Page Bean Modularization

Starting with Release 3.0 update 20100614 CaptainCasa introduced an extended concept for modularization – the modularization via “PageBean” instances. Its aim is to provide a simple, object oriented way of covering complex navigation scenarios.

Typical examples are:

- One page (and its logic) should be usable as object in other pages (and their logic).
- There is a clear relationship between the using page (and its logic) and the used page (and its logic).
- A page can occur multiple times within an other page.
- There may be a complex, multi-level hierarchy of nested pages.

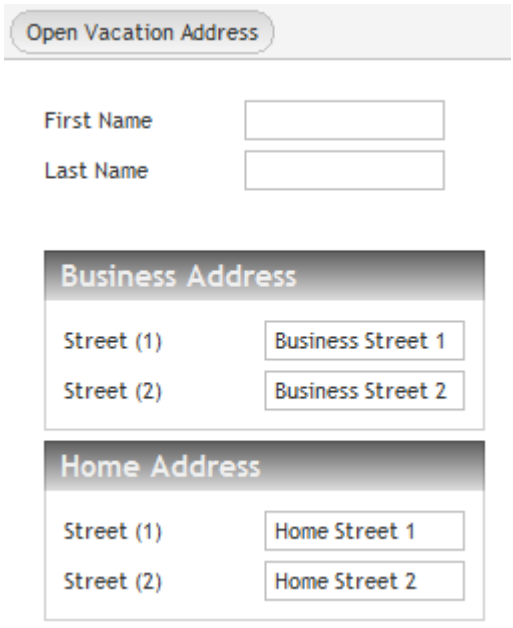
The Page Bean Modularization follows the normal ROWINCLUDE navigation and its possibilities to dynamically update page expressions via the CONTENTREPLACE attribute. Actually it is an automation of both aspects: including one page and updating its expressions, so that it points to the bean that it works with.

Before getting into detail about what happens in the background, let's take a look onto an example scenario.

Example

Page

This is the screenshot of the example:



There is an “outside” page containing two “inside” pages, both sharing the same jsp-page, but containing different instance data. When pressing the “Open Vacation Address” button then a third instance of the address is shown in a popup window.

Java Code

Let's take a look onto the code of the outside page first:

```
package workplace;

...

public class DemoPageBeanUI
    extends WorkpageDispatchedPageBean
    implements Serializable
{
    // ---------------------------------------------------------------------
    // members
    // ---------------------------------------------------------------------

    protected String m_lastName;
    protected String m_firstName;

    DemoPageBeanAddressUI m_homeAddress = new DemoPageBeanAddressUI();
    DemoPageBeanAddressUI m_businessAddress = new DemoPageBeanAddressUI();
    DemoPageBeanAddressUI m_vacationAddress = new DemoPageBeanAddressUI();

    ModalPopup m_popup;

    // ---------------------------------------------------------------------
    // constructors
    // ---------------------------------------------------------------------

    public DemoPageBeanUI(IWorkpageDispatcher dispatcher)
    {
        super(dispatcher);
        initPageBeans();
    }

    public String getPageName() { return "/workplace/demopagebean.jsp"; }
    public String getRootExpressionUsedInPage() { return "#{wp.DemoPageBeanUI}"; }

    // ---------------------------------------------------------------------
    // public usage
    // ---------------------------------------------------------------------

    public DemoPageBeanAddressUI getHomeAddress() { return m_homeAddress; }
    public DemoPageBeanAddressUI getBusinessAddress() { return
m_businessAddress; }

    public String getLastName() { return m_lastName; }
    public void setLastName(String value) { m_lastName = value; }

    public String getFirstName() { return m_firstName; }
    public void setFirstName(String value) { m_firstName = value; }

    public void onOpenVacationAddress(ActionEvent event)
    {
        openModelessPopup(m_vacationAddress,"Vacation Address",0,0,new
ModelessPopup.IModelessPopupListener()
        {
            public void reactOnPopupClosedByUser()
            {
                closePopup(m_vacationAddress);
            }
        });
    }

    // ---------------------------------------------------------------------
    // private usage
    // ---------------------------------------------------------------------

    private void initPageBeans()
    {
        m_homeAddress.setTitle("Home Address");
        m_homeAddress.setStreet1("Home Street 1");
        m_homeAddress.setStreet2("Home Street 2");
        m_businessAddress.setTitle("Business Address");
        m_businessAddress.setStreet1("Business Street 1");
        m_businessAddress.setStreet2("Business Street 2");
        m_vacationAddress.setTitle("Vacation Address");
        m_vacationAddress.setStreet1("Vacation Street 1");
        m_vacationAddress.setStreet2("Vacation Street 2");
    }

}
```

Each contained page is represented by a corresponding page instance (m_homeAddress, m_businessAddress and m_vacationAddress), with corresponding get-methods for accessing the instances. The sub-pages are members / properties of the outside page.

In addition you see that there are two special methods “getPageName()” and “getRootExpressionUsedInPage()” that are implemented.

The code of the contained instances is:

```
package workplace;

...

public class DemoPageBeanAddressUI
    extends PageBean
    implements Serializable
{
    // ------------------------------------------------------------------------
    // members
    // ------------------------------------------------------------------------

    protected String m_title = "<Title>";
    protected String m_street1 = "<Street1>";
    protected String m_street2 = "<Street2>";

    // ------------------------------------------------------------------------
    // public usage
    // ------------------------------------------------------------------------

    public String getPageName() { return "/workplace/demopagebeanaddress.jsp"; }
    public String getRootExpressionUsedInPage() { return
"#{wp.DemoPageBeanAddressUI}"; }

    public String getStreet1() { return m_street1; }
    public void setStreet1(String value) { m_street1 = value; }

    public String getStreet2() { return m_street2; }
    public void setStreet2(String value) { m_street2 = value; }

    public String getTitle() { return m_title; }
    public void setTitle(String value) { m_title = value; }

}
```

Again the code contains an implementation of the “getPageName()” and “getRootExpressionUsedInPage()”.

JSP Page

Now, let's take a look onto the outside jsp page definition:

```
/workplace/demopagebean.jsp:

<t:rowheader id="g_1">
  <t:button id="g_2"
            actionListener="#{wp.DemoPageBeanUI.onOpenVacationAddress}"
            text="Open Vacation Address" />
</t:rowheader>
<t:rowbodypane id="g_3" rowdistance="5">
  <t:row id="g_4">
    <t:label id="g_5" text="First Name" width="100" />
    <t:field id="g_6" text="#{wp.DemoPageBeanUI.firstName}" width="100" />
  </t:row>
  <t:row id="g_7">
    <t:label id="g_8" text="Last Name" width="100" />
    <t:field id="g_9" text="#{wp.DemoPageBeanUI.lastName}" width="100" />
  </t:row>
  <t:rowdistance id="g_10" height="20" />
  <t:rowpagebeaninclude id="g_11"
            pagebeanbinding="#{wp.DemoPageBeanUI.businessAddress}" />
  <t:rowpagebeaninclude id="g_12"
            pagebeanbinding="#{wp.DemoPageBeanUI.homeAddress}" />
</t:rowbodypane>
<t:rowstatusbar id="g_13" />
```

The integration of the two address pages into the outside page is not done by using the ROWINCLUDE – but by using the ROWPAGEBEANINCLUDE component.

The jsp page definition of the address page is a “very normal” one:

```
/workplace/demopagebeanaddress.jsp:

<t:row id="g_1">
  <t:pane id="g_2" border="#00000030" padding="0" rowdistance="0">
    <t:rowtitlebar id="g_3" text="#{wp.DemoPageBeanAddressUI.title}" />
    <t:row id="g_4">
      <t:pane id="g_5" padding="10" rowdistance="5">
        <t:row id="g_6">
          <t:label id="g_7" text="Street (1)" width="100" />
           <t:field id="g_8" text="#{wp.DemoPageBeanAddressUI.street1}"
                                          width="100" />
        </t:row>
        <t:row id="g_9">
          <t:label id="g_10" text="Street (2)" width="100" />
          <t:field id="g_11" text="#{wp.DemoPageBeanAddressUI.street2}"
                                          width="100" />
        </t:row>
      </t:pane>
    </t:row>
  </t:pane>
</t:row>
```

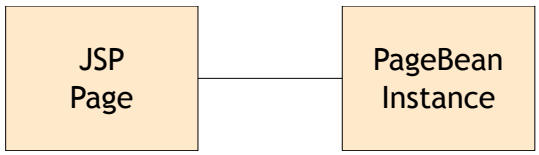
What you see...

You see, that modularization is much simpler than using the “native” ROWINCLUDE component – because the including of the pages is directly reflected within the beans' structure.

You do not have to care about updating expressions by using CONTENTREPLACE at all- all this is manged internally.

Concepts

The concepts are based on the paradigm that for one page there is one bean serving the page:



Example: the “demopagebeanaddress.jsp” is related to the “DemoPageBeanAddressUI” class – all its content is managed by the class.

IPageBean / PageBean Instances

A bean covering all the aspects of one page is called “PageBean” - and implements the interface “IPageBean”, typically be deriving from the “PageBean” class.

A page bean has to implement an interface that allows the modularization framework to integrate it into other JSP pages – executed by the ROWPAGEBEANINCLUDE component. The two methods that need to be provided are:

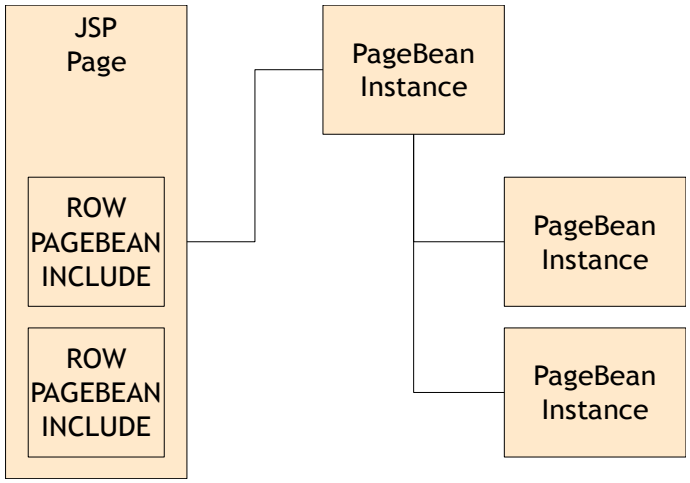
- getPageName() - this method needs to return the page name of the page that is to be used for the bean. Typically this is one page, but it's also possible to return a page name that fits best to the application's situation – if you have multiple JSP-pages that

use the same page bean.

- getRootExpressionUsedInPage() - this method needs to tell the expression that is used inside the .jsp definition for accessing the page bean object. This sounds complex but it isn't: in the example above, within the page “demopagebanaddress.jsp” the page bean is addressed by the expression “#{d.DemoPageBeanAddressUI}” - so that's the root expression that is used within the page definition!

The ROWPAGEBEANINCLUDE Component

This is the component to embed a page bean into a page.



Embedding covers two aspects:

- 1. - The jsp page of the page bean is included.
- 2. - The expressions in the page that is included are updated so that they fit to the current scenario and so that the page's content is really pointing to the correct page bean instance.

Result: there is no additional effort for integrating page bean instances within a page. The page beans instances are normal members / properties of other page beans instances. While the normal ROWINCLUDE component just optically includes a page into another pages, the ROWPAGEBEANINCLUDE compoennt

What happens internally...

Inside the management of the ROWPAGEBANINCLUDE component there is an automated update of the expressions of an included page.

Let's assume the following scenario:

- The outside page is accessed through expression: “#{d.d_1.DemoPageBeanUI}”.
- It includes an insde page bean (as in the example above) that is made available through the property “businessAddress”. Consequence: the expression for accessing the inside page's root object is “#{d.d_1.DemoPageBeanUI.businessAddress}”.
- The inside page is defined within a JSP page that point to its root object via “#{d.DemoPageBeanAddressUI}”. Result: the expression replacement is defined, so that all references within the inside page are updated from “#{d.DemoPageBeanAddresUI.*}” to “#{d.d:1.DemoPageBeanUI.businessAddress.*}”.

Popup Management

Each page bean instance inherits the following methods for opening and closing popup windows:

- openModalPopup(IPageBean pageBean, ...)
- openModelessPopup(IPageBean pageBean, ...)
- closePopup(IPageBean pageBean)

The open-methods return back a just normal instance of ModalPopup / ModelessPopup.

By opening the popup windows through the page bean, all the “rightsizing” of the expressions within the loaded page takes place automatically – there is no further effort required on your side.

Please do not use the core-methods of the popup management when working with page beans but always go through the open-methods of the page ban.

Integration Issues

There are no restrictions for using Page-Bean-pages within non-Page-Bean-pages. You can start to use Page-Bean instances at any level of other pages, that are not managed via Page-Beans.

Vice versa you can use any type of normal page integration via ROWINCLUDE also in the layout definition of “page bean pages” - you just need to carefully think about expressions to be applied.

Classes that are affected with Page Bean Management

The following classes have been added:

- “IPageBean” - interface that is required for each page bean instance
- “PageBean” - default implementation, should be the base of all your implementations
- “DefaulDispatchedPageBean”, “WorkpageDispatchedPageBean” - default implementations for page beans that are used within a dispatcher / workplace scenario.

Summary

The Page Bean Modularization is a significant simplification of dealing with scenarios, in which pages are embedded within other pages for modularization.

The so called Page Bean instances are nested into one another, so that there is a clear object arrangement, in which one page bean owns its contained page beans.

We will, step by step com