# ADF Code Corner

## 92. Caching ADF Web Service results for in-memory filtering


twitter.com/adfcodecorner

**Abstract:**

Querying data from Web Services can become expensive when accessing large data sets. A use case for which Web Service access can be avoided is when filtering table data as it can be done in memory. In Oracle ADF, you access Web Service from JAX-WS proxy clients or the Web Service Data Control. While the Web Service data control does not allow to intercept data queried from Web Services, the jAX-WS proxy client does.

This article shows how to use the JAX-WS client proxy with Oracle ADF to access Web Service and locally cache data for further data operations.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
31-OCT-2011

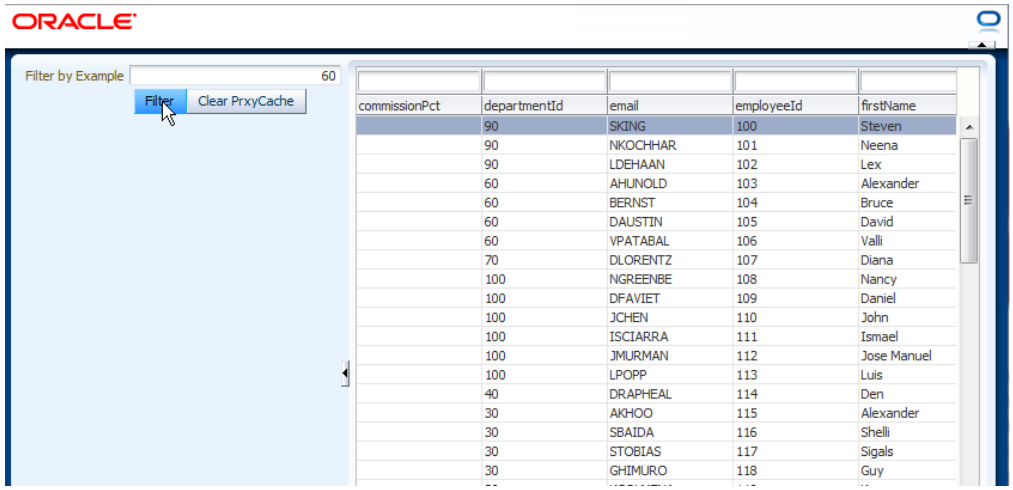
*Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.*

*Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.*

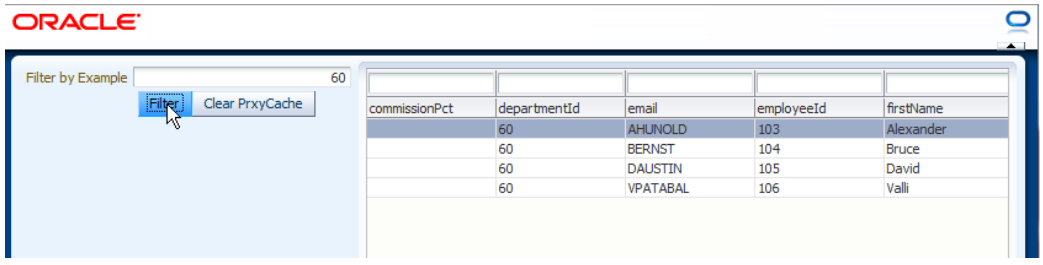
*Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83*

## Introduction

In the example, an EJB based Web Service queries data of the Employees table in the HR schema to display in an ADF bound table. The table is configured to allow users to filter the displayed data by typing search conditions into the search fileds shown in the column headers.

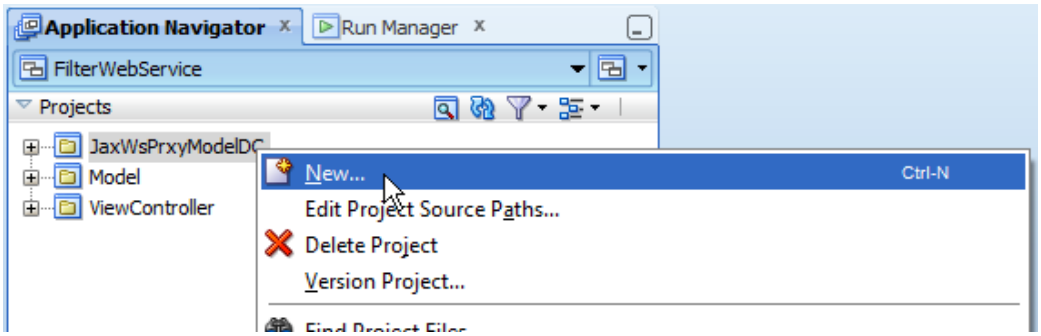



Filtering the table data results in another query sent to the web service. In the example however, both the search filter fields in the column headers and the custom filter field accessing a method exposed on the data control filter the table data in memory, meaning that no request is sent to the webService.

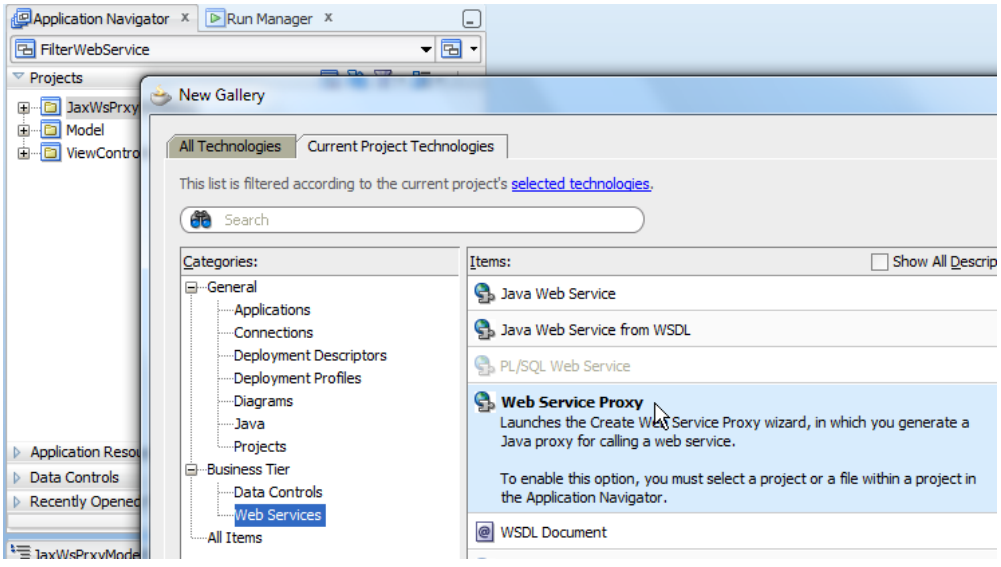


## Building the Web Services Proxy Client

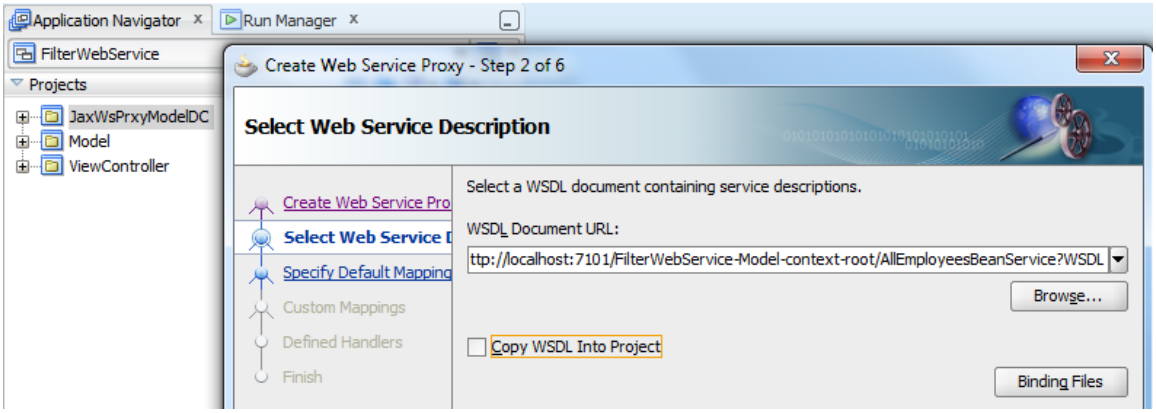
To be able to cache the queried Web Service data displayed in the table, a JAX-WS proxy client is needed to read the data from the Web Service. The proxy client is then accessed from a POJO bean that becomes the ADF Data Control application developers work with. This way, developers get a chance to intercept the Web Service request and response and also protect custom code from the impact of re-generating the proxy client (which may be required when the remote Web Service API changes).




To create a Web Service client proxy, create a new Oracle JDeveloper project and configure it for Web Service support. Select the project, **JaxWsPrxyModelDC** in the example, and choose **New** from the right mouse context menu. In the opened **New Gallery** select **Business Tier | Web Service Proxy** and press **Ok.**

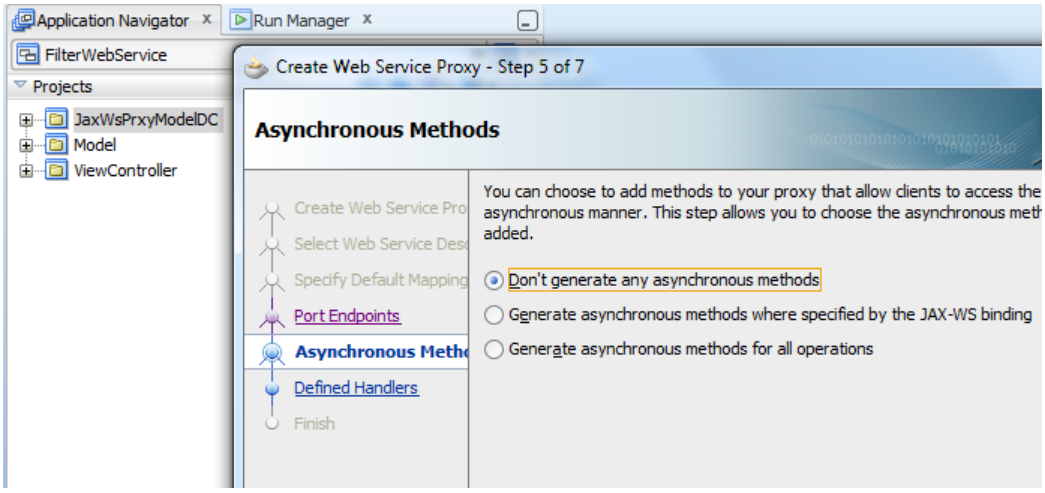



In the second dialog of the web Service proxy creation wizard, add the WSDL reference of the remote Web Service.

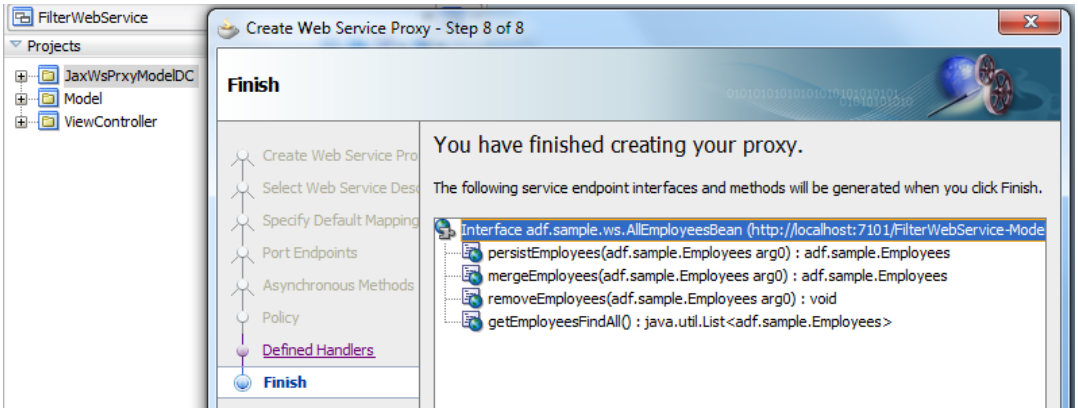



Note: The Web Service in the example is created from an EJB business service created in the **Model** project. In a real scenario, the Web Service would be remote and not contained in the application itself (for demoing Web Services however, it is quite convenient)

In the following dialog, define a base package and special "types" package for the Java artifacts that are getting created based on definitions in the WSDL file.



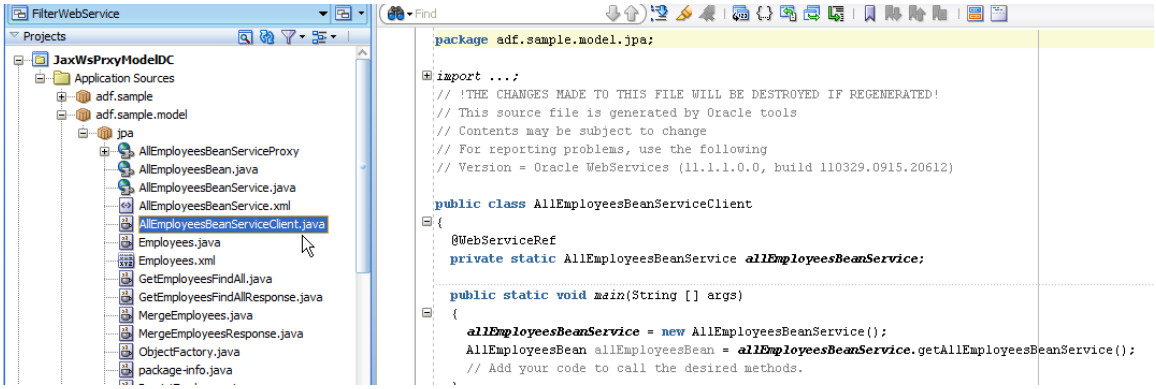

This sample doesn't require asynchronous Web Service access, so that this option can be switched off.




The last screen summarizes the methods exposed by the Web Services, which also become available on the JAX-WS proxy.

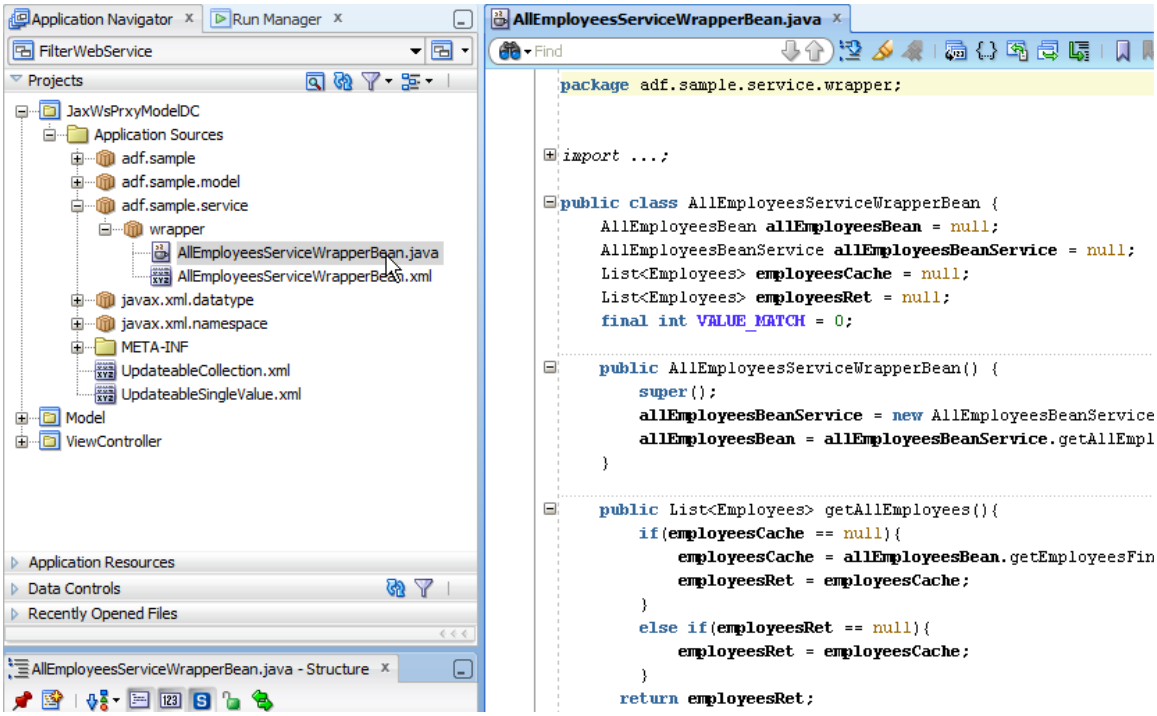
## Using a Java Bean wrapper class as the Data Control

The Web Services client can be tested using the Client **class** which has a **main** method defined. In the sample, the client class is **AllEmployeesBeanServiceClient**.




To access a Web Service client proxy from Oracle ADF, best practice is to do this through a POJO bean access in the proxy class, so the Oracle ADF data control access is decoupled from the implementation of the proxy class. This allows re-generating the Web Service proxy without impacting the Oracle ADF data control access. In the example, this POJO is **AllEmployeesServiceWrapperBean.java.**

The POJO accesses the client proxy and exposes the methods to query the Web Services. Within the POJO, the returned data from the Web Service are cached for later use. In the following, whenever the table data is queried, the Java code first check if the data already exists in the cache and if, it takes it from there.



```
import adf.sample.model.jpa.AllEmployeesBean;
import adf.sample.model.jpa.AllEmployeesBeanService;
import adf.sample.model.jpa.Employees;
import java.util.ArrayList;
import java.util.List;


/**
 * Java Bean that is exposed as a data control. The bean accesses the
 * JAX-WS proxy client to expose methods of the Web Service
*/
public class AllEmployeesServiceWrapperBean {
  //Web Service proxy client class
  AllEmployeesBean allEmployeesBean = null;
  AllEmployeesBeanService allEmployeesBeanService = null;
  //List object for caching
  List<Employees> employeesCache = null;
  List<Employees> employeesRet = null;
  final int VALUE_MATCH = 0;

  public AllEmployeesServiceWrapperBean() {
   super();
   //connect to the proxy client
   allEmployeesBeanService = new AllEmployeesBeanService();
   allEmployeesBean =
      allEmployeesBeanService.getAllEmployeesBeanService();
  }

  //query all employees. This method is called when the ADF table
  //executes its underlying iterator
  public List<Employees> getAllEmployees(){
    if(employeesCache == null){
      employeesCache = allEmployeesBean.getEmployeesFindAll();
      employeesRet = employeesCache;
    }
    else if(employeesRet == null){
      employeesRet = employeesCache;
    }
    return employeesRet;
  }

  //To some point, you may want to re-execute the service to get the
  //latest data.
  //In this case, the cache List is set to null.
```

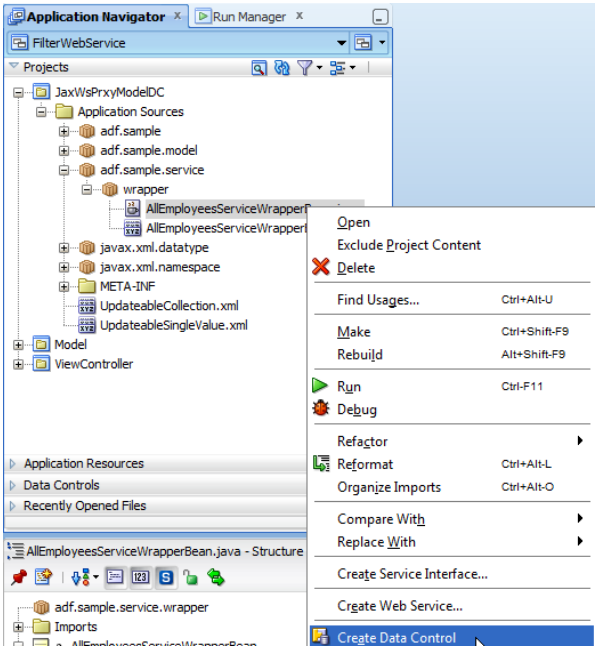
```
  public void clearProxyCache(){
    employeesCache = null;
  }

  //to filter the data displayed in the table, an extra method is
  //exposed on the POJO and thus the Data Control. Instead of querying
  //the Web Service, this method operates on the cached data
  public void filterEmployeesByDepartmentId(Long departmentId){
    employeesRet = new ArrayList<Employees>();
    if(departmentId == null){
      employeesRet = employeesCache;
      return;
    }

    for(Employees emp : employeesCache){
      if(emp.getDepartmentId() != null &&
         emp.getDepartmentId().compareTo(departmentId)== VALUE_MATCH){
         employeesRet.add(emp);
      }
    }
  }
}
```

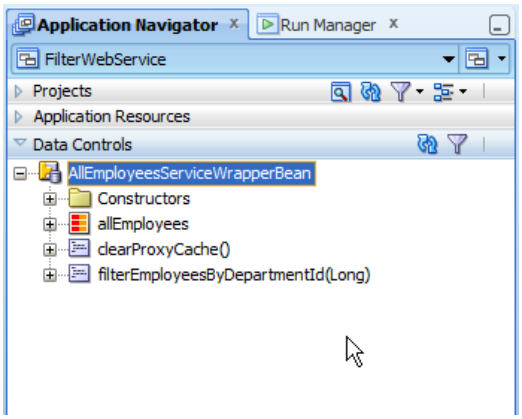
The `AllEmployeesServiceWrapperBean` is the turned into a Data Control to expose its methods – and thus the functionality exposed on the Web Service – to Oracle ADF.

For this, select the `AllEmployeesServiceWrapperBean` file in the Oracle JDeveloper **Application Navigator** and choose **Create Data Control** from the context menu as shown in the image below.



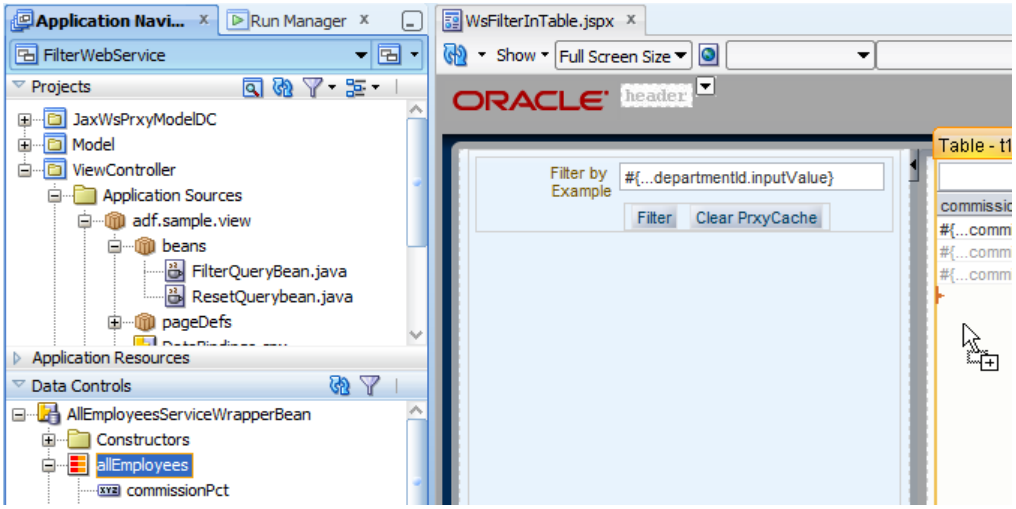
The methods, collections and attributes exposed in the `AllEmployeesServiceWrapperBean` bean now show in the Data Control panel from where they can be dragged into an ADF Faces page.

**Note**: To further customize a collection like **allEmployees**, for example to define UI hints to the attributes it exposes, select the collection in the Data Controls panel and use the right mouse button to display the **Edit Definition** menu option.

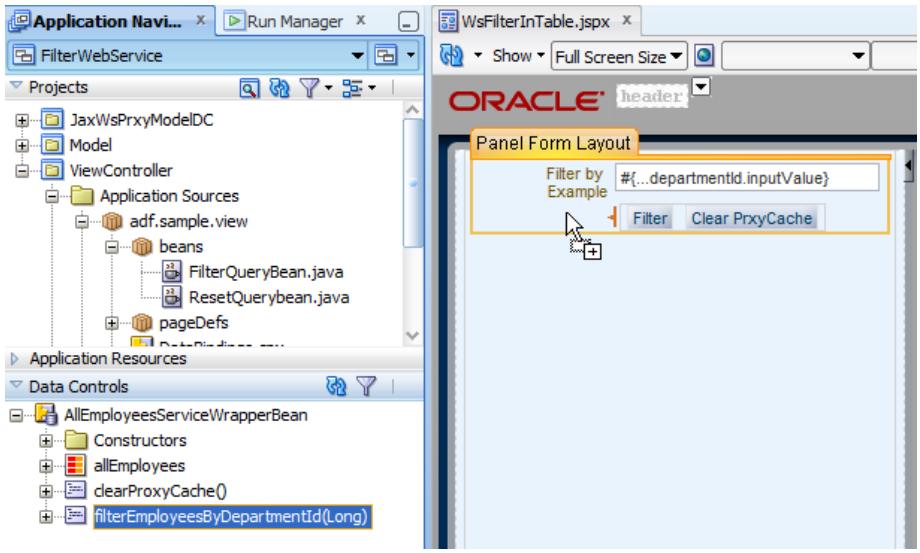



## Display WS data in an ADF Faces table and enable filtering

In the sample, to create a table from a collection, the **allEmployees** collection is dragged from the Data Controls panel and dropped onto the ADF Faces page. In the component context menu that opens, the read only table option was chosen.




In the table edit dialog, the **filter** option was selected to show search fields in the column headers. When users type a search string in, and hit **Enter** the table re-executes the query to apply the filter condition. Because the query is passed on to the wrapper bean and not directly to the Web Service, the filtered data is read from the in-memory cache.




Similar, to create a search field that also filters the data displayed in the table using in memory filtering implemented in the wrapper bean, the **filterEmployeesByDepartmentId** method was dragged from the Data Controls panel and dropped as a parameter form. The command button was renamed to "**Filter**".

To clear the in-memory cache so that the next query again queries data from the Web Service directly, the **clearProxyCache** method was dragged as a command button to the panel form. The **Refresh** condition of the iterators in the PageDef file was set to **ifNeeded** to refresh the data content when preparing the ADF data model.

The sample references the following managed bean codes in the command buttons action property. The managed beans ensure the ADF iterator is re-executed when the data was changed in the bean wrapper exposed by the Data Control

---------- Filter Query Bean ----------

```
import oracle.adf.model.BindingContext;
import oracle.adf.model.binding.DCIteratorBinding;
import oracle.adf.view.rich.component.rich.data.RichTable;
import oracle.binding.BindingContainer;
import oracle.binding.OperationBinding;

public class FilterQueryBean {
  public FilterQueryBean() {}
    public BindingContainer getBindings() {
        return BindingContext.getCurrent().getCurrentBindingsEntry();
    }
    public String cb1_action() {
     BindingContainer bindings = getBindings();
     OperationBinding operationBinding =
         bindings.getOperationBinding("filterEmployeesByDepartmentId");
     Object result = operationBinding.execute();
     if (!operationBinding.getErrors().isEmpty())
        return null;
```

```
      }
      //re-execute iterator to get data from the wrapper bean
     DCIteratorBinding dciter =
          (DCIteratorBinding) bindings.get("allEmployeesIterator1");
     dciter.executeQuery();
     return null;
   }
}
```

---------- Reset Query Bean ----------

```
import oracle.adf.model.BindingContext;
import oracle.adf.model.binding.DCIteratorBinding;
import oracle.binding.BindingContainer;
import oracle.binding.OperationBinding;

public class ResetQuerybean {
  public ResetQuerybean() {}

  public BindingContainer getBindings() {
    return BindingContext.getCurrent().getCurrentBindingsEntry();
  }

  public String cb2_action() {
    BindingContainer bindings = getBindings();
    OperationBinding operationBinding =
                     bindings.getOperationBinding("clearProxyCache");
    Object result = operationBinding.execute();
    if (!operationBinding.getErrors().isEmpty()) {
        return null;
    }
    DCIteratorBinding dciter =
           (DCIteratorBinding) bindings.get("allEmployeesIterator1");
    dciter.executeQuery();
      return null;
    }
}
```

## Sample download

You can download the Oracle JDeveloper 11g R1 workspace as sample 92 from the ADF Code Corner Website:

http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html#CodeCornerSamples

Configure the database access for this demo to point to the HR sample schema of an Oracle XE or enterprise database. Run the JSPX document and either filter the table data using the column header filters or the search field. To reset the query, press the **ClearPrxyCache** command button.

**RELATED DOCOMENTATION**

| | |
|---|---|
| ⊠ | |
| ⊠ | |
| ⊠ | |