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or grossly negligent.
Create the Web Dynpro UI

Web Dynpro Project

- First you have to create a new development component type “Web Dynpro” called “carpool/travel”.

![Web Dynpro Project Creation Screenshot](image)
• In this DC create a new Web Dynpro component called “Travel”. Package: “com.sap.carpool.travel”. Window and view should be generated like the proposed defaults.

• Also create a new application called “TravelSearch” in the same package. Choose “No authentication”. Press “Next”.
  • Choose “use existing component”
  • Press “Next”.
  • Choose the provided component “Travel”.
  • Press “Finish”. The project structure will appear as shown in this picture:
• Next choose the component controller.
• Create a new value node “FindTravel” without structure binding. In the properties of this value node set its cardinality to “1..1”.
• Add to this value node three value attributes: “StartLocation” type String, “TargetLocation” type String and “TravelDate” type Date.
• Switch to the TravelView
• Add the Travel controller to the list of required controllers
• Add the same node with the same fields as described above to the context of the TravelView
• Create a context mapping for these elements to the elements of the same name on the controller.
• In the TravelView please add the following Actions: none of them has any parameters.
  • AddPassenger
  • BrowseEmployee
  • CreateTravel
  • FindTravel
  • Refresh
Web Dynpro Layout

- Next you have to create a Web Dynpro Layout as shown below:

This Layout has to contain the following elements:

- Group called “Group1”, layout “Grid Layout”, colCount 2, header: “Find Carpool”, design “secondarycolor”, including all the following elements
- Label (called “StartLocationLabel”), text “Start Location”
- DropDownListByKey called “StartLocation”, event “onSelect” triggers Action “Refresh”
- Label (called “TravelDateLabel”), text “Travel Date”
- InputField called “StartDate”
- Button called “SearchButton”, text “Search”, colspan 2, event “onAction” triggers Action “FindTravel”
- Table (called TravelTable), colspan 2, paddingTop “large”, header: “Travels Found”, including the following columns. All other columns except the id are optional.
  - Column (called “IdColumn”) with “TextView “ cell editor and header: “ID”
  - Column (called “TravelDateColumn”) with “TextView “ cell editor and header: “Travel Date”
- Column (called “StartLocationColumn”) with “TextView” cell editor and header: “Start Location”
- Column (called “TargetLocationColumn”) with “TextView” cell editor and header: “Target Location”
- Column (called “KeeperFirstNameColumn”) with “TextView” cell editor and header: “Keeper First Name”
- Column (called “KeeperLastNameColumn”) with “TextView” cell editor and header: “Keeper Last Name”
- Column (called “CarIdColumn”) with “TextView” cell editor and header: “Car Tag”
- Another Table (called “PassengerTable”), paddingTop “large”, header: “Passengers On Selected Travel”, including the following columns:
  - Column (called “PassengerFirstNameColumn”) with “TextView” cell editor and header “First Name”
  - Column (called “PassengerLastNameColumn”) with “TextView” cell editor and header “Last Name”
- Group called “Group2”, layout “Grid Layout”, colCount 2, vAlign “middle”, header: “New Passenger”, including all the following elements
  - Button called “ChooseEmployeeButton”, text: “Choose Employee”, colspan 2, event “onAction” triggers Action “BrowseEmployee”
  - Label (called “NPFirstNameLabel”), text: “First Name”
  - InputField called “FirstName”, readOnly: “true”
  - Label (called “NPLastNameLabel”), text: “Last Name”
  - InputField called “LastName”, readOnly: “true”
  - Button called “AddPassengerButton”, text “Add As Passenger”, colspan 2, event “onAction” triggers Action “AddPassenger”
  - Button called “CreateTravelButton”, text “Create New Travel”, colspan 2, event “onAction” triggers Action “CreateTravel”

As a result your outline view could look like this:
Create and publish proxy classes in CAF DC

- First the proxy classes have to be generated in the composite application DC: To do this, right click in the Service Explorer on the DC name ("carpool") and choose "Create Web Dynpro Model"

- You can now check in the Web Dynpro Explorer: The proxy classes are generated and can be browsed in the Web Dynpro DC of the composite application at: /src/packages/com.sap/carpool/services/..
Customize NWDI settings

Before the new “Travel” Web Dynpro DC can access the proxy classes from the CAF DC, you have to customize some necessary NWDI settings.

- In the Web Dynpro Explorer open the Web Dynpro project of the CAF DC (not the new Web Dynpro DC “Travel”)
- Expand the branch “DC Metadata”
- Right click on the node “Public Parts” and choose “New Public Part”!

- Call the public part “carpool”
- Ensure that “Provides an API for developing/compiling other DCs” is checked
- Click “Next”
• Choose “Java Package Tree” as entity type
• Browse to and select: “com.sap.carpool.services.carpool”
• Select “Add Entity to public part as class”
• Push “Finish”
• Save the metadata. Then switch to the Service Explorer, right click on the top node of the project and choose “generate project code”. Afterwards choose in the same context menu “Development Component -> Build..” and press OK.

Import proxy classes into Web Dynpro DC

Now you have to import the new generated classes into Web Dynpro DC “Travel”.

• In the Web Dynpro Explorer of the Web Dynpro perspective expand the new “Travel” Web Dynpro project!
• Expand the nodes DC Meta Data – DC Definition and right click on “Used DCs”
• Choose “Add used DC”
• Find the Web Dynpro project of the CAF DC
• Locate and select the public part with the model in the Web Dynpro project at DC metadata: Public Parts – package_tree – Entities
• Choose Dependency Type “Build Time”
• Press “Finish”
• Then add the following public parts:
  o public part types_compilation of DC caf/core/dict (CAF compartment)
  o public part com.sap.caf.ui.utils of DC caf/UI/ptn/common (CAF compartment)
  o public part default of DC tc/col/runtime (compartment SAP_JTECHS)

Right click on the node and choose “Add”

Choose the “carpool” model and press OK

Create Model Nodes and Binding

Next you have to extend the context of the component controller with two additional nodes:

In the Web Dynpro Explorer open the component controller of the “Travel” Web Dynpro application

In the context view right click on the context root node and choose “New” – “Model Node”

Call the model node “Employee” and press “Finish”
• Right click on the new “Employee” node and choose “Edit Model Binding”

• Choose the model class “AEmployee” and press “Next”

• Select the attributes “firstName” and “lastName”

• Press “Finish”

• The new model node “Employee” will have received two new attributes, which are derived from the model class.
• In the same way Create a new model node “Travel”
• Bind the model class “ATravel” to the model node.
• Create new attributes for the following properties of the model class:
  o car-id
  o car-manufacturer
  o car-model
  o car-numberSeats
  o keeper-firstName
  o keeper-lastName
  o passenger-firstName
  o passenger-lastName
  o startLocation-id
  o startLocation-name
  o targetLocation-id
  o targetLocation-name
  o id
  o startDate

As a result you should get the context shown on the right
• Please now select in the context the new subnode Travel-car and change in the Properties View the property “singleton” to “false”. This is needed to be able to bind these subnodes together later with the super node Travel to a table.

• Repeat for the subnodes Travel-keeper, Travel-startlocation and Travel-targetLocation.

• Save the meta data.

• Switch to the Travel view

• Create two new model nodes in the context tab as described above but without model binding:
  o Employee
  o Travel

• Right-click on the Employee node and choose “Edit Context Mapping”
• Choose the Employee node from the controller context and choose "Next"

• Select the complete “Employee” node with all attributes and press “Finish”

• Repeat for the model node “Travel” and map it to the complete “Travel” node in the controller context as well

• Press “Finish”
• Save the metadata.

As a result you should now have the same context in the Travel view as in the Travel controller; the controller contains the bindings and the view refers to the controller. Modeling it in this way ensures that we adhere closely to the Model-View-Controller architecture.

Now bind the layout to the complete context:

• Please open the Travel view and switch to the Layout tab.

• Please bind the following UI elements to the specified context nodes:

  • Set the “selectedKey” property of the StartLocation “DropDownByKey” element to FindTravel-StartLocation
  • Set the “selectedKey” property of the TargetLocation “DropDownByKey” element to FindTravel-TargetLocation
  • Set the “value” property of the StartDate “InputField” element to FindTravel-TravelDate
  • Set the “dataSource” property of the Travel Table to the model node “Travel”
• Set the property “text” of the Table Cell Editors for each column of the Travel table to its corresponding attribute in the model node “Travel”.

• Set the “dataSource” of the Passenger table to the model sub node Travel-passenger.

• Set the property “text” of the Table Cell Editors for each column of the Passenger table to its corresponding attribute in the model node Travel-passenger.

• Set the property “value” of the input field “firstName” to the model attribute Employee-firstName.

• Set the property “value” of the input field “lastName” to the model attribute Employee-lastName.

As a result your layout should look like this:

Find Carpool

<table>
<thead>
<tr>
<th>Start Location</th>
<th>Target Location</th>
<th>Travel Date</th>
<th>Search</th>
</tr>
</thead>
</table>

Travels Found

<table>
<thead>
<tr>
<th>ID</th>
<th>Travel Date</th>
<th>Start Location</th>
<th>Target Location</th>
<th>Keeper First Name</th>
<th>Keeper Last Name</th>
<th>Can Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Passengers On Selected Travel

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

New Passenger

<table>
<thead>
<tr>
<th>Choose Employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Name: Employee firstName</td>
</tr>
<tr>
<td>Last Name: Employee lastName</td>
</tr>
<tr>
<td>Add As Passenger</td>
</tr>
</tbody>
</table>

Create New Travel

Do the coding

A simple query to get all locations
Next you have to implement the methods. The first step is to initialize the two drop-down boxes for the
locations with all available values. To do this you have to add the following coding to the method “wdDoInit” on
the component controller:

```java
// Initialize location fields:
IAspect aspectList;
IASpectRow aspectRow;
int count;

IWDAttributeInfo attributeInfoStart, attributeInfoTarget;
ISimpleTypeModifiable startLocation, targetLocation;
IModifiableSimpleValueSet valueSetStart, valueSetTarget;

// access the context and get the relevant value nodes:
attributeInfoStart =
    wdContext.nodeFindTravel().getNodeInfo().getAttribute(  
        "StartLocation");
startLocation = attributeInfoStart.getModifiableSimpleType();
valueSetStart =
    startLocation.getSVServices().getModifiableSimpleValueSet();
attributeInfoTarget =
    wdContext.nodeFindTravel().getNodeInfo().getAttribute(  
        "TargetLocation");
targetLocation = attributeInfoTarget.getModifiableSimpleType();
valueSetTarget =
    targetLocation.getSVServices().getModifiableSimpleValueSet();

    // access the proxy and get all available locations
aspectList = TravelLocationServiceProxy.getAll();

    // add wildcards for a general search:
valueSetStart.put("*", "*");
valueSetTarget.put("*", "*");

    // loop over the result list
```
for (count = 0; count < aspectList.size(); count++) {
    aspectRow = aspectList.getAspectRow(count);

    valueSetStart.put(
        aspectRow.getAttributeAsString("key"),
        aspectRow.getAttributeAsString("name"));
    valueSetTarget.put(
        aspectRow.getAttributeAsString("key"),
        aspectRow.getAttributeAsString("name"));
}

Please also add the following line to the import section of the controller:

```
import com.sap.caf.carpool.services.carpool.travellocationservice.TravelLocationServiceProxy;
```

Afterwards right click on the coding and choose “source/organize imports”. You will probably be asked what kind of IAspect you want to import. Choose: “com.sap.tc.col.client.generic.api.IAspect”. Save the metadata.

Now please build, deploy and run the Travel Web Dynpro project.

If you have already created some TravelLocations in your CAF application you should get the Web Dynpro page as displayed here. The travel locations are already retrieved from the CAF and filled in the dropdown boxes. But the other functionality is still missing and we have to implement it next.

If the dropdown boxes for start and target location are still empty please first create some travel locations by using the test framework of the CAF (see also the How-to for testing entity and application services).
Linking an Object Browser pattern to a Web Dynpro Development Component

Next you have to implement a possibility for searching for employees that could be assigned to a chosen travel.

Since this is just a standard task where you can use a standard CAF pattern. This pattern is called “Object Browser” pattern and has to be configured first. This has to be done using the “ConfigBrowser”.

So please start the CAF ConfigBrowser using the link:

http://<hostname>:<portnumber>/webdynpro/dispatcher/sap.com/caf~UI~configbrowser/ConfigBrowser

- Choose the pattern “Object Browser”
- Click on “New Configuration”
- Enter the configuration name “findEmployee”
- Set the object browser title to “Find Employee”
- Enter the search bar configuration name “employeeSearchBar”
- Enter the object browser list configuration name “employeeBrowserList”
If the employeeSearchBar does not already exist:

- Click on “Configure” next to “employeeSearchBar”
- Choose the service: “sap.com/carpool/Employee Service”
- Choose the Find By Operation “findByLastName”
- Click on “Retrieve metadata”
- Select the input parameter “lastName”
- Add the starting phrase for SearchBar: “Search for Employee”
- Click on “Save + Return”
- Next click on the “Configure” button next to “employeeList”
- Choose the service module “sap.com/carpool/EmployeeService”
- Query name: “findByLastName”
- Click “Display Aspect Fields”
- Choose as displayed fields “id”, “firstName” and “lastName”
- In the section “Web Dynpro Application Launched for Selected Object” at the bottom of the screen, choose:
  - Application Name: “ObjectEditor”
  - Development Component: “sap.com/caf~UI~ptn~objecteditor”
  - Configuration Name: “createEmployee”.
  This configuration should be available if you have completed the How-to “Create local entity service with maintenance UI” upfront.
- Leave the field “Label for Button for Object Assignment” empty.
Click “Save + Return” for the Object Browser List Configuration.

Next you can click on “Preview” to check if the pattern works. Just search with an asterisk and check if there are any results. If there are none found create some new employees using the Employee Selector pattern that you have configured in the How-to “Create local entity service with maintenance UI” available at: 

Close the preview and click on “save + return” for the Object Browser pattern configuration.
Now the Object Browser Pattern is configured. Next you have to make it available to the Web Dynpro DC.

- Open the Web Dynpro Explorer in the IDE
- First you have to add some standard DCs that are needed. Please right click again on the node “Used DCs” under DC MetaData – DC Definition in the “Travel” Web Dynpro project and choose “Add Used DC”.
- Please add the following DCs as described above:
  - Public parts of DC "caf/runtime/uicoupling" (CAF) for build time.
  - Public parts
    - Common Interfaces
      - types_compilation of DC "caf/UI/ptn/common" (CAF) for build time

The object browser pattern that is called is nothing other than another Web Dynpro component. Therefore you have to declare it’s usage for the DC:

- Right-Click on “Used Web Dynpro Components”
- Choose “Add Used Component”
• Enter “ObjectBrowser” as name
• Choose “ObjectBrowserCl - com.sap.caf.ui.ptn.objectbrowser.def.” as “Used Web Dynpro Component”
• Click on “Finish”

The result is a new Used Web Dynpro Component called “Object Browser”. Below this node there is another node representing the new interface controller for the component. Please open this controller and switch to the context tab.

Here you find the context parameters of the interface.

The first one is called “configName”. This is the name of the configuration for the object browser pattern.

The next one is called “parameter1”. Usually the key of another entity service will be provided here.
Next a new object browser instance that can be called has to be initialized. This has only to be done once and will therefore happen during the initialization of the Web Dynpro Component.

To be able to refer to another controller this interface controller has to be added to the Web Dynpro component controller first. So please switch on the Travel controller to the “Properties” tab and add the object browser controller to the list of required controllers.

Afterwards please add the following coding to the method “wdDoInit” of the Travel Web Dynpro component controller:

```java
IWDComponentUsage usage;

// Initialize Object Browser
usage = wdThis.wdGetObjectBrowserComponentUsage();
// Create a new instance
usage.createComponent("com.sap.caf.ui.ptn.objectbrowser.ObjectBrowser",
"sap.com/caf~UI~ptn~objectbrowser");
// Set the parameters of the interface component
wdThis
  .wdGetObjectBrowserInterface()
  .wdGetAPI()
  .getContext()
  .getRootNode()
  .getCurrentElement()
  .setAttributeValue("configName", "findEmployee");
wdThis
  .wdGetObjectBrowserInterface()
  .wdGetAPI()
  .getContext()
  .getRootNode()
  .getCurrentElement()
  .setAttributeValue("parameter1", "");

// Initialize the component
wdThis.wdGetObjectBrowserInterface().initialize();
```
Now you have to implement the call for the Object Browser.

- Add a new method to the Travel component controller. It has no parameters, return type “void” and will be called “launchObjectBrowser”
- Add the following line of code to this method:
  ```java
  wdThis.wdGetObjectBrowserInterface().showWindow();
  ```
- Add the following line of code to the action handler for the “Choose Employee” button called “onActionBrowseEmployee”:
  ```java
  wdThis.wdGetTravelController().launchObjectBrowser();
  ```

This is enough to show the Object Browser Pattern. You can test it already if you like: Save the metadata, build the Travel Web Dynpro Component and deploy and run it. If you now click on the “Choose Employee” button the Object Brower should appear. Nevertheless if you select an employee and click on “Add to” his name still will not be shown in the Web Dynpro application, because there is currently no link to transfer the return data created. This is what we do next.

- Open the Travel component controller and add a new method
- The method shall be of type “Event handler”
- Choose “onBrowserReturn” as name
- Event source is the Object Browser Component controller
- Subscribed event is “SelectionAdded”
- Check the field “Create event handler parameters according to subscribed event”
- Click on Finish
In the Implementation tab of this method please add the following coding

```java
IAspectRow aspectRow;
IWDNodeElement node;
String key;

if (selection.size() > 0)
{
    // We only care about the first selected element
    // as the user may not choose multiple parameters anyway.
    aspectRow = (IAspectRow)selection.get(0);
    // The Object Browser pattern only provides us with the
    // key. The remaining attributes have to be read afterwards.
    key = aspectRow.getAttributeAsString("key");
    wdContext.nodeEmployee().bind(EmployeeServiceProxy.read(key));
}
```

Please choose “Source” - “organize imports”!

That is all for the link to the object browser pattern; please save, build your DC, deploy and run it.

Now you should be able to browse for an employee. Choose one and transfer it to the Travel Web Dynpro application by clicking on “Add to”. The chosen employee should be shown there.
## Find Carpool

- **Start Location**: Walldorf
- **Target Location**: Berlin
- **Travel Date**: 09/30/2005

### Travels Found

<table>
<thead>
<tr>
<th>ID</th>
<th>Travel Date</th>
<th>Start Location</th>
<th>Target Location</th>
<th>Keeper First Name</th>
<th>Keeper Last Name</th>
<th>Car Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Passengers On Selected Travel

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### New Passenger

- **Choose Employee**
- **First Name**: Robert
- **Last Name**: De Niro
- **Add As Passenger**
- **Create New Travel**

---

Row 0 of 0
Creation of a new entity service instance – create new travels

As there are currently no travels existing so far, the next step will be to create a new one. This will happen again in the controller:

- Open the Travel Web Dynpro controller and add a new method called “createNewTravel” with return type “void”
- Add 3 parameters to this method:
  - startLocation of type “string”
  - targetLocation of type “string”
  - travelDate of type “date”

To create a new travel instance you also have to assign a car from our carpool. Please create a new method for this task:

- Add another method to the Travel Web Dynpro component controller. This method shall be called “getCarForDate”
- Please choose as return type the proxy class “com.sap.carpool.services.carpool.carservice.ACar”. Do this by using the “Java native type” browsing.
- Add one parameter called “travelDate” of type “date”
Next you have to implement the coding for the method “getCarForDate”. Usually here a complex external availability check for our car fleet would take place. But in this How-to we are keeping things simple: we assume that every car that is not traveling on this day is available. If there is none available, nil will be returned.

Please add the following coding in the Implementation tab to the method “getCarForDate”:

```java
IASpectRow car;
ACar result;
IASpect travel;
IASpect cars;
String carKey;
java.util.Iterator carIterator;

result = null;
cars = CarServiceProxy.getAll(); // get all cars

for (carIterator = cars.iterator(); carIterator.hasNext();){
    car = (IASpectRow)carIterator.next();
    carKey = car.getAttributeAsString("key");
    travel = TravelServiceProxy.findByDateAndCarKey( carKey,
        travelDate);
    if (travel.size() == 0){ //car not booked
        result = CarServiceProxy.read(carKey);
        break;
    } else { // car is booked
        car = null;
    }
}
return result;
```

- Afterwards choose “Source” – “organize imports”
Next you have to implement the logic for the creation of a new travel.

- Please add the following coding in the Implementation tab of the Travel Web Dynpro controller to the method "createNewTravel"

```java
ATravelLocation start, target;
ATravel travel;
ACar car;
AEmployee employee;
IServiceFacade serviceFacade;
IPublicTravel.ITravelElement travelElement;

// Are all mandatory parameters provided?
if ((startLocation != null) && (!startLocation.equals("")) && (targetLocation != null)
    && (!targetLocation.equals("")) && (travelDate != null)
    && (wdContext.currentEmployeeElement() != null)) {

    start = TravelLocationServiceProxy.read(startLocation);
    target = TravelLocationServiceProxy.read(targetLocation);

    // get available car:
    car = this.getCarForDate(travelDate);

    // get the chosen employee
    employee = wdContext.currentEmployeeElement().modelObject();

    if (car != null) {
        // first a new instance is created
        travel = TravelServiceProxy.create();
        // afterwards we can update the properties
        travel.setId(System.currentTimeMillis());
        // workaround for unique id
        travel.setCar(car);
        travel.setStartLocation(start);
        travel.setTargetLocation(target);
        travel.setStartDate(travelDate);
        travel.setKeeper(employee);

        // Last we have to commit our changes to the
        // database. This is done in two steps:
        travel.getAspect().sendChanges();
        serviceFacade =
            CAFServiceFactory.getServiceFacade(
                com.sap.carpool.services.carpoolDefinition.class);
        serviceFacade.save();
    }
}
```
// Add the new travel to the travel table
travelElement = wdContext.nodeTravel().
createTravelElement(travel);
wdContext.nodeTravel().addElement(travelElement);
} else { // no more cars available on selected date
  wdComponentAPI.getMessageManager().reportWarning(
    "No more cars available on selected date.");
  // in "real life" you should externalize this message
}
} else {
  wdComponentAPI.getMessageManager().reportWarning(
    "Select start location, target location, travel date and employee first!");
  // in "real life" you should externalize this message
}

- Afterwards choose “Source” – “organize imports”

Now please switch to the Travel View and implement a call to this create method when the “Create New Travel” button is triggered.

- Switch to the Implementation tab of the Travel View and add the following lines of code

```java
String targetLocationUI, startLocationUI;
java.sql.Date startDate;

// retrieve parameters from context
startLocationUI = wdContext.currentFindTravelElement().getStartLocation();
targetLocationUI = wdContext.currentFindTravelElement().getTargetLocation();
startDate = wdContext.currentFindTravelElement().getTravelDate();

dwThis.wdGetTravelController().createNewTravel(
  startLocationUI,
  targetLocationUI,
  startDate);
```
Now you should be able to create new travels. Try it: save, build, deploy and run the application. Fill in a start location, a target location, a travel date and choose an employee. Then click on “Create New Travel”. As a result a new entry should appear in the table.
Another query – find existing travels

Now we have created our first travel instances. Next we can implement now our central query: We would like to find all possible car pools that are available on a chosen date and travel from the chosen start location to the chosen target location.

- To achieve that, please first create a new method in our controller: Open the component controller, go to the Methods tab and add a new method called “findTravels” with return type “void”. Add the following 3 parameters to this method:
  - startDate of type date
  - startLocation of type string
  - targetLocation of type string

- Please go to the implementation tab and insert the coding for the new method between the “@@begin” and “@@end” tags of the “findTravel” method.

```java
// Let's retrieve all matching travel entities by calling
// the findBy proxy method and binding the result to the
// corresponding context node
wdContext.nodeTravel().bind(
    TravelServiceProxy.findByDateStartAndTargetLocation(startLocation, startDate, targetLocation));
```
Switch to the TavelView and choose the Implementation Tab. The new method on the controller has to be called by the event handler for the “search” button. So please add the following coding to the method “onActionFindTravel”.

```java
String targetLocationUI, startLocationUI;
java.sql.Date startDate;

startLocationUI = wdContext.currentFindTravelElement().getStartLocation();
targetLocationUI = wdContext.currentFindTravelElement().getTargetLocation();
startDate = wdContext.currentFindTravelElement().getTravelDate();

wdThis.wdGetTravelController().findTravels(startDate, startLocationUI, targetLocationUI);
```

That is all that is required for the search functionality. Please save, build, deploy and run your application and check that you can find the travels that you have just created.
Updating an entity service – add a new passenger

Now that we have found a carpool we would also like to be able to add a new passenger. This is the last step of this tutorial.

- Please add a new method to the Travel component controller called “addPassenger” with return type “void” and without any parameters.
- In the Implementation tab of this controller please add the following coding to this new method

```java
ATravel travel;
AEmployee employee;
IASpectRow passenger;
ACar car;
java.util.List passengers;
IServiceFacade serviceFacade;

// get proxy objects from context
travel = wdContext.currentTravelElement().modelObject();
employee = wdContext.currentEmployeeElement().modelObject();

// check if the mandatory attributes are filled
if ((travel == null) || (employee == null)) {
    return;
}

car = travel.getCar();
passengers = travel.getPassenger();

// check if passenger is already assigned
for (int count = 0; count < passengers.size(); count++) {
    passenger = (IASpectRow) passengers.get(count);
    if (new Long(passenger.getAttributeAsString("id")) longValue()
        == employee.getId()) {
```
return;
}
}
// check if new passenger is already keeper
if (employee.getId() == travel.getKeeper().getId()) {
  return;
}
// check if there are still seats available
if (car.getNumberSeats() == passengers.size() + 1) { // car is full
  wdComponentAPI.getMessageManager().reportWarning(
    "No more seat available on selected travel.");
  // in "real life" you should externalize this message
} else {
  travel.addPassenger(employee);
  // commit changes to the entity service
  travel.getAspect().sendChanges();
  serviceFacade =
    CAFServiceFactory.getServiceFacade(carpoolDefinition.class);
  serviceFacade.save();
}

Now we only need to link the button “Add passenger” to the new method.

- Open the Travel View and switch to the implementation tab.
- Add the following line of code to the method “onActionAddPassenger”

  wdThis.wdGetTravelController().addPassenger();
Congratulations! Save, build, deploy and run your application. Now you should be able to

- Select Travel locations by retrieving available entity services
- Search for travels by triggering a query on an entity service
- Choose an employee by calling a separate CAF UI pattern configuration
- Create a new travel object by creating a new instance of an entity service
- Add passengers to a carpool by updating an existing entity service