Creating Component Editors for Composite Web Form Elements

Applies to:

SAP NetWeaver 7.0 SPS 14 Usage Type EP, Web Page Composer For more information, visit the Portal and Collaboration homepage.

Abstract

When creating custom Web forms for the Web Page Composer, the engineers can choose between a number of out-of-the box elements to compose the Web form. Many of them feature a single editor (e.g. an input field or an html editor). However, it is often necessary to have a number of editors for a single composite element. In such a case, it is more convenient to treat them as a single logical unit of editors, operating on the same element.

This guide leads you through a comprehensive and easy to follow step-by-step procedure for creating a component editor for composite elements and integrating it into Web forms.

Author: Tsanko Aleksandrov

Company: SAP Labs Bulgaria

Created on: 01 October 2008

Author Bio

Tsanko Aleksandrov is a Java Developer Internship, working in the area of SAP NetWeaver Portal - Knowledge Management and Collaboration. He is an undergraduate in Software Engineering at the Sofia University, Bulgaria.

Table of Contents

Introduction

When creating custom web forms for Web Page Composer, the engineers can choose between a number of out-of-the box elements to compose the Web form. Many of them features a single editor (e.g an input field or an html editor). However it is often necessary to have a number of editors for a single composite element.

Being able to create more complex editor components, engineers can construct web forms that provide richer user experience. Users are able to easily work with a number of editors, which are structured as a single logical unit. This results in an easier manipulation of web forms in design time. Moreover, you avoid major inconveniences, such as separate manipulation of otherwise logically unified editors and wrong positioning of the different editors.

An example of this approach is the design of a Web form for Frequently Asked Question (FAQ) examined in this article.

The standard design for pages with FAQ information usually comprises a list of pairs, each of them consisting of a question and its associated answer. Therefore, the design of a FAQ web form requires two editors – one for the question and one for the answer.

If we use separate editors, then the following major inconveniences appear:

- When a “question and answer” pair is needed, each element in the form editor should be added separately. Moreover, there is no tight pairing between the questions and the answers and it is a responsibility of the author to ensure that each question has an associated answer and that they appear correctly ordered in the list.
- When we want to move a “question and answer” pair up or down in the list of “question and answer” pairs, we have to move the question and answer editors separately. This is error-prone, especially in long lists since it is a responsibility of the author to move the pair as a logical unit.

These inconveniences could be avoided, if the question and answer editors are treated as a single composite element. In the following part of this article, we will try to show you how this could be achieved by creating a new editor component.

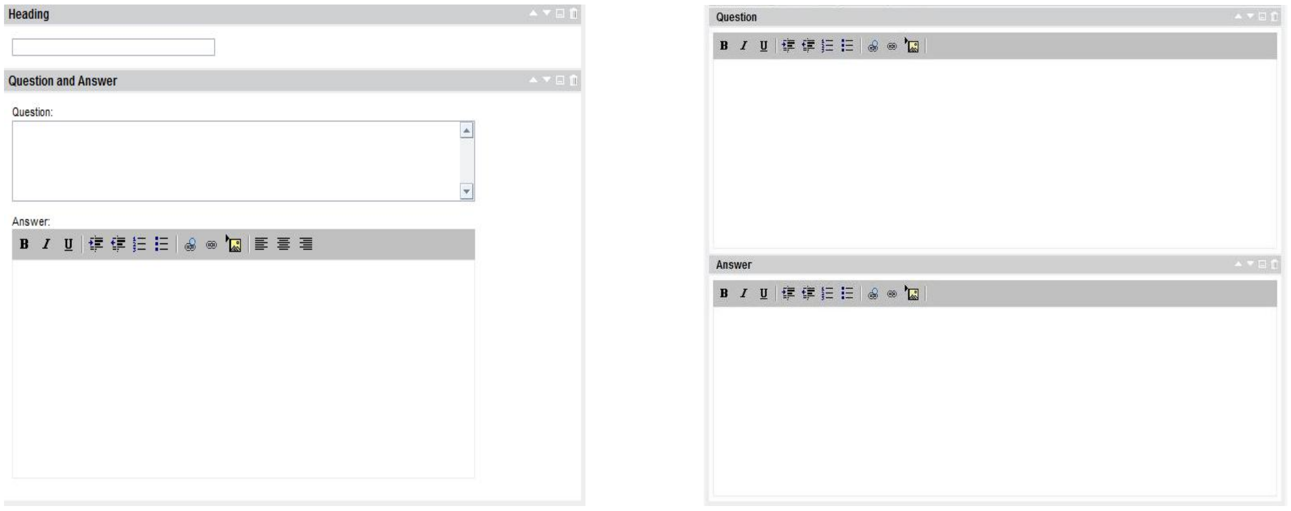


Figure 1 Composite vs. Separated Editors

Prerequisites

- You have installed SAP Net Weaver 7.0 SPS 14 Usage Type EP
- You have deployed the Web Page Composer Add-on
  More Information: SAP Note 1080110: “Installing the Web Page Composer”
- You have the *Content Administrator* and the *System Administrator* roles
- You have the WPC Editor Role assigned to your user (required only for testing)
- You have installed the SAP NetWeaver Developer Studio

Creating an Editor Component

Create a New Portal Application

In order to create a new editor component, you need to create a portal application.

For more information about creating a portal application, see Creating Portal Application Projects

Design of the new Editor Component

Editor components extend the `com.sap.nw.wpc.km.service.editor.component.AbstractEditorComponent`. The `AbstractEditorComponent` implements the basic functionality for all editor components that are simple enough and can be represented by a single input field.

Each editor component stores its value in its `coreValue` instance field. Since the goal is to have several editors, it is essential to provide a mechanism for storing and initializing them. In order to do that we add additional fields for the values of the extra editors.

In the case of the FAQ Web form `coreValue` is used to store the value of the question in the “question and answer” pair.
In order to be able to save the value of the answer in the “question and answer” pair, a new field is being added (“`answerValue`”).

The core aspects of editor component’s functionality are:

- initializing from either XML(when loading the form for editing) or page context (when reloading the design time page)
- storing the content of the editor
- creating the design time UI components (text editors, input fields, html editors, etc.)

An important detail for saving the values in XML is the choice of appropriate XML attributes. XML attributes are defined in the interface `com.sap.nw.wpc.km.service.editor.IEditorConsts`.

Create a New Editor Component

In your portal application project in SAP NetWeaver Developer Studio, create a new class that extends the AbstractEditorComponent.

```
The following methods should be overridden: init (), initializeFromPageContext (),
initializeFromXML (), getXMLElement ().The skeleton of the FAQ class looks like this:

   public class FAQComponent extends AbstractEditorComponent {
   private static final String PADDING_RIGHT = "10px";
   private static final String PADDING_LEFT = "0px";
   private static final String PADDING_BOTTOM = "10px";
   private static final String Question_ = "question_";
   private static final String Answer_ = "answer_";
   private String answerValue;

   public FAQComponent() {

   }

   public Element getXMLElement(
          Document document,
          String elementId,
          String tagName,
          boolean includePropertyValues) {

          // will be explained later
   public void initialiseFromXML(
          String xmlContent,
          Attributes attributes,
          IPropertyMap properties,
          AbstractEditorObject object,
          IPortalComponentRequest request) {

// will be explained later
   public void initializeFromPageContext(
          AbstractEditorObject editorObject,
          IPageContext context,
          ComponentConfig cc) {

// will be explained later
   private Component getComponent(
          TextEdit question,
          HtmlEditorComponent answer,
          AbstractEditorObject editorObject,
          Locale locale) {

// will be explained later
   }
   public void init(AbstractEditorObject editorObject, ComponentConfig cc) {
          initializeFromPageContext(editorObject, null, cc);
// will be explained later
   }
}
```

Note: The extra field for storing the value of the answer Html editor is defined as follows private String answerValue.

- **init**()-the only thing the init() method does is to call the initializeFromPageContext()

```
public void init(AbstractEditorObject editorObject, ComponentConfig cc) {
initializeFromPageContext(editorObject, null, cc);
}
initializeFromPageContext () that method initializes the composite editor from the
page context. The localization takes place in this method as well.

   public void initializeFromPageContext(
          AbstractEditorObject editorObject,
          IPageContext context,
          ComponentConfig cc) {
          super.initializeFromPageContext(editorObject, context, cc);

          TextEdit question = null;
          HtmlEditorComponent answer = null;

          if (context != null) {
                 question =
                        (TextEdit) context.getComponentForId(
                               Question_ + editorObject.getId());
                 String value =
(String) context.getAttribute(Question_ + editorObject.getId());

                 if (value != null) {
                        question.setText(value);
context.setAttribute(Question_ + editorObject.getId(), null);
                 }
                 answer =
                        (HtmlEditorComponent) context.getComponentForId(
                               Answer_ + editorObject.getId());

                 if (question != null) {
                        setCoreValue(question.getText());
                 }

                 if (answer != null) {
                        this.answerValue = answer.getText();
                 }

          }
          IPortalComponentRequest request =
                 (IPortalComponentRequest) context.getRequest();

          Locale locale = Utils.getCurrentUserLocale(request);
setHtmlbComponent(getComponent(question, answer, editorObject, locale));
   }
```

In the preceding method implementation, `editorObject` has a vital part in retrieving the editors from the context component. `EditorObject` is an instance of a class that extends `com.sap.nw.wpc.km.service.editor.document.AbstractEditorObject`. Due to the fact that we have more than one editors, a way to differ them is required. This is handled by adding a prefix to the ID of the `editorObject`.

```
question = (TextEdit) context.getComponentForId (Question_ + editorObject.getId())
      and
answer = (HtmlEdit) context.getComponentForId(Answer_ + editorObject.getId())
```

- **initializeFromXML** () performs the same actions as initializeFromPageContext() but initialization data is taken from the XML file.

```
  public void initialiseFromXML(

         String xmlContent,
         Attributes attributes,
         IPropertyMap properties,
         AbstractEditorObject object,
         IPortalComponentRequest request) {

TextEdit question = new TextEdit(Question_ + this.editorObject.getId());
HtmlEditorComponent answer = new HtmlEditorComponent(Answer_ +
this.editorObject.getId());

         question.setText(attributes.getValue(IEditorConsts.ATTR_TITLE));
         setCoreValue(attributes.getValue(IEditorConsts.ATTR_TITLE));

         answer.setText(attributes.getValue(IEditorConsts.ATTR_TARGET));
         this.answerValue = attributes.getValue(IEditorConsts.ATTR_TARGET);

         Locale locale = Utils.getCurrentUserLocale(request);
         setHtmlbComponent(getComponent(question, answer, editorObject,
  locale));
    }
```

getXMLelement() - This method creates an XML element to be added in the output XML file.

```
  public Element getXMLElement(
         Document document,
         String elementId,
         String tagName,
         boolean includePropertyValues) {

         Element element = document.createElement(tagName);
         element.setAttribute(IEditorConsts.ATTR_TYPE, elementId);
         element.setAttribute(IEditorConsts.ATTR_TITLE, getCoreValue());
         element.setAttribute(IEditorConsts.ATTR_TARGET, answerValue);
         return element;}
```

Note: Hence, the attributes `target` and `title` have been chosen for saving the values of the editors. They are chosen among the attributes defined in `IEditorConsts`. Choosing attributes that are not defined in `IEditorConsts`, results in XML parsing errors.

getComponent() – This private method is invoked by initializeFromXML() and initializeFromPageContext() methods in order to create the UI widgets. In Web forms HTMLB UI components are used.

```
private Component getComponent(
         TextEdit question,
         HtmlEditorComponent answer,
         AbstractEditorObject editorObject,
         Locale locale) {

         FormLayout fl = new FormLayout();
         if (question == null) {
               question = new TextEdit(Question_ + editorObject.getId());
               question.setText(editorObject.getDefaultValue());
         }

         String qLabelText =
               FAQResourceBundle.getInstance().getString("question", locale);
         Label qLabel = new Label(qLabelText);

         String aLabelText =
               FAQResourceBundle.getInstance().getString("answer", locale);
         Label aLabel = new Label(aLabelText);

         if (answer == null) {
answer = new HtmlEditorComponent(Answer_ + editorObject.getId());
         }


         question.setCols(100);

         FormLayoutCell flc1 = fl.addComponent(1, 1, qLabel);
         flc1.setPaddingLeft(PADDING_LEFT);

         FormLayoutCell flc2 = fl.addComponent(2, 1, question);
         flc2.setPaddingLeft(PADDING_LEFT);
         flc2.setPaddingBottom(PADDING_BOTTOM);

         FormLayoutCell flc3 = fl.addComponent(3, 1, aLabel);

         FormLayoutCell flc4 = fl.addComponent(4, 1, answer);
         flc4.setPaddingLeft(PADDING_LEFT);
         flc4.setPaddingBottom(PADDING_BOTTOM);
         return fl;
   }
```

For more information on HTMLB UI components, consult SAP HTMLB Guide Lines .

Create Service Wrapper

The registration of the deployable unit is handled by a service. This service implements the IService and each deployable unit should have a unique key for the purpose of identification and class loading.

```
public interface IFAQServiceWrapper extends IService {
         public static final String KEY = "com.sap.nw.wpc.core.faq";
  }
package com.sap.nw.wpc.km.service.editor.component;
import com.sap.tc.logging.Location;
import com.sapportals.portal.prt.service.IServiceConfiguration;
import com.sapportals.portal.prt.service.IServiceContext;
import com.sapportals.wcm.crt.CrtClassLoaderRegistry;

public class ComponentServiceWrapper implements IFAQServiceWrapper {

   private static Location LOCATION =
         Location.getLocation(ComponentServiceWrapper.class);

   private IServiceContext mm_serviceContext;

   public void init(IServiceContext arg0) {
         mm_serviceContext = arg0;
         LOCATION.debugT("PB Core : Service wrapper init");
         CrtClassLoaderRegistry.addClassLoader(
                this.getKey(),
                this.getClass().getClassLoader());

   }

public void afterInit() {
}
   public void configure(IServiceConfiguration arg0) {
   }
   public void destroy() {
   }
   public void release() {

   }
   public IServiceContext getContext() {
         return null;
}
   public String getKey() {
         return null;
   }
}
```

Enable Internationalization

For the purpose of internationalization, you need to extend the `com.sap.nw.wpc.bundles.ResourceBundleBase` and to create you own bundle files. The process is not different than the localization in Java. The necessary minimum is represented by the following sample code.

```
public class FAQResourceBundle extends ResourceBundleBase {
      public static final String PROPERTIES_FILE_PATH =
      "com.sap.nw.wpc.service.editor.bundle.mybundlefile";
      private static FAQResourceBundle instance;


      private FAQResourceBundle() {
      }

      public String getPropertiesFilePath(){
            return PROPERTIES_FILE_PATH;
      }

      protected ClassLoader getClassLoader() {
            return CrtClassLoaderRegistry.getClassLoader();
      }

      public static FAQResourceBundle getInstance(){
            if(instance == null){
                  instance = new      FAQResourceBundle();
            }
            return instance;
      }

    }
```

Deploy

There are two ways to deploy your composite editor:

- Deploy by placing a generated PAR file into the portal's file system and restart the portal afterwards. Generated PAR files can be placed under apps/sap.com/irj/servlet_jsp\irj\root\WEB-INF\deployment\pcd.

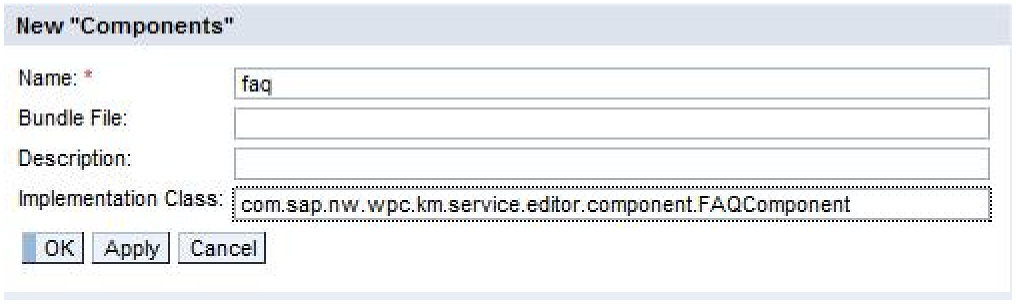
- Deploy to a running portal using either the deployment iView or the PAR Export Wizard of the SAP NetWeaver Developer Studio. More information: SAP Help Portal

Registering the Composite Editor Component in the Web Page Composer

The composite editor component has to be registered as an editor component for the Web Page Composer.

To do this, perform the following:

1. Go to System Administration > System Configuration > Knowledge Management > Web Page Composer > Editors > Editor Component > Components.
2. Choose New.
3. Specify the properties of the new component.


New "Components"

Name: *	faq
Bundle File:	
Description:	
Implementation Class:	com.sap.nw.wpc.km.service.editor.component.FAQComponent

Figure 2: A Registered Composite Editor

4. Result

You have configured the new editor component, thus you can now use them when you define your Web Forms in XML .

Creating a Custom Web Form

For more information about the required procedures for creating a custom Web form, see Create New Web Forms

An example of an XML file:

```
   <documenttype id="wpc_faq" description="xml.xlbl.web_faq" showpreview="true"
showelementlist="true">
     <properties>
<property id="fileName" description="xml.xlbl.filename" type="inputfield" size="25"
isrequired="true" isfilename="true" />
<property id="title" description="xml.xlbl.title" type="inputfield" size="25" isrequired="false"
/>
<property id="TableOfContents" description="TableOfContents" type="checkbox"
defaultvalue="false" property="wpc_wcm_toc" />
<property id="ShowBTT" description="ShowBTT" type="checkbox" defaultvalue="false"
property="wpc_wcm_toc" />
    </properties>
    <elements>
                    <element id="heading" description="heading" type="inputfield" default="true"
/>
                    <element id="qanda" description="qanda" type="faq" default="true" />
    </elements>
   </documenttype>
```

Result

The result design time editor should look similiar to this:

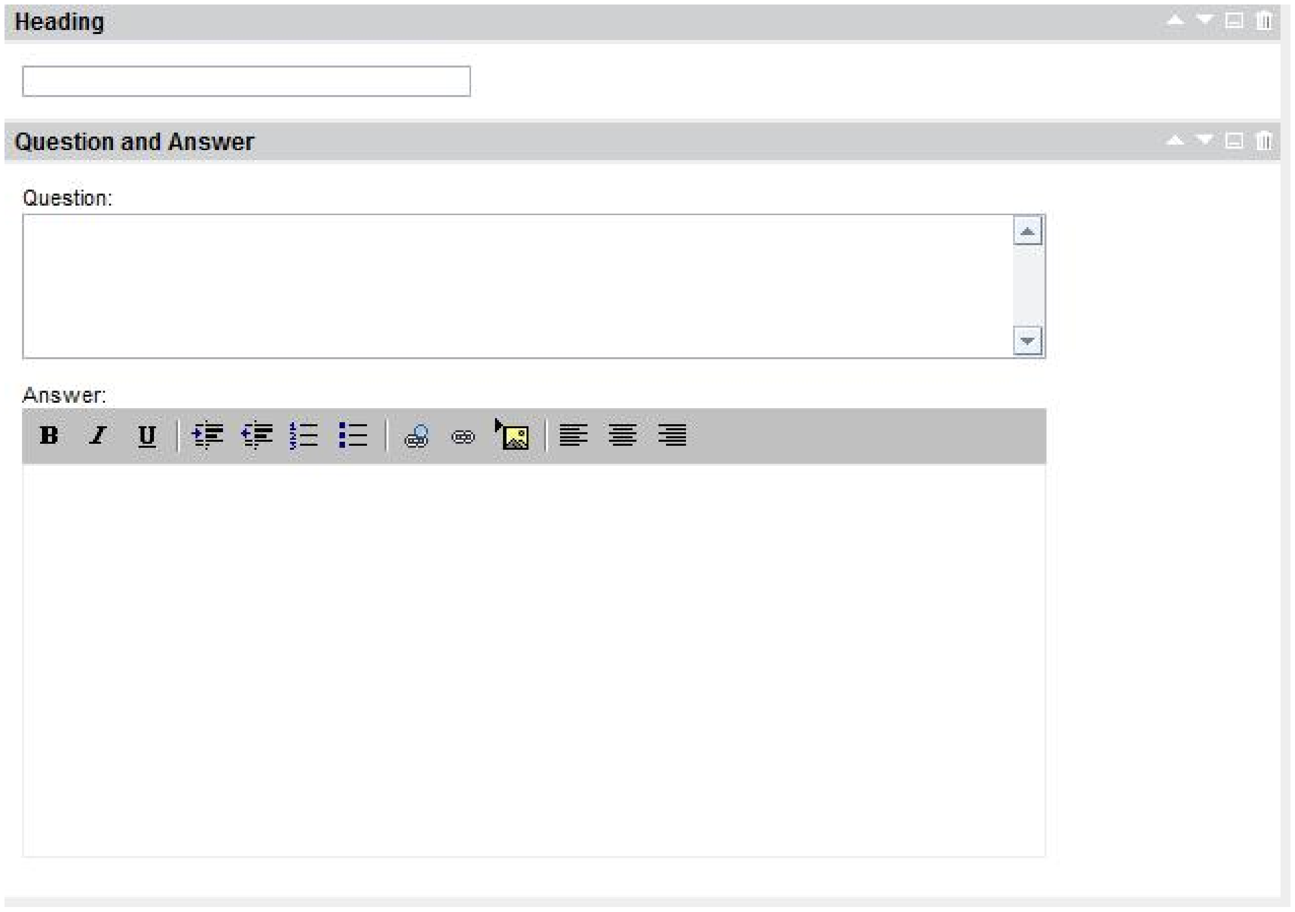
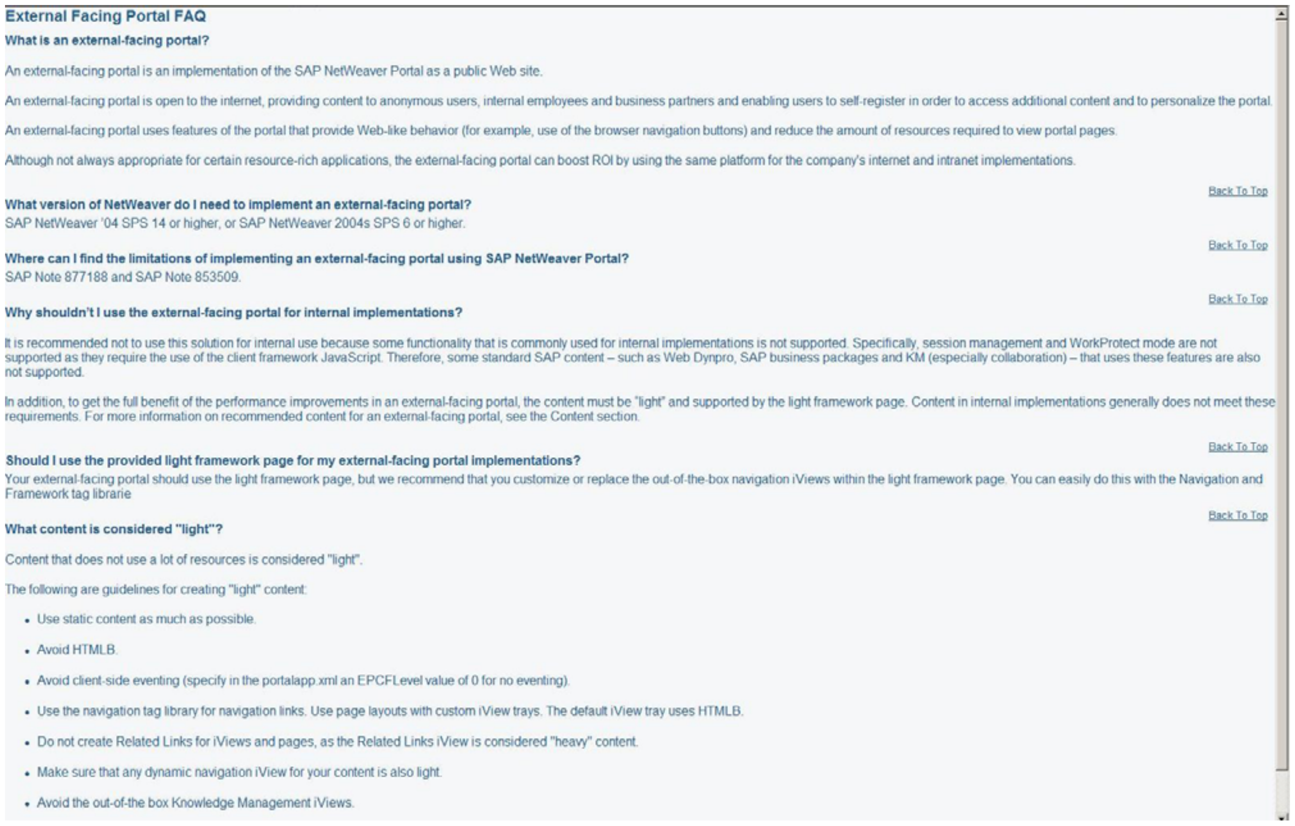


Figure 3: Final FAQ Form

When the FAQ Form is added to a WPC page, it should look like this in runtime:



External Facing Portal FAQ

What is an external-facing portal?

An external-facing portal is an implementation of the SAP NetWeaver Portal as a public Web site.

An external-facing portal is open to the internet, providing content to anonymous users, internal employees and business partners and enabling users to self-register in order to access additional content and to personalize the portal.

An external-facing portal uses features of the portal that provide Web-like behavior (for example, use of the browser navigation buttons) and reduce the amount of resources required to view portal pages.

Although not always appropriate for certain resource-rich applications, the external-facing portal can boost ROI by using the same platform for the company's internet and intranet implementations.

Back To Top

What version of NetWeaver do I need to implement an external-facing portal?

SAP NetWeaver '04 SPS 14 or higher, or SAP NetWeaver 2004s SPS 6 or higher.

Back To Top

Where can I find the limitations of implementing an external-facing portal using SAP NetWeaver Portal?

SAP Note 877188 and SAP Note 853509.

Back To Top

Why shouldn't I use the external-facing portal for internal implementations?

It is recommended not to use this solution for internal use because some functionality that is commonly used for internal implementations is not supported. Specifically, session management and WorkProtect mode are not supported as they require the use of the client framework JavaScript. Therefore, some standard SAP content – such as Web Dynpro, SAP business packages and KM (especially collaboration) – that uses these features are also not supported.

In addition, to get the full benefit of the performance improvements in an external-facing portal, the content must be "light" and supported by the light framework page. Content in internal implementations generally does not meet these requirements. For more information on recommended content for an external-facing portal, see the Content section.

Back To Top

Should I use the provided light framework page for my external-facing portal implementations?

Your external-facing portal should use the light framework page, but we recommend that you customize or replace the out-of-the-box navigation iViews within the light framework page. You can easily do this with the Navigation and Framework tag librarie

Back To Top

What content is considered "light"?

Content that does not use a lot of resources is considered "light".

The following are guidelines for creating "light" content:

- Use static content as much as possible.
- Avoid HTMLB.
- Avoid client-side eventing (specify in the portalapp.xml an EPCFLevel value of 0 for no eventing).
- Use the navigation tag library for navigation links. Use page layouts with custom iView trays. The default iView tray uses HTMLB.
- Do not create Related Links for iViews and pages, as the Related Links iView is considered "heavy" content.
- Make sure that any dynamic navigation iView for your content is also light.
- Avoid the out-of-the box Knowledge Management iViews.

Figure 4 Embedded Web Form in WPC page

Summary:

By constructing more complex editor components, engineers can provide their users with web forms that better facilitate their work due to the fact that a number of editors can be now structured as a single logical unit. This results in an more effective use of web forms in design time. Moreover, you avoid major inconveniences, such as separate manipulation of otherwise logically unified editors and wrong positioning of the different editors.

Related Content

Create Custom Web Forms

For more information, visit the Portal and Collaboration homepage.

Copyright