**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,</td>
</tr>
<tr>
<td></td>
<td>screen titles, pushbuttons labels, menu names, menu paths, and menu</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>names, transaction codes, table names, and key concepts of a programming</td>
</tr>
<tr>
<td></td>
<td>language when they are surrounded by body text, for example, SELECT and</td>
</tr>
<tr>
<td></td>
<td>INCLUDE.</td>
</tr>
<tr>
<td>Example text</td>
<td>Output on the screen. This includes file and directory names and their</td>
</tr>
<tr>
<td></td>
<td>paths, messages, names of variables and parameters, source text, and names</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>and characters with appropriate entries to make entries in the system.</td>
</tr>
<tr>
<td>EXAMPLE TEXT</td>
<td>Keys on the keyboard, for example, F2 or ENTER.</td>
</tr>
</tbody>
</table>

**Icons**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔴</td>
<td>Caution</td>
</tr>
<tr>
<td>📚</td>
<td>Example</td>
</tr>
<tr>
<td>📚</td>
<td>Note</td>
</tr>
<tr>
<td>🌟</td>
<td>Recommendation</td>
</tr>
<tr>
<td>🌟</td>
<td>Syntax</td>
</tr>
</tbody>
</table>
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>About This Document</td>
<td>2</td>
</tr>
<tr>
<td>Scenario Overview</td>
<td>2</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>3</td>
</tr>
<tr>
<td>Disclaimer</td>
<td>3</td>
</tr>
<tr>
<td>Creating the WebDynpro UI</td>
<td>3</td>
</tr>
<tr>
<td>Setting Up a WebDynpro Project</td>
<td>3</td>
</tr>
<tr>
<td>Creating the WebDynpro Layout</td>
<td>6</td>
</tr>
<tr>
<td>Creating and Publishing Proxy Classes in CAF DC</td>
<td>9</td>
</tr>
<tr>
<td>Importing Proxy Classes in the WebDynpro DC</td>
<td>12</td>
</tr>
<tr>
<td>Create Model Nodes and Binding</td>
<td>14</td>
</tr>
<tr>
<td>Do the Implementation</td>
<td>20</td>
</tr>
<tr>
<td>Result</td>
<td>39</td>
</tr>
</tbody>
</table>
Introduction

About This Document

This tutorial is a part of a series that describes how to implement a composite application using SAP Composite Application Framework (CAF) capabilities.

The tutorial describes how to create a custom Web Dynpro user interface for your composite application.

Scenario Overview

The CAF Service and UI Layer series is based on a scenario, in which a company offers educational services to employees as participants and by employees as trainers. Educational services are offered at educational events. The company has offices at different locations both in the country and abroad, and employees need to travel between these locations. For cost-saving purposes the company decides that employees traveling on the same day between the same locations will share a car. This behavior shall be encouraged by certain financial benefits for the usage of carpools. The car can be either the personal company car of an employee or a car from the car pool of the company.

To help people find other travellers and set up car pools, the company needs a new application. This series demonstrates how to develop a prototype for such application using local data sources. The real-life productive system should work with data available in the HR (employees, travel planning, and compensations), MM (equipment management) and CRM (customer data) systems.

Data Model

The implementation of the application requires the use of the following business objects:

- Employee – the object is defined with a set of parameters, such as ID, first and last name, e-mail, and so on
- Location – this object represents travel destination; it is defined by fields, such as ID, address, city, and ZIP
- Car – the object represents either a pool car or a company car; it is assigned to an individual employee
- Travel – the object represents a trip between two locations; a travel is always one-way

The relations between them are shown in the figure below.

Data Model

An employee may create multiple travel instances. A travel always starts at a certain location and ends at a different one. Therefore a round trip to a location is maintained as two separate travel instances. Each travel is assigned to a single car. However, a car may be assigned to multiple travels limited by the number of available seats.
Functions
The basic functions of the application are:

- Master data maintenance for the following entities:
  - Employee
  - Location
  - Car
- Search for planned travels at a certain arrival date to a certain location
- Schedule travels based on the search results or assign a new car from the pool. If a pool car is not available, a message is displayed.

Prerequisites
The following table describes the prerequisites for running this tutorial.

<table>
<thead>
<tr>
<th>Software</th>
<th>The tutorial is compliant with:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Sneak Preview SAP NetWeaver 04 – Web Application Server 6.40 Java. You can download the sneak preview at <a href="http://sdn.sap.com">http://sdn.sap.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Documents</th>
<th>Before you start with this tutorial, see:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Configuring Your System for CAF Development</td>
</tr>
<tr>
<td></td>
<td>• Creating a Local Entity with Maintenance UI</td>
</tr>
<tr>
<td></td>
<td>• Creating Data Types</td>
</tr>
<tr>
<td></td>
<td>• Creating an Application Service</td>
</tr>
</tbody>
</table>

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.

Creating the WebDynpro UI

Setting Up a WebDynpro Project

1. Create a new development component (DC) of Web Dynpro type called carpool/travel.
2. In this DC, create a new Web Dynpro component called Travel. For package name enter \texttt{com.sap.carpool.travel}. Use the defaults to generate a window and an embedded view.

3. Create a new application called \texttt{TravelSearch} in the same package. Enable the option \textit{No authentication}.
4. Choose Next. Select *Use existing component*.
5. Choose Next. Select *Travel*.
6. Choose *Finish*.

The project structure is shown in the following figure.

7. Open the component controller, switch to the Context tab page, and create a new value node entitled *FindTravel* without structure binding. In the value node properties set the cardinality to $1..1$.

8. Add the following value attributes to the value node that you created:
   - StartLocation (type String)
   - TargetLocation (type String)
   - TravelDate (type Date)

9. Open the view *TravelView*.
10. Add the *Travel* component controller to the list of required controllers.
11. In the *TravelView*’s context add the same node with the same fields as described for the Travel component controller.
12. Create a context mapping between the relevant elements in the contexts of the view and the component controller.
13. In the Actions tab page of the TravelView editor, add the following actions without parameters:

- AddPassenger
- BrowseEmployee
- CreateTravel
- FindTravel
- Refresh

### Creating the WebDynpro Layout

Next you must create a Web Dynpro layout as shown in the figure.

The layout contains the following elements:

- **Group called Group1 with the following properties:**
  - Layout – Grid Layout
  - colCount – 2
  - Header – Find Carpool
The group described above must include all of the following elements:

- **Label**:
  - Id – StartLocationLabel
  - Text – Start Location

- **DropDownByKey**:
  - Id – StartLocation
  - Event – onSelect; triggers action Refresh

- **Label**:
  - Id – TargetLocationLabel
  - Text – Target Location

- **DropDownByKey**:
  - Id – TargetLocation
  - Event – onSelect; triggers action Refresh

- **Label**:
  - Id – StartDateLabel
  - Text – Travel Date

- **InputField**:
  - Id – StartDate

- **Button**:
  - Id – SearchButton
  - Text – Search
  - Colspan – 2
  - Event – onAction; triggers action FindTravel

- **Table**:
  - Id – TravelTable
  - Colspan – 2
  - paddingTop – large
  - header – Travels Found

The table includes the following columns (all are optional except the ID column):

- Id – IdColumn; cell editor – TextView; header – ID
- Id – TravelDateColumn; cell editor – TextView; header – Travel Date
- Id – StartLocationColumn; cell editor – TextView; header – Start Location
- Id – TargetLocationColumn; cell editor – TextView; header – TargetLocation
- Id – KeeperFirstNameColumn; cell editor – TextView; header – Keeper First name
- Id – KeeperLastNameColumn; cell editor – TextView; header – Keeper Last Name
- Id – CarIdColumn; cell editor – TextView; header – Car Tag

- **Table**
  - Id – PassengerTable
The table includes the following columns:
- Id – PassengerFirstNameColumn; cell editor – TextView; header – First Name
- Id – PassengerLastNameColumn; cell editor – TextView; header – Last Name
- Id – PassengerLastNameColumn; cell editor – TextView; header – Last Name

- Group:
  - Id - Group2
  - Layout – Grid Layout
  - colCount – 2
  - vAlign – middle
  - header – New Passenger

The group includes the following elements:
- Button
  - Id – ChooseEmployeeButton
  - Text – Choose Employee
  - Colspan – 2
  - Event – onAction; triggers action BrowseEmployee
- Label
  - Id – NPFirstNameLabel
  - Text – First Name
- InputField
  - Id – FirstName
  - readOnly – true
- Label
  - Id – NPLastNameLabel
  - Text – Last Name
- InputField
  - Id – LastName
  - readOnly – true
- Button
  - Id – AddPassengerButton
  - Text – Add As Passenger
  - Colspan – 2
  - Event – onAction; triggers action AddPassenger
- Button
  - Id – CreateTravelButton
  - Text – Create New Travel
Creating and Publishing Proxy Classes in CAF DC

First, you must generate the proxy classes in the composite application DC.

1. To do this, use the secondary mouse button to open the context menu of the carpool project in the Service Explorer view. Choose Create Web Dynpro Model.
2. You can check now in the Web Dynpro Explorer. The proxy classes are generated and can be browsed in the WebDynpro DC of the composite application under /src/packages/com/sap/carpool/services/.

3. In the Web Dynpro Explorer view, open the Web Dynpro project of the CAF DC.
4. Expand the DC Metadata node, select the node Public Parts, and open its context menu.
5. Choose New Public Part.

The proxy classes now exist in the CAF DC. To access them from another DC (the new Travel Web Dynpro DC), you need to make the classes public.
6. Enter carpool as the public part name.
7. Make sure that the option *Provides an API for developing/compiling other DCs* is enabled.
8. Choose Next.

9. Select *Java Package Tree* as entity type, and browse to select `com.sap.carpool.services.carpool`

10. Choose Finish, and save the metadata.

11. In the Service Explorer view, open the context menu of the CAF project, and choose *Generate Project Code*.

12. Finally, in the context menu choose *Development Component → Build*. 
Importing Proxy Classes in the WebDynpro DC

Now you must import the new generated classes into the Web Dynpro DC Travel.

1. In the Web Dynpro Explorer view of the Web Dynpro perspective, expand the Travel project tree.

2. Choose DC Meta Data → DC Definition → Used DCs, and open its context menu using the secondary mouse button.

3. Choose Add used DC.

   a. In the Web Dynpro project of the CAF DC, select DC metadata → Public Parts → package_tree → Entities → carpool.

   b. Choose dependency type Build Time.

5. Repeat the above steps to add the following public parts:
   - types_compilation in DC caf/core/dict (CAF domain)
   - com.sap.caf.ui.utils of DC caf/UI/ptn/common (CAF domain)
   - default of DC tc/col/runtime (SAP_JTECHS domain)

6. In the Web Dynpro Explorer view, expand the Travel project tree, and choose Web Dynpro → WebDynpro Components → Travel → Used Models.

7. Open the context menu and choose Add.

8. Choose the carpool model, and then OK.
Create Model Nodes and Binding

Next you will extend the Web Dynpro context with two additional nodes:

1. In the Web Dynpro Explorer view, open the Travel WebDynpro component controller.
2. In the Context tab page, create a new model node.

3. Enter Employee as model node name, and choose Finish.
4. Open the context menu of the new Employee node, and choose Edit Model Binding.
5. Select the model class AEmployee, and then choose Next.
6. Select the attributes `firstName` and `lastName`. Choose Finish.

The model node Employee now has two new attributes that are derived from the model class.

7. Repeat the procedure to create a new model node named `Travel`.
8. Bind the model class *ATravel* to the model node.

9. Create new attributes for the following properties of the model class:
   - car-id
   - car-manufacturer
   - car-model
   - car-numberSeats
   - keeper-firstName
   - keeper-lastName
   - passenger-firstName
   - passenger-lastName
   - startLocation-id
   - startLocation-name
   - targetLocation-id
   - targetLocation-name
   - id
   - startDate

As a result, the context should contain the entries, as shown in the figure below.

10. In the context tree, select *Travel* → *car* and change the value of its property *singleton* to *false*. You do this in *Properties* view. This is needed so that you can bind the subnode together with the supernode to a table.
11. Repeat the previous step for the following subnodes of the Travel node:
   o keeper
   o startlocation
   o targetLocation

12. Save the metadata.

13. Open TravelView, and go to the Context tab page.

14. Create the following model nodes.
   o Employee
   o Travel

   The procedure is the same as described for the component controller. However, you must create the nodes without model binding here.

15. Open the context menu of the Employee node and choose Edit Context Mapping.

16. Select Employee, and choose Next.
17. Choose to map the component controller’s context node to the Employee node in the view context. Make sure that the Employee node is selected with all its attributes and choose Finish.

18. Repeat the same procedure to map the Travel node in the component controller’s context to the Travel node in the view’s context.

19. Save the metadata.

As a result, the contexts must be the same for the Travel view and the Travel component controller. The controller contains the bindings, and the view simply refers to the controller. Thereby, the Model-View-Controller (MVC) architecture is applied in the component’s model.
Now you must bind the layout to the complete context.

20. Open TravelView and switch to the Layout tab page.

21. Bind the following UI elements to the relevant context nodes:

   o StartLocation element – set the selectedKey property to FindTravel \(\rightarrow\) StartLocation.
   
   o TargetLocation element – set the selectedKey property to FindTravel \(\rightarrow\) TargetLocation.
   
   o StartDate element – set the value property to FindTravel \(\rightarrow\) TravelDate.
   
   o Travel table – set the value of the dataSource property to the model node Travel.
   
   o For each column of the Travel table, set the text property of the table cell editors to the relevant attribute in the model node Travel.
   
   o Passenger table – set the value of the dataSource property to the model node Travel \(\rightarrow\) passenger.
   
   o For each column of the Passenger table, set the text property of the table cell editors to the relevant attribute in the model node Travel \(\rightarrow\) passenger.
   
   o firstName element – set the value property to the model attribute Employee \(\rightarrow\) firstName.
   
   o lastName element – set the value property to the model attribute Employee \(\rightarrow\) lastName.
As a result, the layout should look as shown in the following figure:

**Do the Implementation**

**Implementing a Simple Query to Get All Locations**

Next, you must implement the Web Dynpro component.

1. First, you must initialize the two drop-down boxes for the locations with all available values. To do this, you add the following coding to the method wdDoInit of the component controller:
// Initialize location fields:
IAспект aspectList;
IAспектRow aspectRow;
int count;

IWDАтрибут attributeInfoStart, attributeInfoTarget;
IпростыеTypeModifiable startLocation, targetLocation;
IModifiableSimpleValueSet valueSetStart, valueSetTarget;

// access the context and get the relevant value nodes:
attributeInfoStart =
    wдКонтекст.nodeFindTravel().getNodeInfo().getAttribute( "StartLocation");
startLocation = attributeInfoStart.getModifiableSimpleType();
valueSetStart =
    startLocation.getSVServices().getModifiableSimpleValueSet();
attributeInfoTarget =
    wдКонтекст.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"));
}
2. Add the following line in the import section of the controller:

```java
import com.sap.caf.carpool.services.carpool.travellocationservice.TravelLocationProxy;
```

3. Position the cursor somewhere in the source code, and use the secondary mouse button to open the context menu. Choose Source → Organize imports.

   If prompted, choose to import com.sap.tc.col.client.generic.api.IAspect.

4. Build, deploy and run the Travel Web Dynpro project.

If you have already created any TravelLocations instances in your CAF application, you should see a Web Dynpro screen similar to the one shown in the figure below. The travel locations are already retrieved and entered into the dropdown lists.

If the dropdown lists for the start and target locations are still empty, create some test entries using the CAF testing framework. For more information, see the tutorial Testing Entity and Application Services.

**Linking an Object Browser Pattern to a Web Dynpro Development Component**

Next, you must implement a possibility to search for employees that can be assigned to a selected travel instance.

This is a standard task, and you can use the Object Browser UI pattern. To configure it, you must use the ConfigBrowser application.

1. Start the CAF ConfigBrowser using the following link:

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

2. Choose the **Object Browser** pattern.

3. Choose **New Configuration**.
4. Enter `employeeBrowser` for the configuration name.

5. Enter `Find Employee` for the object browser title.

6. Enter `employeeSearchBar` for the search bar configuration name.

7. Enter `employeeBrowserList` for the object browser list configuration name.

8. Configure the employee search bar:
   a. Choose `Configure` next to the `SearchBar configuration name` field.
   b. Select service `sap.com/carpool/EmployeeService`.
   c. Select the `findByLastName` operation.
   d. Choose `Retrieve metadata`.
   e. Select the input parameter `lastName`.
   f. Add a default text for the search bar – for example, `Search for Employee`.
   g. Choose `Save + Return`. 
9. Configure the employee browser list:
   a. Choose Configure next to the Object BrowserList configuration name field.
   b. Select service sap.com/carpool/EmployeeService.
   c. For query name, choose findByLastName.
   d. Choose Display Aspect Fields.
   e. For displayed fields, select id, firstName, and lastName.
   f. For Web Dynpro Application Launched for Selected Object, choose development component sap.com/caf-UI-ptn-objecteditor, application ObjectEditor, and configuration createEmployee.
   g. Leave the field Label for Button for Object Assignment empty.
   h. Choose Save + Return.

10. To check whether the pattern works, choose Preview.
    If a search with an asterisk (*) does not return any results, you must create test employee entries, using the EmployeeSelector pattern. For more information about the configuration of this pattern, see the tutorial Creating a Local Entity with Maintenance UI.

11. Close the preview and choose Save + Return for the Object Browser pattern configuration.
You have configured the Object Browser pattern. Next you must make it available to the Web Dynpro DC.

12. First, you must add some required standard DCs.

In the Travel Web Dynpro project, choose DC MetaData \(\rightarrow\) DC Definition \(\rightarrow\) Used DCs. Open the context menu and choose Add Used DC.

Add the following DC dependencies:
- Build time dependency to the public parts of the DC caf/runtime/uicoupling
- Build time dependency to the public parts Common Interfaces and types_compilation of the DC caf/UI/ptn/common

13. Since the object browser pattern that you use is just another Web Dynpro component, you must declare its usage for the your DC:
   a. Select the node Used Web Dynpro Components, and open the context menu.
   b. Choose Add Used Component.
c. Enter `ObjectBrowser` as a name.


e. Choose Finish.

As a result, a new node called Object Browser appears in the project tree under *Used Web Dynpro Components*. Under this node, you can see the new interface controller for the component.

14. Open the new interface controller and switch to the *Context* tab page.

   The following parameters are available:
   
   o `configName` – the name of the configuration for the object browser pattern that you want to use
   
   o `parameter1` – the key of another entity service

15. Next, you must initialize a new object browser instance. This is done once only and therefore it is appropriate to do it during the initialization of the Web Dynpro component.
a. Open the Properties tab page of the Travel component controller, and add the object browser controller to the list of required controllers.

b. Add the following coding to the wdDoInit method of the Travel component:
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", "employeeBrowser");
    wdThis.
        .wdGetObjectBrowserInterface().
        wdGetAPI().
        .getContext().
        .getRootNode().
        .getCurrentElement().
        .setAttributeValue("parameter1", "");

    // Initialize the component
    wdThis.wdGetObjectBrowserInterface().initialize();

16. Now you must implement the call for the object browser instance
   
   a. Add a new method to the Travel component controller. It does not have any parameters,
      the return type is void, and the name is launchObjectBrowser.
   
   b. Add the following coding to the method’s implementation:

```
wdThis.wdGetObjectBrowserInterface().showWindow();
```
   
   c. Add the following coding to the action handler for the Choose Employee button. The
      handler is named onActionBrowseEmployee.

```
wdThis.wdGetTravelController().launchObjectBrowser();
```

17. To test your developments, save the metadata, build the Travel Web Dynpro component, deploy
    and run it.

    Click on Choose Employee, and check if the object browser appears.

    If you select an employee and choose Add to, the name will not be shown in the Web Dynpro
    application, because you have not implemented the data transfer yet. This is what you must do
    next.
18. Open the Travel component controller and add a new method of type `Event handler`.
   a. Enter `onBrowserReturn` for the method’s name.
   b. For event source choose the Object Browser component controller.
   c. For subscribed event choose `SelectionAdded`.
   d. Enable the option `Create event handler parameters according to subscribed event`.
   e. Choose `Finish`.

![New Method](image)

19. In the Implementation tab page, implement the `onBrowserReturn` method as follows:

   ```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));
   }
   ```

20. Choose `Source → Organize imports`.

21. Save, build the development component, 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 displayed there.
Creating Travel Entity Instances

So far, you have not created any travel instances. To create a new one, you must implement the relevant methods in the Travel component controller.

1. Open the Travel component controller and add a new method called `createNewTravel` with a return type `void`.

2. Add the following parameters to the method:
   
   - `startLocation` of type `String`
   - `targetLocation` of type `String`
   - `travelDate` of type `Date`
To create a new travel instance, you must also assign a car from the carpool. Therefore, you must create a new method for this task.

3. Add another method to the Travel component controller, called `getCarForDate`.
   As a return type, choose the proxy class `com.sap.carpool.services.carpool.carservice.ACar`. To do this, use the Java native type browsing.

4. Add a parameter called `travelDate` of type `Date`.

5. Implement the `getCarForDate` method. In a real-life application, you must perform a check for the car availability. However, such is not implemented in this tutorial for simplicity.
   
   In the Implementation tab page, add the following coding for the `getCarForDate` method:
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;

6. Choose Source → Organize imports.

7. Add the following coding in the implementation of the createNewTravel method:
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(
                carpoolDefinition.class);
8. Choose Source → Organize imports.

Now you must implement a call to the createNewTravel method in the TravelView. The method is called when the Create New Travel button is triggered.

9. In the Implementation tab page of the TravelView add the following code for the onActionCreateTravel event handler:

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

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

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

10. You should now be able to create new travels using the Web Dynpro UI.

To test it, save, build, deploy and run the application. Enter a start location, a target location, and a travel date, and select an employee. Choose Create New Travel. As a result, a new entry should appear in the table.
Implementing Travel Search Functions

You must now implement a query that searches for all car pools available on the selected date, and for all travels between the selected start and target locations.

1. Create a new method in the Travel component controller. The method is called `findTravels` and its return type is `void`.

2. Add the following parameters to the method:
   - `startDate` of type `Date`
   - `startLocation` of type `String`
   - `targetLocation` of type `String`
3. In the Implementation tab page implement the `findTravel` method by adding the following code between the “//@begin” and “//@end” tags.

```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 TravelView 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”.

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);
```

4. Save, build, deploy and run the application. Check if you can find the travels that you have just created.

**Updating an Entity Service**

You must now implement the possibility to add a new passenger to an appropriate car pool that you have found.
1. Add a new method to the Travel component controller called `addPassenger` with a return type `void` and without any parameters.

2. In the Implementation tab page of the component controller, implement the method as follows:
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 =
    // called by the server
Now you must link the button *Add passenger* to the new method.

3. Open the TravelView and switch to the implementation tab page.
4. Add the following code to the method `onActionAddPassenger`:
   ```java
   wdThis.wdGetTravelController().addPassenger();
   ```
5. Save, build, deploy and run your application.

**Result**

As a result of the implementation of this tutorial, you can:

- 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