REGIONE AUTONOMA DELLA SARDEGNA

SISaR

ENGINEERING

Servizio di Transizione

Titolo: XMPI/JBF/JAUF
Codice: AREAS_AP_01_JBF
Profilo: Analista Programmatore

www.eng.it

ALLEGATO 1: CODE SNIPPETS

DOCUMENT INDEX

1. Code snippets JBF

Designing the Model

1. Write down all the property of the POJO
2. Implement the interfaces **Persisent, ValidableInterface,** and **DynamicInfoInterface**
3. Declare the **memento** properties

```
protected transient static JDBCMementoDefs memDefs = new JDBCMementoDefs("TABLE_NAME
", new String[] { "PK" }, "UT_INS", "DT_INS", "UT_VAR", "DT_VAR");
protected transient JDBCMemento mem = new JDBCMemento(memDefs);
```

4. Implement the Store, Retrieve, Remove methods

```
@Override
  public void Store(PersistenceHandler ph) throws PersistenceException {
    mem.clear();

    mem.loadFromBean(this);
    ph.Store(mem);
  }

  @Override
  public void Retrieve(PersistenceHandler ph) throws PersistenceException {
    mem.clear();

    mem.loadKeyFromBean(this);
    ph.Retrieve(mem);
    mem.saveToBean(this);
  }

  @Override
  public void Remove(PersistenceHandler ph) throws PersistenceException {
    mem.loadKeyFromBean(this);
    ph.Remove(mem);
  }
```

5. Implement the **getObjectInfo** method and declare the **infoDelegate**

```
protected static DynamicInfoInterface infoDelegate;

@Override
  public Object getObjectInfo(String info, String propertyName) {
    if(infoDelegate == null) {
      infoDelegate = DynamicInfoUtils.getBeanDynamicInfo(MyBean.class);
    }

    return infoDelegate.getObjectInfo(info,propertyName);
  }
```

6. Write down the XML descriptor

```xml
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<Bean Description="{{SHORT_DESCRIPTION}}" Name="{{BEAN_NAME}}"
TableName="{{TABLE_NAME}}">
  <Keys>
    <Key Name="{{KEY_PROPERTY}}"/>
  </Keys>
  <Properties>
    <Property Name="{{PROPERTY_NAME}}" Field="{{DB_FIELD}}" Caption="{{CAPTION}}"
Classe="{{JAVA_CLASS}}" AutoMapping="true" IsDetail="false">
      <Info Length="{{LENGTH}}" Mandatory="0"/>
    </Property>
  </Properties>
</Bean>
```

Designing the Query

1. Extends the abstract class **CustomQueryProfile**
2. Declare filters, fields and captions inside the constructor

```
public MyQuery() {
    formalFilters = "filter1,filter2";
    fieldsName = "field1,field2";
    fieldsCaption = "caption1,~caption_hidden"; //use tilde to hide a column
  }
```

3. Implement the method **getCustomSqlString**

```
@Override
  protected String getCustomSqlString() throws QueryException {
    String sql = "SELECT "+getFieldsName() +" FROM X WHERE 1 = 1 ";

    // Check if the user set this filter
    if(isFilterPresent("filter1")) {
      // Take the filter value
      Object filterValue = getFilterValue("filter1");
      // Build the SQL
      sql += " AND FIELD = " + JDBCDataMapper.objectToSQL(filterValue);
    }

    return sql;
  }
```

Designing the Find View

1. Start from this template

```
<%@ page language="java"%>
<%@ taglib uri="/WEB-INF/PSGTagLibrary.tld" prefix="psg"%>
<%@ taglib uri="/WEB-INF/PSGTagLibraryForms.tld" prefix="forms"%>

<HTML>
<HEAD>
<TITLE>{{WINDOW_TITLE}}</TITLE>
<LINK REL="stylesheet" HREF="PSGLibrary/psgcommon.css" TYPE="text/css">
</HEAD>

<%@ include file="/PSGLibrary/topInfo.jsp"%>
<psg:controller />

<BODY bgcolor="white" onload="doOnLoad();">
      <FORM action="<psg:controller options="formAction"/>" method="post">
            <psg:controller options="defaultFormFields" />
            <psg:error propertyName="GLOBAL" />
            <forms:tabber>
                  <FIELDSET>
                        <LEGEND class="fieldset_text">{{PAGE_TITLE}}</LEGEND>

                        <table width="100%"  border="0" cellspacing="2"
cellpadding="2">
                        <tr><td>
                        <TABLE cellpadding="2" align="center" cellspacing="0"
border="1" width="95%" class="table_ricerca" bordercolordark="<%=bordercolordark%>"
bordercolorlight="<%=bordercolorlight%>">
                              <!-- START: You can edit from here -->

                              <TR class="<forms:onOff offString="row0"
onString="row1"/>">
                                    <TD>
                                          <!--Label-->
                                    </TD>
                                    <TD colspan="3">
                                          <!--Field-->
                                    </TD>
                              </tr>

                              <!-- END: You can edit from here -->
                        </TABLE>
                        </td></tr>
                        </table>

                  </FIELDSET>
                  <%@ include file="/PSGLibrary/w3StatusButtons.jsp" %>
                  <%@ include file="/PSGLibrary/findResult.jsp"%>
            </forms:tabber>
      </FORM>

      <script language="JavaScript" src="PSGLibrary/psgcommon.js"></script>
      <psg:controller options="scripts" />
</BODY>
</HTML>
```

2. Model your page and form fields. Use the **linkPropertyTag** for bindings

```
<psg:linkProperty propertyName="filterName" isFilter="Y">
      <!-- HTML Widget -->
      <input type="text" />
      <psg:error />
</psg:linkProperty>
```

Designing the Insert/Edit View

1. Start from this template

```
<%@ page language="java"%>
<%@ taglib uri="/WEB-INF/PSGTagLibrary.tld" prefix="psg"%>
<%@ taglib uri="/WEB-INF/PSGTagLibraryForms.tld" prefix="forms"%>

<HTML>
<HEAD>
<TITLE>{{WINDOW_TITLE}}</TITLE>
<LINK REL="stylesheet" HREF="PSGLibrary/psgcommon.css" TYPE="text/css">
</HEAD>

<%@ include file="/PSGLibrary/topInfo.jsp"%>
<psg:controller />

<BODY bgcolor="white" onload="doOnLoad();">
       <FORM action="<psg:controller options="formAction"/>" method="post">
              <psg:controller options="defaultFormFields" />
              <psg:error propertyName="GLOBAL" />

              <forms:tabbedPanel width="100%">
                     <table width="100%"  border="0" cellspacing="2" cellpadding="2">
                            <tr><td>
                            <TABLE cellpadding="2" align="center" cellspacing="0" border="1"
width="95%" class="table_ricerca" bordercolordark="<%=bordercolordark%>"
bordercolorlight="<%=bordercolorlight%>">
                                   <!-- START: You can edit from here -->

                                   <TR class="<forms:onOff offString="row0"
onString="row1"/>">
                                          <TD>
                                                 <!--Label-->
                                          </TD>
                                          <TD colspan="3">
                                                 <!--Field-->
                                          </TD>
                                   </tr>

                                   <!-- END: You can edit from here -->
                            </TABLE>
                            </td></tr>
                            </table>
                     </forms:tabbedPanel>
                     <%@ include file="/PSGLibrary/w3StatusButtons.jsp" %>
       </FORM>

       <script language="JavaScript" src="PSGLibrary/psgcommon.js"></script>
       <psg:controller options="scripts" />
</BODY>
</HTML>
```

3. Model your page and form fields. Use the **linkPropertyTag** for bindings

```
<psg:linkProperty propertyName="beanProperty" useBeanInfo="Y">
      <!-- HTML Widget -->
      <input type="text" />
      <psg:error />
</psg:linkProperty>
```

Designing the Controller

1. Extends the class **W3ControllerPersistent**
2. Implement the **init** method of the class

```
@Override
  public void init(HttpServletRequest req) throws Exception {
    super.init(req);

    // Init the Query class
    ServiceResultBuilder serviceResultBuilder = new
ServiceResultBuilder(MyQuery.class.getName(), "pk_property_name", user);
    serviceResultBuilder.setActionPath(getControllerName() + ".do?ACTION=EDIT");
    serviceResultBuilder.setPaginable(true);
    setFindResultBuilder(serviceResultBuilder);

    // Init the Views
    setReturnPage(STATUS_FIND, "/my/jspFind.jsp");
    setReturnPage(STATUS_EDIT, "/my/jspEdit.jsp");
    setReturnPage(STATUS_INSERT, STATUS_EDIT);
    setReturnPage(STATUS_VIEW, STATUS_EDIT);

    // Init the first view to show
    setStatus(STATUS_FIND);
    setLastPageForward(getReturnPage(getStatus()));

    // Controller settings
    useService = true;
    enableProfiler(MyBean.class.getClass());
  }
```

3. Declare the mapping in the **controller.properties** of your module

```
mapping.my.super.mapping = package.of.my.ControllerW3
bean.package.of.my.ControllerW3 = package.of.my.Bean
```

Creating a ComboBox

1. Create inside the controller a method that returns the list of available options

```
public List getComboOptionsList() {
    //key/description comma separeted
    String options = "key1,Description1,key2,description2";
    return PSGLibrary.util.ListUtils.createCodeValueList(options);
  }
```

2. Add the combo box inside your JSP

```
<psg:linkProperty propertyName="status" optionsSourceProperty="comboOptionsList"
optionsSourceShowKey="N" optionBlank="N" >
        <select> </select>
        <psg:error/>
</psg:linkProperty>
```

- **optionsSourceShowKey**, indicates if you want to show inside the combo box also the keys of the option. Possible values:
    - S, Yes
    - N, No
- **optionBlank**, indicates if you want to always add the “empty” option inside your combo. Possible values:
    - S, Yes
    - N, No

Creating a CheckBox

1. Declare your property as a String. It will be a flag (S,N)
2. Add the checkbox in your JSP

```
<psg:linkProperty propertyName="remote" useBeanInfo="Y">
      <INPUT TYPE="checkbox" value="S">
      <psg:error />
 </psg:linkProperty>
```

Set the visibility of fields

1. Implement the interface InteractiveInterface
2. Implement the method **getPropertyInfo** with the required logics

```
@Override
  public long getPropertyInfo(String propertyName, int info) {
    if(InteractiveInterface.INFO_STATUS == info){
      if("beanProperty".equals(propertyName)) {
        return InteractiveInterface.VALUE_REQUIRED;
       // InteractiveInterface.VALUE_LOCKED
      }
    }

Object result = getObjectInfo(DynamicInfoInterface.PROPERTY_INTERACTIVE
+"."+info,propertyName);
    if (result instanceof Number) {
      return ((Number)result).longValue();
    } else {
      return 0;
    }
  }
```

Use a sequence

1. Create the Oracle sequence

```
CREATE SEQUENCE SEQ_MY_SEQUENCE
```

2. Declare a new property (type Long) in your bean
3. Modify your Store method to use the sequence and to assign its value to the property

```
@Override
  public void Store(PersistenceHandler ph) throws PersistenceException {
    mem.clear();

    mem.loadFromBean(this);
    if (id == null) {
      mem.setProperty("PK_FIELD", new OracleSequence("SEQ_MY_SEQUENCE"));
    }

    ph.Store(mem);
    id = mem.getPropertyAsLong("PK_FIELD");
  }
```

Create a Detail

1. Model the new bean as an usual one (including the XML)
2. Add in the detail bean a reference to the parent

```
protected String idParent;
protected transient MyParent parent;

public MyDetail(Object parent) {
  this.parent = (MyParent) parent;
  this.idParent = this.parent.getId();
}
```

3. Modify the parent bean in order to manage the list of details

```
private List<MyDetail> details = new ArrayList<>();

@Override
  public void Retrieve(PersistenceHandler ph) throws PersistenceException {
    mem.clear();

    mem.loadKeyFromBean(this);
    ph.Retrieve(mem);
    mem.saveToBean(this);

    ph.getPersistenceManager().Find(MyDetail.class.getName(), new Class[]
{Object.class}, new Object[]{this}, details,
mem.getReferenceForeignKey("PARENT_FIELD"));
  }

public List<MyDetail> getDetails() {
    return details;
  }

  public void setDetails(List<MyDetail> details) {
    this. details = details;
  }
```

4. Create a JSP to handle the details

```
<%@ page language="java"%>
<%@ taglib uri="/WEB-INF/PSGTagLibrary.tld" prefix="psg"%>
<%@ taglib uri="/WEB-INF/PSGTagLibraryForms.tld" prefix="forms"%>

<html>
<head>
<TITLE>{{WINDOW_TITLE}}</TITLE>
<LINK REL="stylesheet" HREF="PSGLibrary/psgcommon.css" TYPE="text/css">
</head>

<%@ include file="/PSGLibrary/topInfo.jsp"%>

<psg:controller />
<body bgcolor="white" onLoad="doOnLoad();">
  <form action="<psg:controller options="formAction"/>" method="post">
        <psg:controller options="defaultFormFields" />

        <psg:error propertyName="GLOBAL" />

        <table width="95%" border="0" cellpadding="2" cellspacing="2">
```

```
                <tr>
                      <td CLASS="label_l" width="19%">{{PARENT_NAME}}</td>
                      <td CLASS="label">
                            <psg:linkProperty propertyName="{{property_parent}}"
  source="parentObject" />
                      </td>
                </tr>
          </TABLE>

          <forms:tabbedPanel>

                <table width="95%" align="center">
                      <tr>
                            <td>
                                  <table width="100%">
                                        <tr>
                                              <td class="detailBox">
                                                    <div
class="genericResult200">
                                                          <TABLE width="100%">
                                                                <thead>
                                                                      <TR>
                                                                            <!-
- @EDIT START: List of columns -->

                                                                            <td
class="results">{{COLUMN1}}</td>
                                                                            <td
class="results">{{COLUMN2}}</td>
                                                                            <!-
- END: List of columns -->

                                                                            <td
class="results">&nbsp;</td>
                                                                      </tr>
                                                                </thead>

                                                                <tbody
class="detail_results">

   <psg:iterate propertyName="{{detail_property}}"    source="parentObject"
itemTarget="item">

                                                                            <TR
   class="<forms:onOff offString="result_cell0" onString="result_cell1"/>">


   <!-- @EDIT START: List of values -->

   <td>

         <psg:linkProperty propertyName="{{bean_property_column1}}" source="item"
/>

   </td>

   <td>

         <psg:linkProperty propertyName="{{bean_property_column2}}" source="item"
/>
```

```
   </td>

   <!-- END: List of values -->


   <td align="center">

         <BUTTON type="button" class="BTN" value=">>"

               ONCLICK="doActionAndIndex('SELECT','<psg:linkProperty
propertyName="object" source="item.index"/>')" STYLE="height: 18">

               <IMG SRC='images/freccine_dx.gif' />

         </BUTTON>

   </td>

   </tr>

   </psg:iterate>
                                                                  </tbody>
                                                            </TABLE>
                                                      </div>
                                                </td>
                                                <td class="buttonBox">
                                                <%@ include
   file="/PSGLibrary/w3EditDetailButtonsVert.jsp"%>
                                                </td>
                                          </tr>
                                    </table>
                              </td>
                        </tr>
                  </table>

                  <table width="100%" align="center" border="0">
                        <tr>
                              <td>
                                    <TABLE cellpadding="2" cellspacing="0"
align="center" border="1" width="95%" class="table_ricerca"
bordercolordark="<%=bordercolordark%>" bordercolorlight="<%=bordercolorlight%>">

                                          <!-- @EDIT START: You can edit from here -
->
                                          <TR class="<forms:onOff offString="row0"
onString="row1"/>">
                                                <TD>{{label}}</TD>
                                                <TD colspan="3">{{form_field}}</TD>
                                          </tr>
                                          <!-- END: You can edit from here -->
                              </TD>
                        </TR>

                  </table>
         </forms:tabbedPanel>
   </FORM>
   <script language="JavaScript" src="PSGLibrary/psgcommon.js"></script>
   <psg:controller options="scripts" />
</BODY>
</html>
```

5. Create a new Controller that extends **W3ControllerPersistentDetail** and implement the **init** method as following

```
@Override
  public void init(HttpServletRequest req) throws Exception {
    super.init(req);

    // Link to parent controller
    linkToParent();
    linkToParentDetail("details"); // The name of the property in the parent Bean
    linkTabsFromController(getParentController());

    // Init the Views
    setReturnPage(ControllerStatusInterface.STATUS_EDIT, "/my/jspDetail.jsp");
    setReturnPage(ControllerStatusInterface.STATUS_INSERT, STATUS_EDIT);
    setReturnPage(ControllerStatusInterface.STATUS_VIEW, STATUS_EDIT);
    setLastPageForward(getReturnPage(getStatus()));
    setStatus(STATUS_INSERT);

    // Controller settings
    useService = true;
    enableProfiler(MyDetail.class.getName());
  }
```

6. Declare the mapping of the new Controller

7. Modify the parent controller to show the details

```
@Override
  public void init(HttpServletRequest req) throws Exception {
    super.init(req);

    // Init the Query class
    ServiceResultBuilder serviceResultBuilder = new
ServiceResultBuilder(MyQuery.class.getName(), "code", user);
    serviceResultBuilder.setActionPath(getControllerName() + ".do?ACTION=EDIT");
    serviceResultBuilder.setPaginable(true);
    setFindResultBuilder(serviceResultBuilder);

    // Init the Views
    setReturnPage(STATUS_FIND, "/my/jspFind.jsp");
    setReturnPage(STATUS_EDIT, "/my/jspEdit.jsp");
    setReturnPage(STATUS_INSERT, STATUS_EDIT);
    setReturnPage(STATUS_VIEW, STATUS_EDIT);

    // Init the first view to show
    setStatus(STATUS_FIND);
    setLastPageForward(getReturnPage(getStatus()));

    useService = true;
    enableProfiler(MyBean.class.getClass());

    //Init the tabs
    Tab tab = getTabsList().addTab("TAB_MAIN", "My Parent");
    tab.setAction(ACTION_TAB);

    tab = getTabsList().addTab("TAB_DETAL1", "Detail 1");
    tab.setAction(ACTION_TAB);
    tab.setExecutor(new ButtonExecutorInterface(){
      public Object execute(ControllerInterface controller, ButtonInterface button,
HttpServletRequest req, String action) throws ButtonExecutorException {
        return new ControllerForward(req,"/mapping.of.my.detail.do?ACTION=INSERT",
CourseW3.this);
      }
    });

    getTabsList().setActive("TAB_MAIN");
  }
```

Creating a Calendar

3. Add the field in your JSP

```
<psg:linkProperty propertyName="date" format="dd/MM/yyyy" calendar="Y"
useBeanInfo="Y">
      <INPUT TYPE="text">
      <psg:error />
</psg:linkProperty>
```

- **format**, It indicates the date format to use (Italian standard is dd/MM/yyyy)
- **calendar**, It indicates if you want to use a date picker
    - Y, Yes
    - N, No

Create a Detail in Grid mode

1. Follow the instruction to create a normal detail
2. Create a JSP to design your grid

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/PSGTagLibrary.tld" prefix="psg" %>
<%@ taglib uri="/WEB-INF/PSGTagLibraryForms.tld" prefix="forms" %>

<HTML>
  <HEAD>
    <TITLE>{{WINDOW_TILE}}</TITLE>
    <LINK REL="stylesheet" HREF="PSGLibrary/psgcommon.css" TYPE="text/css">
  </HEAD>

  <%@ include file="/PSGLibrary/topInfo.jsp" %>
  <psg:controller/>

  <BODY bgcolor="white" onLoad="doOnLoad()">
    <BR>

    <form action="<psg:controller options="formAction"/>" method="post" >
      <psg:controller options="defaultFormFields"/>
      <psg:error propertyName="GLOBAL" />

      <TABLE  width="100%" border="0">
      <TR><TD>
        <TABLE cellpadding="2" align="center" cellspacing="0" border="1" width="100%"
class="table_ricerca" bordercolordark="<%=bordercolordark%>"
bordercolorlight="<%=bordercolorlight%>">
        </TABLE>
      </TD></TR>
      </TABLE>
      <forms:tabbedPanel propertyName="tabsList" width="100%">
      <TABLE width="100%">
      <TR><TD>
        <TABLE width="95%" align="center">
          <TR><TD>
            <TABLE width="100%" border="1">
              <TR>
                <TD class="detailBox">
                  <div class="genericResult400" id="resultData">
                    <TABLE cellpadding="0" width="100%" cellspacing="0" border="1"
bordercolordark="<%=bordercolordarkFindResult%>"
bordercolorlight="<%=bordercolorlightFindResult%>">
                      <thead>
                      <TR>
                        <TD class="results" width="1%">
                          <input type="checkBox" name="ALLROWS"
onClick="selDesel('ROW',this)"><psg:if propertyName="errors" source="controller"
operator="GT" propertyValue="0"><IMG src='images/error.gif'/></psg:if>
                        </TD>
                        <TD class="results" width="18%">{{COLUMN1}}</TD>
                        <TD class="results" width="30%">{{COLUMN2}}</TD>
                        <TD class="results" width="1%"></TD>
                      </TR>
                      </thead>
                      <tbody class="detail_results">
```

```
                        <psg:iterate propertyName="detailList" source="controller"
itemTarget="item" autoIndexPropertyName="Y">

                          <psg:callback methodName="callBackDetailStatus"/>
                          <TR class="<forms:onOff name="detail" offString="result_cell0"
onString="result_cell1"/>">
                            <TD width="1%" align="center">
                              <psg:decorate renderer="DetailItemStatus"
source="item.itemStatus" propertyName="object" >
                                (<psg:linkProperty propertyName="object"
source="item.index"/>,checkBox)
                              </psg:decorate >
                            </TD>
                    <TD colspan="1">
                      <p class="par_down">
                        <psg:linkProperty propertyName="{{property1}}" useBeanInfo="Y"
source="item">
                          <input type="text">
                        </psg:linkProperty>
                      </p>
                    </TD>
                    <td>
                    <psg:linkProperty propertyName="{{property2}}" useBeanInfo="Y"
source="item">
                          <input type="text">
                        </psg:linkProperty>
                        </td>
                  </TR>
                </psg:iterate>
                </tbody>
              </TABLE>
            </div>
            <div align="center">
              <psg:linkProperty propertyName="detailList.htmlPanel" source="controller"
escape="NO"/>
            </div>
          </TD>
        </TR>
      </TABLE>
    </TD></TR>
</TABLE>
        </TD></TR>
        </TABLE>

          <%@ include file="/PSGLibrary/w3StatusButtons.jsp" %>
        </forms:tabbedPanel>
      </form>
      <script language="JavaScript" src="PSGLibrary/psgcommon.js"></script>
      <psg:controller options="scripts"/>
    </BODY>
</HTML>
```

3. Edit your detail controller to set the grid mode

```
@Override
  public void init(HttpServletRequest req) throws Exception {
    super.init(req);
    useService = true;

    // Lin to parent controller
    linkToParent();
    linkToParentDetail("details");
    linkTabsFromController(getParentController());

    // Init the Views
    setReturnPage(ControllerStatusInterface.STATUS_GRID, "/course/jspGrid.jsp");
    setReturnPage(ControllerStatusInterface.STATUS_INSERT, STATUS_GRID);
    setReturnPage(ControllerStatusInterface.STATUS_VIEW, STATUS_ GRID);
    setStatus(STATUS_GRID);
    setLastPageForward(getReturnPage(getStatus()));

    enableProfiler(MyDetail.class.getName());
  }
```

Define a new Lookup

1. Insert the new lookup inside the table **SI_LOOKUP**

```
INSERT
INTO SI_LOOKUP
  (
    LK_CODICE,
    LK_TITLE,
    LK_TABLENAME,
    LK_KEYFIELDS,
    LK_LOOKUPFIELDS,
    LK_LISTFIELDS,
    LK_CAPTIONFIELDS,
    LK_UT_INS,
    LK_DT_INS
  )
  VALUES
  (
    'LOOKUP_NAME',
    'List of something',
    'TABLE_NAME',
    'PK',
    'FIELD1,FIELD2',
    'PK,FIELD1,FIELD2',
    '~PkCaption,Caption1,Caption2',
    'BTU',
    SYSDATE
  );
```

2. Add the property in your model
3. Modify the **getObjectInfo** to indicate the lookup to use for your new property

```
@Override
  public Object getObjectInfo(String info, String propertyName) {
    if(DynamicInfoInterface.PROPERTY_LOOKUP.equals(info)) {
      if("property_name".equals(propertyName)) {
        return "LOOKUP_NAME";
      }
    }

    if (infoDelegate == null) {
      infoDelegate = DynamicInfoUtils.getBeanDynamicInfo(Exam.class);
    }

    return infoDelegate.getObjectInfo(info, propertyName);
  }
```

4. Use the **LookupPropertyTag** on your JSP to show the lookup widget

```
<psg:linkProperty propertyName="property_name" useBeanInfo="Y">
      <input type="hidden">
</psg:linkProperty>

<psg:lookupProperty propertyNames="property_name" lookupCodeField="CODE_FIELD1"
useBeanInfo="Y">
      <input type="text" name="property_name_field1">
</psg:lookupProperty>

<psg:lookupProperty propertyNames="property_name" lookupField="FIELD2"
useBeanInfo="Y">
      <input type="text" name="property_name_field2">
</psg:lookupProperty>

<psg:lookupProperty propertyNames="property_name" >
      <input type="button" class="btn" value="...">
</psg:lookupProperty>
```

Add filters to a Lookup

1. Modify the **getObjectInfo** as following

```
@Override
  public Object getObjectInfo(String info, String propertyName) {
    if(DynamicInfoInterface.PROPERTY_LOOKUP_FILTERS.equals(info)) {
      if("property_name".equals(propertyName)) {
        Map<String, Object> filters = new HashMap<>();
        filters.put(filterName, filterValue);

        return filters;
      }
    }

    if (infoDelegate == null) {
      infoDelegate = DynamicInfoUtils.getBeanDynamicInfo(Exam.class);
    }

    return infoDelegate.getObjectInfo(info, propertyName);
  }
```

Define a FIND2

1. Follow all the instruction to define a lookup
2. Modify your JSP, in the section relative to the button

```
<psg:lookupProperty propertyNames="property_name"
linkedController="controller.mapping">
      <input type="button" class="btn" value="...">
</psg:lookupProperty>
```

Add filters to a FIND2

1. Modify the **init** method of your controller to include this section

```
Find2ActionHandler find2Handler = (Find2ActionHandler)
actionBinder.getActionHandlers().get(ACTION_FIND2);

find2Handler.setFilterDecorator(new ControllerForwardDecoratorInterface() {
      public void beforeControllerForward(String controllerForward,
ControllerContext context, HttpServletRequest req, String options, Map
originalReqParams) {
        if("mapping.of.destination.controller ".equals(controllerForward)) {
          Map filters = new HashMap();
          filters.put("filtername", filterValue);

          context.set(CONTEXT_FIND_FILTERS, filters);
        }
      }
    });
```

Define a FIND3

1. Modify your JSP to include the widget

```
<forms:find3 propertyName="items" source="controller" lookupName="LOOKUP_NAME"
controllerForward="/mapping.of.controller.do" lookupField="FIELD_TO_SHOW">
      <SELECT MULTIPLE size="10" STYLE="width:200px">
      </SELECT>
</forms:find3>
```

2. Modify the controller to include the property that will receive the list selected in the find3

```
private List<Object> items = new ArrayList<>();

  public List<Object> getItems() {
    return items;
  }

  public void setItems(List<Object> items) {
    this.items = items;
       //getFindFilter().put("filtro",ListUtils.toCommaSeparatedString(items));
  }
```

Implement a new Action

1. Implement the method doAction in your controller in order to manage your action

```
@Override
  protected Object doAction(String action, HttpServletRequest req) throws Exception
{
    if("MY_ACTION".equals(action)) {
      return doSomething(req);
    }

    return super.doAction(action, req);
  }

  protected Object doSomething(HttpServletRequest req) throws Exception {
     setDataProperties(req, null); //to set into your properties the data in
your request

    // I create a service context
    ServiceContext serviceContext = new ServiceContext(user);

    Connection connection = null;
    try {
      // I take a DB connection
      connection = ConnectionManager.getConnection(user, 1000);
      serviceContext.setConnection(connection);

      // I create a new Object
      BinderInterface newBinder = ServiceManager.getObject(MyBean.class.getName(),
null, serviceContext);

      // I set the properties into the binder
      newBinder.setPropertyValue("property1", "value", serviceContext);
      newBinder.setPropertyValue("property2", "value", serviceContext);

      // I save the new binder
      ServiceManager.save(newBinder, serviceContext);

      // I read the properties from the binder
      Object id = newBinder.getPropertyValue("id");

      // I read a binder by ID
      BinderInterface binder = ServiceManager.getObject(MyBean.class.getName(), id,
serviceContext);

      connection.commit();
    } catch(Exception e) {
      addError("GLOBAL", "Something was wrong!");
    } finally {
      /* I always release the DB connection, no matter the result.
       * The release also do the rollback of the connection.
       * NEVER close the connection!
       */

      ConnectionManager.releaseConnection(connection);
    }
```

2. Add in your JSP the fire of the new Action

```
<input type="button" class="btn" onclick="doAction(‘MY_ACTION’)">
```

JSP Tag IF

```
<psg:if propertyName="property" operator="=" propertyValue="S" source="controller">

…<your html>…

</psg:if>
```

The value of the source option (that is also available in the **LinkPropertyTag**) can be:

- **mainObject** (default value if no source is specified). The value of the propertyName is taken from the bean
- **controller**. The value of the propertyName is taken from the controller
- **findFilter** (only in find mode). The value of the propertyName is taken from the filters passed to the query class

2. Code snippets JAUF

Designing the Filter View

1. Implement the interface **FindFilterBuilder**
2. Implement the method **getFindService** to specify your Query class

```
@Override
  public FindService getFindService() {
    return new FindService(YouQuery.class.getName(), "PK_FIELD") {
      @Override
      public Map getBeanKeyMapByRow(List record) {
        Map map = new HashMap();
        map.put("pk_property_name", record.get(0));

        return map;
      }
    };
  }
```

3. Model your page and form fields by implementing the **getFindComponent** method. Use the **addBinding** method for bindings with the underneath **MAP**.

```
@Override
  public ArrangeablePanel getFindComponent(Bindings bindings) {
    ArrangeablePanel panel = new ArrangeablePanel();

    // Create and set up the field
    FormTextField yourField = new FormTextField();
    yourField.setLabel("Label");
    // Add the field to your main panel
    panel.add(yourField, 12);
    // Bind the field with the underneath model
    bindings.addBinding(new SimpleMapBinding<>(yourField, "filter_name"));

    return panel;
  }
```

Designing the Insert View

2. Implement the interface **InsertBuilder**
3. Implement the method **getBinderService** to specify your bean class

```
@Override
  public BinderService getBinderService() {
    return new BinderService(YourBean.class.getName());
  }
```

4. Give a name to the Insert section

```
  @Override
  public String getInsertTitle() {
    return "Insert a new thing";
  }
```

5. Model your page and form fields by implementing the **getInsertComponent** method. Use the **addBinding** method for bindings with the underneath **Binder**.

```
@Override
  public ArrangeablePanel getInsertComponent(Bindings bindings) {
    ArrangeablePanel panel = new ArrangeablePanel();

    // Create and set up the field
    FormTextField nameField = new FormTextField();
    nameField.setLabel("Label");
    // Add the field to your main panel
    panel.add(nameField, 12);
    // Bind the field with the underneath model
    bindings.addBinding(new SimpleBinderBinding<>(nameField, "bean_property_name"));

    return panel;
  }
```

Designing the Edit View

1. Implement the interface **EditBuilder**
2. Implement the method **getBinderService** to specify your bean class

```
@Override
  public BinderService getBinderService() {
    return new BinderService(YourBean.class.getName());
  }
```

6. Give a title to all the Edit section. You can customize it using the **binder** object.

```
 // Overall title
  @Override
  public String getTabTitle(BinderInterface binder) {
    try {
      return "Modify thing: " + binder.getPropertyValue("bean_property_name ");
    } catch (JBFException e) {
      throw new RuntimeException(e);
    }
  }

  // Section title
  @Override
  public String getTitle(BinderInterface binder) {
    try {
      return "Thing: " + binder.getPropertyValue("bean_property_name ");
    } catch (JBFException e) {
      throw new RuntimeException(e);
    }
```

7. Model your page and form fields by implementing the **getEditComponent** method.

```
@Override
  public ArrangeablePanel getEditComponent(Bindings bindings) {
    ArrangeablePanel panel = new ArrangeablePanel();

    // Create and set up the field
    FormTextField nameField = new FormTextField();
    nameField.setLabel("Label");
    // Add the field to your main panel
    panel.add(nameField, 12);
    // Bind the field with the underneath model
    bindings.addBinding(new SimpleBinderBinding<>(nameField, "bean_property_name"));

    return panel;
  }
```

Designing the Controller

1. Extends the class **W3ControllerPersistentJauf**
2. Implement the **init** method of the class

```
@Override
  public void init(HttpServletRequest req) throws Exception {
    super.init(req);

    setTitle("Animal registry");
  }
```

3. Annotate the controller

```
@UseLayout(StandardLayoutBuilder.class)
@RequireModules({
  // Your components
  AnimalFilterBuilder.class,
  AnimalInsertBuilder.class,
  AnimalEditBuilder.class,

  /* Standard components */
  StandardResultBuilder.class,
  // Buttons
  StandardFindButtonBarBuilder.class,
  StandardFindResultButtonBarBuilder.class,
  StandardEditButtonBarBuilder.class,
  StandardInsertButtonBarBuilder.class,
  // Persistence
  StandardDetailBinderLoader.class,
  StandardBinderLoader.class,
  StandardBinderSaver.class,
  StandardBinderEraser.class,
})
public class YourControllerW3 extends W3ControllerPersistentJauf {
```

4. Declare the mapping in the **controller.properties** of your module

```
mapping.my.super.mapping = package.of.my.ControllerW3
bean.package.of.my.ControllerW3 = package.of.my.Bean
```

Creating a ComboBox

```
FormSelectField kindField = new FormSelectField();
kindField.setLabel("Kind");
// Optional, if you want to have an empty option
kindField.addEmptyKeyValueOption();
// List all your options
kindField.add(new KeyValueOption("KEY1", "Description 1"));
kindField.add(new KeyValueOption("KEY2", "Description 2"));
bindings.addBinding(new SelectKeyValueOptionsXYZBinding(kindField, "kindFilter"));
```

Instead of XYZ, specify the binding based on your model (Map, Binder). KEY1 and KEY2 are the values that are going to be set and stored into the model.

Creating a CheckBox

1. Declare your property as a String. It will be a flag (Y,N)
2. Add your checkbox

```
FormCheckField checkField = new FormCheckField();
checkField.setLabel("Label");
bindings.addBinding(new CheckboxXYZBinding(checkField, "property_name", "Y", "N"));
```

Create a Detail

1. Model the new bean as an usual one (including the XML)
2. Add in the detail bean a reference to the parent (idParent is your parent property)

```
protected Object idParent;
protected transient MyParent parent;

public MyDetail(Object parent) {
  this.parent = (MyParent) parent;
  this.idParent = this.parent.getId();
}
```

3. Modify the parent bean in order to manage the list of details

```
private List<MyDetail> details = new ArrayList<>();

@Override
  public void Retrieve(PersistenceHandler ph) throws PersistenceException {
    mem.clear();

    mem.loadKeyFromBean(this);
    ph.Retrieve(mem);
    mem.saveToBean(this);

    ph.getPersistenceManager().Find(MyDetail.class.getName(), new Class[] {Object.class}, new Object[]{this}, details,
mem.getReferenceForeignKey("DETAIL_FIELD_FK"));
  }

public void Remove(PersistenceHandler ph) throws PersistenceException {
    //     Remove the children
    ph.getPersistenceManager().RemoveAll(details);

    mem.loadKeyFromBean(this);
    ph.Remove(mem);
  }

public List<MyDetail> getDetails() {
    return details;
  }

  public void setDetails(List<MyDetail> details) {
    this. details = details;
  }
```

4. Create a new class that implements the interface **DetailEditBuilder**
5. Implement the method **getBinderService**

```
@Override
  public BinderService getBinderService() {
    return new BinderService(YourDetailBean.class.getName(),
"property_name_of_the_list_in_parent_bean");
  }
```

6. Implement the **getTitle** method

```
@Override
  public String getTitle(BinderInterface parentBinder) {
    return "My details";
  }
```

7. Create a constructor that receive a **W3ControllerPersistentJAUF** as parameter

```
public MyDetailEditBuilder(W3ControllerPersistentJauf controller) {
    this.controller = controller;
  }
```

8. Implement the **getGridDataModel** method

```
@Override
  public GridDataModel getGridDataModel(Bindings parentBindings, List<Persistent>
detailBinder) {
    ServiceContext context = new ServiceContext(controller.getUser());
    context.setParameterValue(ServiceContext.PARENT_OBJECT,
parentBindings.getModel());

    // We create the detail model
    GridDataModel model = new
ListBinderModelData(getBinderService().getMainBinderRef(), context, detailBinder);
    // We allow the user to edit the model
    model.setEditMode(true);

    return model;
  }
```

9. Draw the grid that will manage your detail, by implementing the **getEditComponent**

```
@Override
  public ArrangeablePanel getEditComponent(GridDataModel gridDataModel, Bindings
parentBindings, List<Persistent> detailBinder) {
    final ArrangeablePanel panel = new ArrangeablePanel();

    // Create a grid and add it to your main panel
    final Grid grid = new Grid("Title of your grid", gridDataModel);
    panel.add(grid, 24);

    // Define the list of the columns you want in your grid
    List<Column> columns = new ArrayList<Column>();
    columns.add(new Column().type(new DeleteButtonColumnType()).width(30));

    columns.add(new Column().caption("Caption of your
column").name("property_name_in_your_bean").type(new
StringColumnType()).factory(SimpleBinderBinding.getFactory(true)).width(100));

    grid.setColumns(columns);
    grid.setPageSize(15);

    ServiceContext context = new ServiceContext(controller.getUser());
    grid.setBindingsProvider(new ListBinderBindingsProvider(gridDataModel,
context));

    return panel;
  }
```

method

10. Go back to your controller and list your **DetailEditBuilder** implementation among the other components

```
@UseLayout(StandardLayoutBuilder.class)
@RequireModules({
  // Your components
  AnimalFilterBuilder.class,
  AnimalInsertBuilder.class,
  AnimalEditBuilder.class,
  VaccinationDetailBuilder.class,

  /* Standard components */
  StandardResultBuilder.class,
  // Buttons
………etc…
```

Creating a Calendar

```
FormDateField bornField = new FormDateField(DateFormat.ITALIAN_DATE);
bornField.setLabel("Born date");
panel.add(bornField, 6);
bindings.addBinding(new SimpleXYZBinding<>(bornField, "bornDate"));
```

Define a new Lookup

5. Insert the new lookup inside the table **SI_LOOKUP**

```
INSERT
INTO SI_LOOKUP
  (
    LK_CODICE,
    LK_TITLE,
    LK_TABLENAME,
    LK_KEYFIELDS,
    LK_LOOKUPFIELDS,
    LK_LISTFIELDS,
    LK_CAPTIONFIELDS,
    LK_UT_INS,
    LK_DT_INS
  )
  VALUES
  (
    'LOOKUP_NAME',
    'List of something',
    'TABLE_NAME',
    'PK',
    'FIELD1,FIELD2',
    'PK,FIELD1,FIELD2',
    '~PkCaption,Caption1,Caption2',
    'BTU',
    SYSDATE
  );

UPDATE SI_LOOKUP SET LK_DATEENABLE = 'ST_START_DATE', LK_DATEDISABLE='ST_END_DATE'
WHERE LK_CODICE='SA_STUDENT_LK';
```

6. Add the property in your model
7. Modify the **getObjectInfo** to indicate the lookup to use for your new property

```
@Override
  public Object getObjectInfo(String info, String propertyName) {
    if(DynamicInfoInterface.PROPERTY_LOOKUP.equals(info)) {
      if("property_name".equals(propertyName)) {
        return "LOOKUP_NAME";
      }
    }

    if (infoDelegate == null) {
      infoDelegate = DynamicInfoUtils.getBeanDynamicInfo(YourBean.class);
    }

    return infoDelegate.getObjectInfo(info, propertyName);
  }
```

8. Place your lookup on your view

```
final SimpleLookupExecutor executor = new SimpleLookupExecutor(controller.getUser(),
"property_name");
executor.setLookupName("LK_CODICE_LOOKUP");
executor.add(new LookupFieldDef(0, "LK_KEYFIELDS_LOOKUP", "Codice", 1,
FieldType.key, true, String.class));
executor.add(new LookupFieldDef(1, "LK_LOOKUPFIELDS_LOOKUP", "Descrizione", 2,
FieldType.description, true, String.class));
executor.setDisplayPattern("{0} - {1}");

FormLookupField ownerField = new FormLookupField(executor);
ownerField.setLabel("Label");
panel.add(ownerField, 6);
bindings.addBinding(new SimpleBinderBinding(ownerField, "property_name"));




// In case of Grid
LookupFactory<LookupBuilderContext> factory = new
LookupFactory<LookupBuilderContext>() {
      @Override
      public AbstractLookupExecutor build(LookupBuilderContext ctx) {
       SimpleLookupExecutor executor = new
       SimpleLookupExecutor(controller.getUser(), "studentID");
       executor.setLookupName("SA_STUDENT_LK");
       executor.add(new LookupFieldDef(0, "ST_ID", "Codice", 1, FieldType.key, true,
       Long.class));
       executor.add(new LookupFieldDef(1, "ST_NAME", "Name", 2,
       FieldType.description, true, String.class));
       executor.add(new LookupFieldDef(2, "ST_SURNAME", "Surname", 2,
       FieldType.description, true, String.class));
       executor.setDisplayPattern("ID {0} - {1} {2}");
         executor.setFilterProvider(ctx.getFilterProvider());
         return executor;
      }
    };

columns.add(new Column().caption("Student").name("studentID").type(new
LookupNumericColumnType<>(factory, new GridLookupBuilderContext(controller,
"studentID"))).factory(SimpleBinderBinding.getFactory(true)).width(200));
```

Define a FIND2

1. Add the destination controller to your lookup executor

```
executor.setFindForward(caller, "destination.mapping.do");
```

Define a FIND3

1. Define your Find3Component

```
ControllerForwardDescriptor descriptor = new
ControllerForwardDescriptor(controller);
descriptor.forwardTo("/destination.mapping.do");

Find3Component ddt = new Find3Component("Title", descriptor);
ddt.setFind3KeyCb(new CallbackParameter<List<Map<String,Object>>>() {
       @Override
       public void call(List<Map<String,Object>> keys) {
              //get the keys
       }
});

ddt.attachAndOpen();
```

Implement a listener

1. To whatver widget call the method **addEventListener** and specify the kind of event you want to listen to (click, change, mousedown…etc…)

```
Button button = new Button(new HTML("Click me!"));
button.addEventListener(new EventListener(BaseEvent.CLICK, "Wait, I'm
processing...", button) {
      @Override
      public void onEvent(Map<String, String> eventProperties) {
        doSomething();
       }
});

  protected void doSomething() throws Exception {


    Connection connection = null;
    try {
      // I take a DB connection
      connection = ConnectionManager.getConnection(user, 1000);

// I create a service context
    ServiceContext serviceContext = new ServiceContext(user);
      serviceContext.setConnection(connection);

      // I create a new Object
      BinderInterface newBinder = ServiceManager.getObject(MyBean.class.getName(),
null, serviceContext);

      // I set the properties into the binder
      newBinder.setPropertyValue("property1", "value", serviceContext);
      newBinder.setPropertyValue("property2", "value", serviceContext);

      // I save the new binder
      ServiceManager.save(newBinder, serviceContext);

      // I read the properties from the binder
      Object id = newBinder.getPropertyValue("id");

      // I read a binder by ID
      BinderInterface binder = ServiceManager.getObject(MyBean.class.getName(), id,
serviceContext);

      connection.commit();
    } catch(Exception e) {
      controller.getContainerComponent(TopBar.class).addGlobalError("Error!");
    } finally {
      /* I always release the DB connection, no matter the result.
       * The release also do the rollback of the connection.
       * NEVER close the connection!
       */

      ConnectionManager.releaseConnection(connection);
    }
```

3. Misc

Use an Attach

```
    // I create the container
    AttachContainer container = new AttachContainer("ATTACH_TYPE", "KEY", null);

    // I read one of the attach inside the container
    Attachment attachment = container.getAttachment("FILE_NAME");
    InputStream is = attachment.getAttach();

    // I save a new attach
    Attachment newAttach = new Attachment(container);
    newAttach.setNomeFile("file_name");
    newAttach.setIs(is);
    container.AttachFile(connection, newAttach);
```

Read a profile key

```
// Read a user profile key
      Object keyValue =
ProfileManager.getProfileManager().getObject(ProfileManager.USER_TYPE,
"path.of.the.key", user, dimension can be null);

      // Read a parametr profile key
      keyValue =
ProfileManager.getProfileManager().getObject(ProfileManager.PARAMETER_TYPE,
"path.of.the.key", user, dimension can be null);
```

Execute a query

```
Object[] result = ResultSetHelper.SingleRowSelect(connection, "SELECT COUNT(*) FROM
TABLE");
    if(result!=null) {
      Object value = result[0];
    }



List<List> result = new ArrayList<>();
        ResultSetHelper.fillListList(connection, "SELECT * FROM DUAL",
result);

        for (List record : result) {
          Object column1 = record.get(0);
          //...
        }
```

Define and use a logger

```
private static final org.apache.log4j.Logger logger =
org.apache.log4j.Logger.getLogger(YourClass.class);


logger.error(e, e);
logger.warning(e, e);

//Error - Super bad
//Warning
//Info
//Debug - Used by developers
```

Useful API

```
//Converts an object to whatever class
ClassMapper.classToClass(object, String.class);
```

ChangeLog: create table

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">

       <changeSet id="changeset-test" author="Mario Rossi" context="application:PSGExt">
              <preConditions onFail="MARK_RAN">
                     <not>
                            <tableExists tableName="MY_TABLE" />
                     </not>
              </preConditions>

              <createTable tableName="MY_TABLE">
                     <column name="MY_COLUMN" type="varchar(255)">
                            <constraints nullable="false" />
                     </column>
              </createTable>
       </changeSet>
</databaseChangeLog>
```

ChangeLog: insert a record

```xml
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">

       <changeSet id="changeset-test-insert" author="Mario Rossi"
context="application:PSGExt">
              <preConditions onFail="MARK_RAN">
                     <sqlCheck expectedResult="0">SELECT count(*) FROM MY_TABLE WHERE
MY_COLUMN = 'X';</sqlCheck>
              </preConditions>

              <sql>
                     INSERT INTO MY_TABLE VALUES('X');
              </sql>
       </changeSet>
</databaseChangeLog>
```