How to Use the EJB Model in a Web Dynpro for Java Application

Applies to:
Web Dynpro for Java 7.11
For more information, visit the User Interface Technology homepage.

Summary
This tutorial covers all the steps necessary to create an EJB model and to use it within a Web Dynpro application. The tutorial application is a simple Web Dynpro application, which allows to search for cities upon entering a prefix of the city name. The business function to search for cities is provided by an Enterprise JavaBean (EJB) application, where the logic is implemented through a stateless session bean.

Author: Web Dynpro Java Team
Company: SAP AG
Created on: 29 June 2010
# Table of Contents

- Introduction .................................................................................................................. 3
- Prerequisites .................................................................................................................. 3
- Objectives ...................................................................................................................... 3
- Creating an EJB Development Component (DC) ............................................................ 4
  - Developing the Business Logic ................................................................................... 4
    - Specifying Dependencies to GeoNames data model ............................................. 5
    - Creating and Implementing the City class ............................................................ 5
    - Creating and Implementing the CityModel .......................................................... 8
    - Creating and Implementing the Session Bean ....................................................... 9
- Creating an Enterprise Application DC ........................................................................ 11
- Preparing DC’s for Use by Other Components ............................................................... 12
  - Specifying Permissions in the EJB DC ..................................................................... 12
  - Creating the Public Part in the Enterprise Application DC ...................................... 12
- Developing The Web Dynpro Application ................................................................... 14
  - Setting Up the Web Dynpro Development Component (DC) .................................. 14
    - Creating a Web Dynpro DC .................................................................................. 14
    - Specifying Dependencies ..................................................................................... 14
    - Defining the Basic Project Structure .................................................................... 15
    - Command Pattern in the EJB Model ..................................................................... 17
  - Creating the Context ................................................................................................ 17
    - Adding a Model to the Web Dynpro Component .................................................. 17
    - Binding the Component Controller Context to the EJB Model ......................... 18
    - Mapping the Controllers ...................................................................................... 19
  - Instancing, Registering and Executing the Model ....................................................... 20
    - Registering the Model in the Component Controller .......................................... 20
    - Implementing a Method to Call the Model ............................................................ 20
  - Specifying the View Layout and the Data Binding ................................................... 21
    - Defining the UI Elements ..................................................................................... 22
- Tutorial Result ................................................................................................................ 23
  - Deploying and Running the Sample Application ....................................................... 23
- Restrictions .................................................................................................................... 24
- Further Information ...................................................................................................... 24
- Legend 24
  - Text Symbols ........................................................................................................... 24
- Copyright ....................................................................................................................... 25
Introduction

This tutorial covers all the steps necessary for using existing business functions from within a Web Dynpro application. The business functions are provided in the form of an Enterprise JavaBean (EJB) application where the business logic is implemented through a stateless session bean. In order to use the existing functions in the Web Dynpro application, we will import a model that facilitates its usage.

In this sample application we implement a simple business logic in an EJB 3.0 session bean, while the user interface is developed using Web Dynpro for Java. The goal of this tutorial is to provide a simple Web Dynpro application which allows you to search for cities upon entering a prefix of the city name.

Prerequisites

You need to install the NetWeaver Developer Studio (Version 7.11 or later) in order to compile and deploy the tutorial application. The SAP Java AS to which this application is deployed should have the same or newer version as the NWDS. The screen shots and descriptions however are all created with version 7.11 and so there could be minor deviations, if you use a version greater than version 7.11.

The tutorial application is available as 3 development components (DC’s). You need to import the software component HM-WDUIDMKTCDNT, which contains the DC’s:

- tc/wd/tut.ejb/cities
- tc/wd/tut.ejb/app
- tc/wd/tut.ejb/wdapp.

The exact steps are described in a separate document.

Knowledge

- You are experienced in working with the SAP NetWeaver Developer Studio.
- You have acquired some basic experience with Web Dynpro applications.
- You have basic knowledge of the EJB 3.0 programming model.

Objectives

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

- Set up Development Component (DC) based projects in the Developer Studio.
- Prepare Java EE 5 DC projects for use by other components (such as in Web Dynpro).
- Generate a model to be used for linking up the business logic of the EJB DC from within the Web Dynpro DC.
- Create the context in the controllers in such a way so that a connection to the model is defined.
- Define the data binding for the UI elements.
- Add the Java code required when accessing the model in Web Dynpro.
Creating an EJB Development Component (DC)
To develop the EJB layer to be integrated into Web Dynpro, you first need to create an EJB Development Component (in short: DC) project.

- To create a new DC based project of type EJB, proceed as follows:
- Choose the menu path File → New → Project to start the New Project Wizard.
- Expand the Development Infrastructure and select Development Component. Choose Next.
- Expand the root node with the vendor ID (for example: sap.com) and select J2EE and then EJB Module. Choose Next.

The wizard generates a new EJB DC project. As soon as generation is complete, you might be asked to switch to the J2EE Perspective. The new project is then displayed in the Project Explorer view.

Developing the Business Logic
The tutorial application uses the geographical database “GeoNames” as base of the business logic, especially the cities in this database are used.

The "GeoNames" database is available at http://www.geonames.org/ under a Creative Commons License.

The CityBean, which is described here, uses existing DC’s from our tutorial applications, which expose this model with the corresponding methods. It encapsulates this GeoNames model as EJB model specializing it to a CityModel. The mentioned DC’s are:

- tc/wd/tut/data/geodata
- tc/wd/tut/data/geolib

Now, you need to create and implement the different classes, and the session bean CityBean, that defines the business methods required in our sample scenario.
Specifying Dependencies to GeoNames data model

- In the context menu of the EJB DC project choose Development Component → Show In → Component Properties.
- Choose Dependencies → Add…
- Expand the the node with your Development Configuration and select the tc/wd/tut/data/geodata DC. Choose Finish.

Creating and Implementing the City class

- In the Project Explorer, expand your EJB DC.
- In the context menu of ejbModule, choose New → Class.
- Enter com.sap.test.tc.wd.tut.ejb.cities in the Package field. Enter City in the Name field. Choose Finish. You have created the City class.
- To implement the class, open the Java editor and add the following source code:

```java
package com.sap.test.tc.wd.tut.ejb.cities;
import java.sql.Date;
import java.util.List;
import com.sap.test.tc.wd.tut.data.geo.model.GeoRecord;

/**<pre>
The main 'geoname' table has the following fields :
-------------------
geonameid : integer id of record in geonames database
name : name of geographical point (utf8) varchar(200)
asciname : name of geographical point in plain ascii characters, varchar(200)
alternatenames: alternatenames,comma separated varchar(4000) (varchar(5000) for SQL Server)
latitude : latitude in decimal degrees (wgs84)
longitude : longitude in decimal degrees (wgs84)
feature class : see http://www.geonames.org/export/codes.html, char(1)
feature code : see http://www.geonames.org/export/codes.html, varchar(10)
country code : ISO-3166 2-letter country code, 2 characters
cc2 : alternate country codes, comma separated, ISO-3166 2-letter country code, 60 characters
admin1 code : fipscode (subject to change to iso code), isocode for the us and ch, see file admin1Codes.txt for display names of this code; varchar(20)
admin2 code : code for the second administrative division, a county in the US, see file admin2Codes.txt; varchar(80)
admin3 code : code for third level administrative division, varchar(20)
```
admin4 code : code for fourth level administrative division, varchar(20)
population : bigint (4 byte int)
elevation : in meters, integer
gtopo30 : average elevation of 30'x30' (ca 900mx900m) area in meters, integer
timezone : the timezone id (see file timeZone.txt)
modification date : date of last modification in yyyy-MM-dd format

These fields are mapped to corresponding names for City records
</pre>
*/

public class City {
  public String altNamesAsString;
  private final GeoRecord geoRecord;

  protected City( GeoRecord geoRecord ) {
    this.geoRecord = geoRecord;
  }

  // accessors
  public String getCityId() {
    return geoRecord.getGeoNameId();
  }
  public String getName() {
    return geoRecord.getName();
  }
  public String getAsciiName() {
    return geoRecord.getAsciiName();
  }
  public List<String> getAltNames() {
    return geoRecord.getAltNames();
  }
  public String getAltNamesAsString() {
    StringBuffer altNameBuffer = new StringBuffer();
    boolean first = true;
    for( String altName : geoRecord.getAltNames() ) {
      if ( !first )
        altNameBuffer.append(", ");
      else
        first = false;
      altNameBuffer.append(altName);
    }
    return altNameBuffer.toString();
  }
  public String getLatitude() {
    return geoRecord.getLatitude();
  }
  public String getLongitude() {
public String getFeatureClass()
{
    return geoRecord.getFeatureClass();
}
public String getFeatureCode()
{
    return geoRecord.getFeatureCode();
}
public String getCountryCode()
{
    return geoRecord.getCountryCode();
}
public List<String> getAltCountryCodes()
{
    return geoRecord.getAltCountryCodes();
}
public String getRegionCode()
{
    return geoRecord.getAdmin1Code();
}
public String getDistrictCode()
{
    return geoRecord.getAdmin2Code();
}
public String getAdmin3Code()
{
    return geoRecord.getAdmin3Code();
}
public String getAdmin4Code()
{
    return geoRecord.getAdmin4Code();
}
public Integer getPopulation()
{
    return geoRecord.getPopulation();
}
public Integer getElevation()
{
    return geoRecord.getElevation();
}
public Integer getGtopo30()
{
    return geoRecord.getGtopo30();
}
public String getTimeZone()
{
    return geoRecord.getTimeZone();
}
public Date getModificationDate()
{
    return geoRecord.getModificationDate();
}

// additional fields used for OVS
Creating and Implementing the CityModel

The CityModel class is only a proxy, which delegates the model relevant methods to the GeoNamesModel.

```java
package com.sap.test.tc.wd.tut.ejb.cities;
import java.util.ArrayList;
import java.util.List;
import com.sap.test.tc.wd.tut.data.geo.model.GeoNamesModel;
import com.sap.test.tc.wd.tut.data.geo.model.GeoRecord;

class CityModel {
    // the wrapped GeoNames model
    private static GeoNamesModel geoNamesModel = GeoNamesModel.getInstance();

    protected static City findRecord(String geoNameId) {
        GeoRecord record = geoNamesModel.findRecord(geoNameId);
        if (record != null) return new City(record);
        else return null;
    }

    public static List<City> findMatchingRecords(String prefix, boolean includeAlternateNames) {
        List<GeoRecord> geoRecords = geoNamesModel.findMatchingRecords(prefix, includeAlternateNames);
        List<City> cityList = new ArrayList<City>();
        for (GeoRecord geoRecord : geoRecords) {
            cityList.add(new City(geoRecord));
        }
        return cityList;
    }

    public static String getCountryName(String countryIsoCode) {
        return geoNamesModel.getCountryName(countryIsoCode);
    }

    public static List<String> getLanguages(String countryIsoCode) {
        return geoNamesModel.getLanguages(countryIsoCode);
    }

    public static String[] getCountryIsoCodes() {
        return geoNamesModel.getCountryIsoCodes();
    }

    public static String getRegionName(String countryCode, String regionCode) {
        return geoNamesModel.getRegionName(countryCode, regionCode);
    }
}
```
Creating and Implementing the Session Bean

- In the Project Explorer, expand your EJB DC.
- In the context menu of ejbModule, choose New → Other. Expand EJB and choose EJB Session Bean 3.0. Choose Next.
- Enter CityBean in the EJB Name field and com.sap.test.tc.wd.tut.ejb.cities in the Default EJBPpackage field. Set the Local option and choose Next.
- The next screen confirms that the Business Interface CityLocal is automatically created. Choose Finish.
- To implement the CityBean session bean, use the following source code:

```java
package com.sap.test.tc.wd.tut.ejb.cities;
import java.util.List;
import javax.ejb.Stateless;
@Stateless(name="CitiesBean")
public class CitiesBean implements CitiesLocal {
    public City findCity(String cityId) {
        return CityModel.findRecord(cityId);
    }
    public List<City> findMatchingCities(String cityPrefix, boolean includeAlternateNames) {
        return CityModel.findMatchingRecords(cityPrefix, includeAlternateNames);
    }
    public String getCountryName(String countryIsoCode) {
        return CityModel.getCountryName(countryIsoCode);
    }
    public List<String> getLanguages(String countryIsoCode) {
        return CityModel.getLanguages(countryIsoCode);
    }
    public String[] getCountryIsoCodes() {
        return CityModel.getCountryIsoCodes();
    }
    public String getRegionName(String countryCode, String regionCode) {
        return CityModel.getRegionName(countryCode, regionCode);
    }
    public String getDistrictName(String countryCode, String regionCode, String districtCode) {
        return CityModel.getDistrictName(countryCode, regionCode, districtCode);
    }
}
```

- Switch to CityLocal.java and insert the following source code:

```java
package com.sap.test.tc.wd.tut.ejb.cities;
import java.util.List;
import javax.ejb.Local;
```
Result
You have created and implemented all classes and interfaces you need for your EJB DC.

```java
@Local
public interface CitiesLocal {
    public City findCity(String geoNameId);
    public List<City> findMatchingCities(String cityPrefix, Boolean includeAlternateNames);
    public String getCountryName(String countryIsoCode);
    public List<String> getLanguages(String countryIsoCode);
    public String[] getCountryIsoCodes();
    public String getRegionName(String countryCode, String regionCode);
    public String getDistrictName(String countryCode, String regionCode, String districtCode);
}
```
Creating an Enterprise Application DC

To deploy your EJB to the server, you need to package it with an Enterprise Application development component.

To create a new Development Component based project of type Enterprise Application, proceed as follows:

- Choose the menu path File → New → Project. A New Project Wizard appears.
- Choose Development Infrastructure → Development Component. Choose Next.
- Expand the root node with the vendor ID (for example: sap.com) and select J2EE, and then Enterprise Application. Choose Next.
- Expand the node with your Development Configuration and select the SC wd_uidemokit_HM_WDUIDMKTNCNT_test_sap_com/test.sap.com. Choose Next.
- Enter tc/wd/tut/ejb/app in the Name field. Choose Next.
- Select 5.0 for Java EE version. Choose Next.
- Assign the reference to the EJB DC project you have created in the last procedure and choose Finish.
- Specifying Dependencies to GeoNames data model as before for EJB DC.

Result

The wizard generates a new Enterprise Application DC. The resulting project structure is displayed in the Project Explorer view.
Preparing DC’s for Use by Other Components

In following steps, you have to specify some DC properties in such a way that the EJBs defined in the EJB DC can be used from another DC’s.

Specifying Permissions in the EJB DC

- Select the project node for your EJB DC project and choose Development Component → Show in Component Properties in the context menu.
- A multi page editor appears where you can use to specify DC-specific properties.
- Select the Permissions tab and activate the option Entity Forwarding Allowed.

Creating the Public Part in the Enterprise Application DC

- Select the project node for your Enterprise Application DC project (tc/wd/tut/ejb/app) and choose Development Component → Show in Component Properties in the context menu.
- In the properties editor, select the Public Parts tab.
- Choose Add… to create a new public part. Enter cityejb_pp in the Name field and select Compilation as Purpose option. Then choose Finish.
- Select the node for just created public part and choose Reference Entities in the context menu.
- In the next screen, you have to reference the public part of the EJB DC.
- Expand the node with your Development Configuration and select the SC wd__uidemokit_HM_WDUIDMKTCTNT_test_sap_com[test.sap.com] and select the EJB DC (tc/wd/tut/ejb/cities). Then mark the client public part.
• Choose Finish.

• To perform the Build for Enterprise Application DC, select the project node again and choose Development Component → Build.

Result
You have specified all properties required for importing EJBs using the Enterprise JavaBean model in Web Dynpro.
Developing The Web Dynpro Application

In compliance with the MVC paradigm, a model in Web Dynpro enables the user to access business data that is stored outside the Web Dynpro application. In our case the business data that belongs to our sample application is defined by means of EJBs.

Within the Web Dynpro project, you will obtain such a model as a result of proxy generation. It is represented by a set of model classes, including their relations. The model mainly consists of special model classes that you can use to link context structures to the model.

After importing the model information, we only need to bind the controller context to corresponding model objects. Each context has a node that represents the corresponding model object. Finally, we only need to take care of the link from the UI elements to the business data referenced in the view controller context (data binding for the Web Dynpro view’s UI elements).

Setting Up the Web Dynpro Development Component (DC)

Creating a Web Dynpro DC

- Choose the menu path File → New → Project to start the New Project Wizard.
- Expand the Development Infrastructure and select Development Component. Choose Next.
- Expand the root node with the vendor ID (for example: sap.com) and select Web Dynpro. Choose Next.
- Expand the node with your Development Configuration and select the SC wd__uidemokit_HM_WDUIDMKT_CNT_test_sap_com [test.sap.com]. Choose Next.
- Enter tc/wd/tut/ ejb/ wdapp in the Name field to specify the name of the DC. Leave the other settings unchanged and choose Finish.

Specifying Dependencies

- In the context menu of the Web Dynpro project choose Development Component → Show In → Component Properties.
- Choose Dependencies → Add…
- Expand the node with your Development Configuration and select the tc/wd/tut/ ejb/ app project. Choose Finish.

```
<table>
<thead>
<tr>
<th>tc/wd/tut/ ejb/ wdapp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
</tr>
</tbody>
</table>
```

Required DCs

Specify the list of DCs required for this DC:

- tc/aii/base/offline/facade
- tc/bl/exception/lib
- tc/bl/jrfc/api
- tc/bl/logging/api
- tc/cm/ ejb/api
- tc/ct
- tc/ddic/runtime/facade
- tc/ ejblib/api
- tc/wd/api
- tc/wd/tut/ ejb/app
- city/ ejb_pp
Defining the Basic Project Structure
- Expand the Web Dynpro node in the structure of the project.
- To start the relevant wizard, open the context menu of the Applications node and choose Create Application.
- Enter a name for your Web Dynpro application, such as TutorialApp, and specify the package name for the Java classes to be generated. Then choose Next.
- Accept the selection Create a new component Tutorial and choose Next again.
- Accept also the predefined settings for the window, the interface view, and the startup plug. Choose Finish.

Creating the EJB Model
- In the project structure, expand the node Web Dynpro → Models. From the context menu, choose Create Model to start the appropriate wizard.
- Choose the Enterprise JavaBean Model option, followed by Next.
- Enter CityModel in the Model Name field and specify com.sap.test.tc.wd.tut.ejb.wdapp.model as the Model Package. Choose Next.
- Select the Enterprise Application DC tc/wd/tut/ejb/app that provides the public part with the Enterprise JavaBean required for model import. Choose Next.
- A list of Session Beans interfaces and their business methods appear. Select **findMatchingCities** method from the **CitiesLocal** interface. Choose **Next**.

- A list of model classes to be generated, is displayed on the next screen. Choose **Next**.

- Start the model generation routine with **Finish**.
Result
You have now created a model named CityModel in your Web Dynpro project. In accordance with the MVC paradigm, the model was not simply generated as part of the Web Dynpro component, but as an independent development object.

The generated model classes are now visible within the project structure in the Web Dynpro Explorer under the Models node.

Command Pattern in the EJB Model
Command pattern means to encapsulate an operation and give it an object representation. It is quite often used in Web Dynpro and is also applied in the newly introduced EJB Model. According to the command pattern, each business method of the Enterprise JavaBean is represented by a so called Request model class. The Request model class contains all the business method input parameters either as properties (in case of simple types) or relations (in case of JavaBeans, collections or arrays). If the business method has a return type, then the corresponding Response model class is created as well and linked as a relation in the Request model class.

Creating the Context
In the procedure below, you first add the newly created model to the list of used models of your Web Dynpro application. Then you will create the necessary context structure.

Adding a Model to the Web Dynpro Component
- Expand the project structure Components → Tutorial → Used Models.
- In the context menu of the Used Models node, choose Add and select the Model you have created in the last step.
- Confirm with OK.
Binding the Component Controller Context to the EJB Model

- Select your component Tutorial in the Web Dynpro Explorer and choose Open Component Modeler from the context menu.
- Select Create Data Link and draw an arrow from the Component controller to the Model as shown in the screenshot below. The Edit Model Binding wizard opens.

- Choose Request_City_Local_findMatchingCities model class, in the following screen select all and choose Finish.
- The Request model class should be mandatory for the context. This forces the Web Dynpro context to implicitly create its instance (with all of its mandatory relations in depth).
- Navigate to Properties tab of the Request_CitiesLocal_findMatchingCities context node and change the Collection Cardinality property to 1..1.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td></td>
</tr>
<tr>
<td>Model Class</td>
<td>CityModel.Request_CitiesLocal_findMatchingCities</td>
</tr>
<tr>
<td>Supplying Relation Role</td>
<td></td>
</tr>
<tr>
<td>Misc</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Request_CitiesLocal_findMatchingCities</td>
</tr>
<tr>
<td>Collection Cardinality</td>
<td>1..1</td>
</tr>
<tr>
<td>Initialize Load Selection</td>
<td>true</td>
</tr>
<tr>
<td>Selection Cardinality</td>
<td>0..1</td>
</tr>
<tr>
<td>Singleton</td>
<td>true</td>
</tr>
<tr>
<td>_supply Function</td>
<td></td>
</tr>
<tr>
<td>Typed Access Required</td>
<td>true</td>
</tr>
<tr>
<td>Documentation</td>
<td></td>
</tr>
<tr>
<td>Quick Info</td>
<td></td>
</tr>
<tr>
<td>Technical Documentation</td>
<td></td>
</tr>
</tbody>
</table>

Mapping the Controllers
- Select Create Data Link again and draw an arrow from the View Controller TutorialView to the Component Controller.
- Choose the Request_CitiesLocal_findMatchingCities node and map it to the Root node of the view context TutorialView Confirm with Finish.

Result
You have created the necessary view context structure and also bound the context nodes to the corresponding model classes.

![Diagram](image-url)
Instancing, Registering and Executing the Model

To be able to send a request to the EJB, you must ensure that an instance of the model class is available. That’s why you have to instantiate and register the model class in the `wdDoInit()` method.

In addition you will create a method `findMatchingCities` in which the model instance is executed. This method will be called later on from an action event handler in the view controller.

Registering the Model in the Component Controller

Add the following source code in the `wdDoInit()` method of the component controller.

```java
public void wdDoInit() {
    //@@begin wdDoInit()
    CityModel myModel = new CityModel();
    wdComponentAPI.getModelInstanceMap().putDefaultInstance
        ("com.sap.test.tc.wd.tut.ejb.wdapp.model.CityModel", myModel);
    //@@end
}
```

Implementing a Method to Call the Model

- Switch to the Methods tab of the component controller Tutorial and add a new method called `findMatchingCities` with return Type `void`.
- Add the following source code:

```java
public void findMatchingCities( ) { 
    //@@begin findMatchingCities()
    try {
        wdContext.currentRequest_CitiesLocal_findMatchingCitiesElement().
            modelObject().execute();
    } catch (Exception e) {
        wdThis.wdGetAPI().getMessageManager().reportException(e);
        logger.throwing(e);
    }
    //@@end
}
```
Specifying the View Layout and the Data Binding

Now you can design the view:

- An InputField and a Checkbox is used as input for the City search and are bound to the corresponding attributes of the Request node (*Request_CitiesLocal_findMatchingCities*) in the context. This is given as a parameter to the model call.

  ```java
  public void onActionSearch(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent)
  { //80begin onActionSearch(ServerEvent)
    wdThis.wGetTutorialController().findMatchingCities();
  //80end
  ```

- A Button so that the user can trigger the event to call the model. In detail the event triggered when the button is clicked calls the action event handler of the view controller and this calls the execute method in the component controller.
A Table that displays the Cities data we received from the EJB.

<table>
<thead>
<tr>
<th>City Name</th>
<th>Country Code</th>
<th>Region Code</th>
<th>Population</th>
<th>Timezone</th>
<th>alternate Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamburg</td>
<td>DE</td>
<td>08</td>
<td>8,672</td>
<td>Europe/Paris</td>
<td></td>
</tr>
<tr>
<td>Hamburg</td>
<td>DE</td>
<td>04</td>
<td>2,835</td>
<td>Europe/Paris</td>
<td></td>
</tr>
<tr>
<td>Hamburg</td>
<td>DE</td>
<td>04</td>
<td>8,672</td>
<td>Europe/Paris</td>
<td></td>
</tr>
</tbody>
</table>

Defining the UI Elements

Cause the intention of this tutorial is not to show in detail, how to create the different UI elements, I will only show how the View Layout should look at the end:
Tutorial Result

Deploying and Running the Sample Application

- First you have to deploy the Java EE Application `tc/wd/tut.ejb/app`
- Then deploy and run the Web Dynpro Application
  - In the Web Dynpro Explorer, open the context menu for the application object `TutorialApp`.
  - Choose Deploy new Archive and Run.

Then you should see the following screen (after you have entered e.g. “Wall” and clicked “Search”)

![City Search](image-url)

City Name | Country Code | Region Code | Population | Timezone |
---|---|---|---|---|
Wallasey | GB | ENG | 58,794 | Europe/London | WALLasey
Wallbrum | DE | 01 | 12,113 | Europe/Berlin | WALLbrum
Wallenborn | DE | 06 | 24,201 | Europe/Berlin | WALLenborn
Wallenhorst | DE | 08 | 30,104 | Europe/Berlin | WALLenhorst
Waller | US | WA | 8,803 | America/Detroit | WALLer
Wallers | FR | 54 | 5,506 | Europe/Paris | WALLers
Wallersdorf | DE | 02 | 8,702 | Europe/Berlin | WALLersdorf
Wallsend | GB | ENG | 3,199 | Europe/London | WALLsend
Wallston | US | CT | 17,712 | America/New_York | WALLston
Wallton | US | NJ | 11,469 | America/New_York | WALLton
Walltown | CH | ZH | 11,705 | Europe/Zurich | WALLtown
Wallerstein / Wallstein-Ost | CH | ZH | 5,031 | Europe/Zurich | WALLstein-Ost
Wallvel | GB | ENG | 42,739 | Europe/London | WALLvel
Wallsf | DE | 55 | 5,658 | Europe/Berlin | WALLsf
Restrictions
As is customary, the features described here are only guaranteed to work in browser versions that are officially supported by the described Web Dynpro release.

Further Information
For more information, see the Web Dynpro Java section of the SAP Developer Network (SDN) http://www.sdn.sap.com/irj/sdn/nw-wdjava

Legend

Text Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Note</td>
</tr>
<tr>
<td></td>
<td>Recommendation</td>
</tr>
<tr>
<td></td>
<td>Warning</td>
</tr>
<tr>
<td></td>
<td>See also</td>
</tr>
</tbody>
</table>
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.