hyperCMS ContentManagementServer

Version 6.0

Portlet Guide

2016-01-06

Inhalt

1 Portlet nach JSR-168 Standard

1.1 Einleitung

Portlets sind beliebig kombinierbare Komponenten einer Benutzeroberfläche, die von einem Portalserver angezeigt und verwaltet werden. Sie erzeugen Fragmente von HTML-Code und fügen sich in einer Portalseite ein. Typischerweise besteht eine Portalseite aus vielen, nicht-überlappenden Portlet-Fenstern („Kacheln“), in denen jeweils ein Portlet ausgeführt wird. Beispiele für Portlets sind E-Mail, Wetterbericht, Diskussionsforen oder Nachrichten.

1.2 Allgemeines

1.2.1 Portlet Specification

Der JSR 168 stellt einen der wichtigsten Meilensteine in der Geschichte der Portale dar. Er ebnete den Weg für unabhängig vom verwendeten Portal entwickelte Portlets. Dies bietet den Kunden die Möglichkeit, Anwendungen zu schreiben, ohne sich an einen Anbieter binden zu müssen. Wenn auch dieser Gedanke nicht von allen Portalherstellern konsequent durchgesetzt wird, führte der JSR 168 dazu, dass es unterdessen viele Standardportlets gibt, die eine standardisierte Funktionalität unabhängig vom eingesetzten Portal anbieten und von vielen Drittanbietern auf den Markt kommen.

Die in der JSR 168 dokumentierten Portlet API 1.0 enthält 12 Klassen und 14 Interfaces. Von den 12 Klassen sind acht Exceptions. Die Spezifikation standardisiert das Zusammenspiel zwischen Portlet-Container und Portlets.

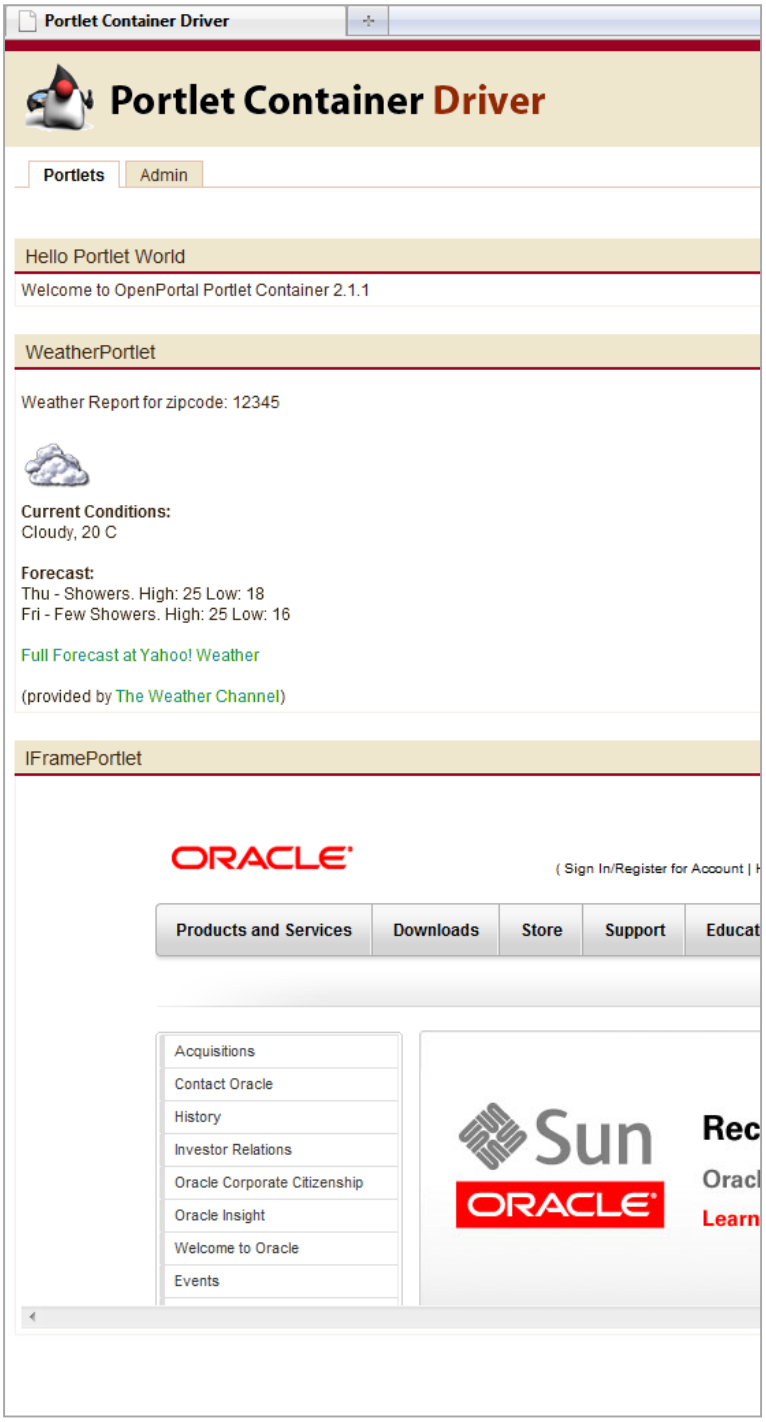

Abbildung 1: Portal mit Portlets (helloworld, weather, iFrame)

1.2.2 Portlet-Container

Ein Portlet-Container ist so zu sagen der Lebensraum eines Portlets. In dieser wird ein Portlet instanziiert, verwendet und schlussendlich zerstört (siehe Abbildung 2). Architektonisch gesehen stellt der Portlet-Container ein Interface zwischen Portal und Portlets dar (siehe Abbildung 3).

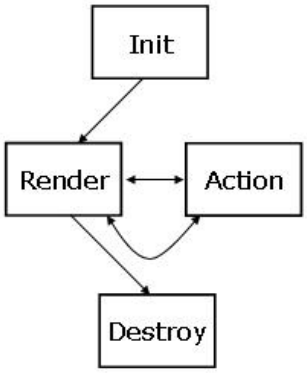

Abbildung 2: Lebenszyklus eines Portlets

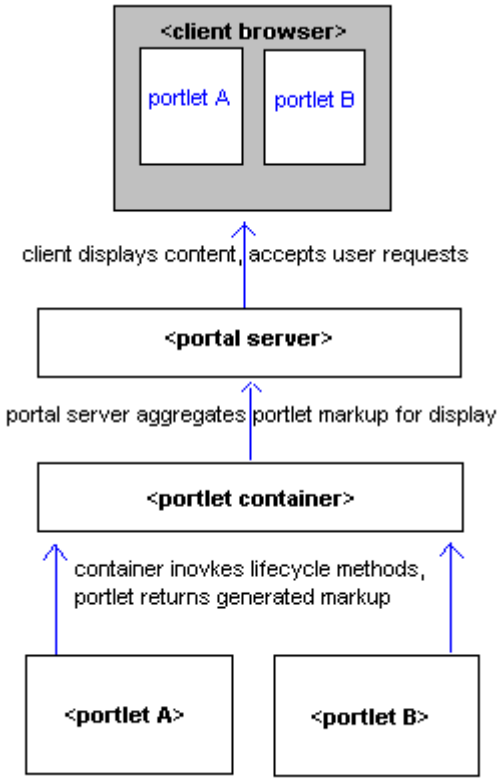

Abbildung 3: Architektur

1.2.3 Portlet Mode

Portlets haben 3 verschieden Modi View, Edit und Help. View ist der Standard Anzeige Modus, im Edit Mode kann man Präferenzen/Einstellungen editieren und im Help Mode wird eine Hilfe zum Portlet angezeigt. Natürlich ist es möglich alle Modi für Anzeige/Editieren/Hilfe zu verwenden aber wenn man dem JSR Standard hin arbeiten will, sollte man dies beachten.

Abbildung 4: WeatherPortlet im View-Mode

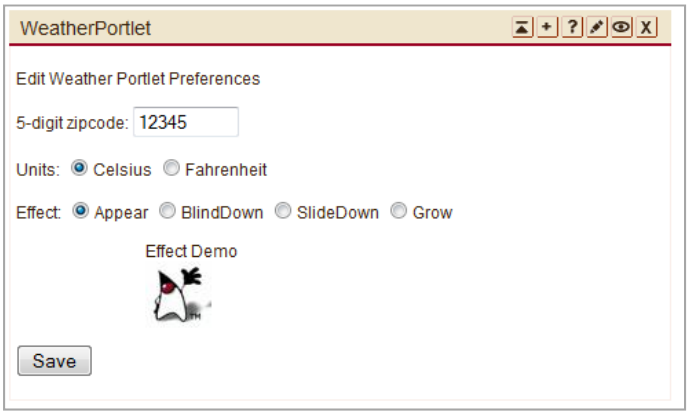

Abbildung 5: WeatherPortlet im Edit-Mode

Abbildung 6: WeatherPortlet im Help-Mode

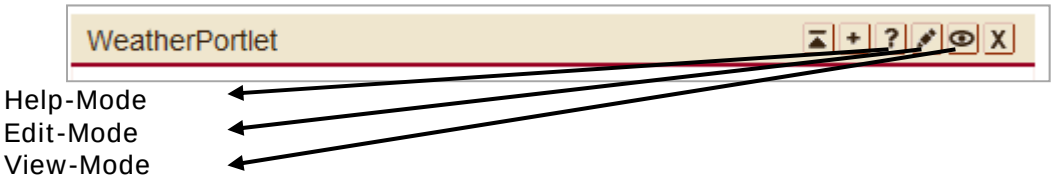

Abbildung 7: WeatherPortlet - zu den Modi

1.2.4 Window State

Mittels Windows-State kann man bestimmen wie viel Raum ein Portlet innerhalb eines Portals verbraucht und das Portlet kann damit auch entscheiden wie viel Information gerendert werden soll. Es gibt folgende 3 States: *minimized, maximized und standard.*

Abbildung 8: Window State

1.3 Entwicklungsumgebung

Als IDE wurde NetBeans 6.8 mit Portal Pack 3.0.3 Plugin verwendet. Ein Glassfish 3 Server mit OpenPortal Portlet-Container diente als WebSpace und Portlet-Container.
Achtung: sicherlich gibt es IDEs, Portal Server, Frameworks usw. die es einem leichter machen, einen Portlet zu entwickeln! (Verlinkungen, Resorucenzugriffe… etc) Diese sind dann aber meistens kostenpflichtig. Alle die hier verwendeten Technologien stehen kostenlos zur Verfügung.

1.4 Vorgang

1.4.1 Vorbereitung

Bevor man zum Entwickeln beginnt, muss der OpenPortal Portlet-Container ins IDE eingebunden werden. Dies kann man im Menü unter „Tool – Servers – Add Server“ machen .

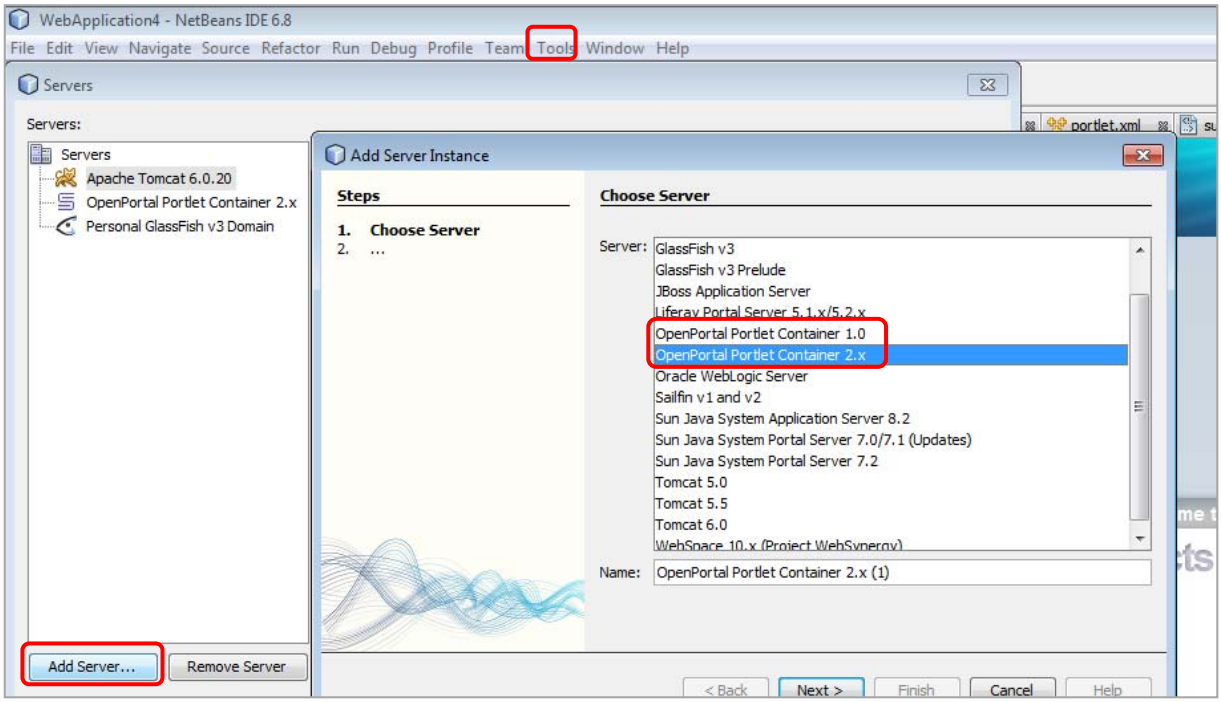

Abbildung 9: NetBeans - Portlet-Container hinzufügen

1.4.2 Webapplikation

Einschub: *Häufig werden die Begriffe Portlet und Portlet-Applikation synonym verwendet. Dies ist jedoch nicht korrekt. Ein Portlet bezeichnet eine Klasse, die das Portlet-Interface implementiert und somit als Portlet fungieren kann. Unter einer Portlet-Applikation wird das gesamte Webprojekt verstanden, inklusive Deployment-Deskriptoren und sonstigen Ressourcen. Eine Portlet-Applikation wird in der Regel als war-Datei deployt. Innerhalb einer Portlet-Applikation kann es durchaus mehrere Portlets geben, die alle in der portlet.xml definiert sein müssen.*
[http://it-republik.de/jaxenter/artikel/Portale-und-Portlets-%281%29-%96-Grundlagen-2076.html]

Zu Beginn muss man ein neues Projekt anlegen, genauer gesagt eine neue Webapplikation. Im Popup-Menü dazu muss man unter dem Punkt „Server Setting“ den vorher in die Entwicklungsumgebung eingebundenen OpenPortal PortletContainer auswählen und unter dem Punkt „Frameworks“ natürlich Portlet Support auswählen (siehe Abbildung 10 – 12).

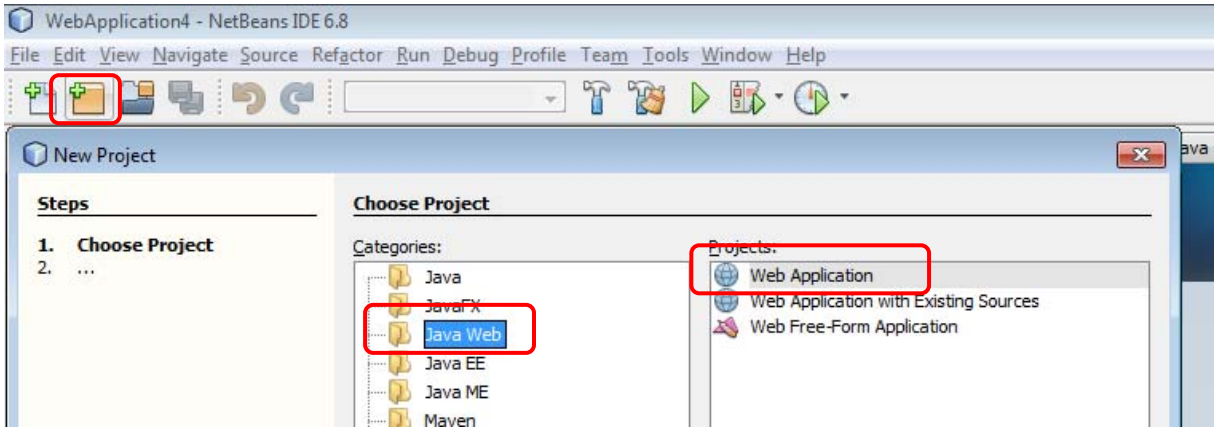

Abbildung 10: Neues Projekt

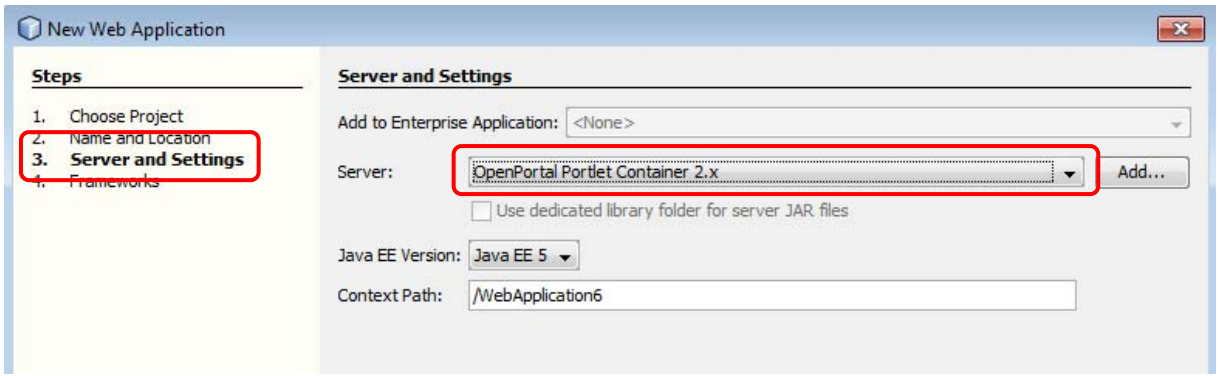

Abbildung 11: Neues Projekt

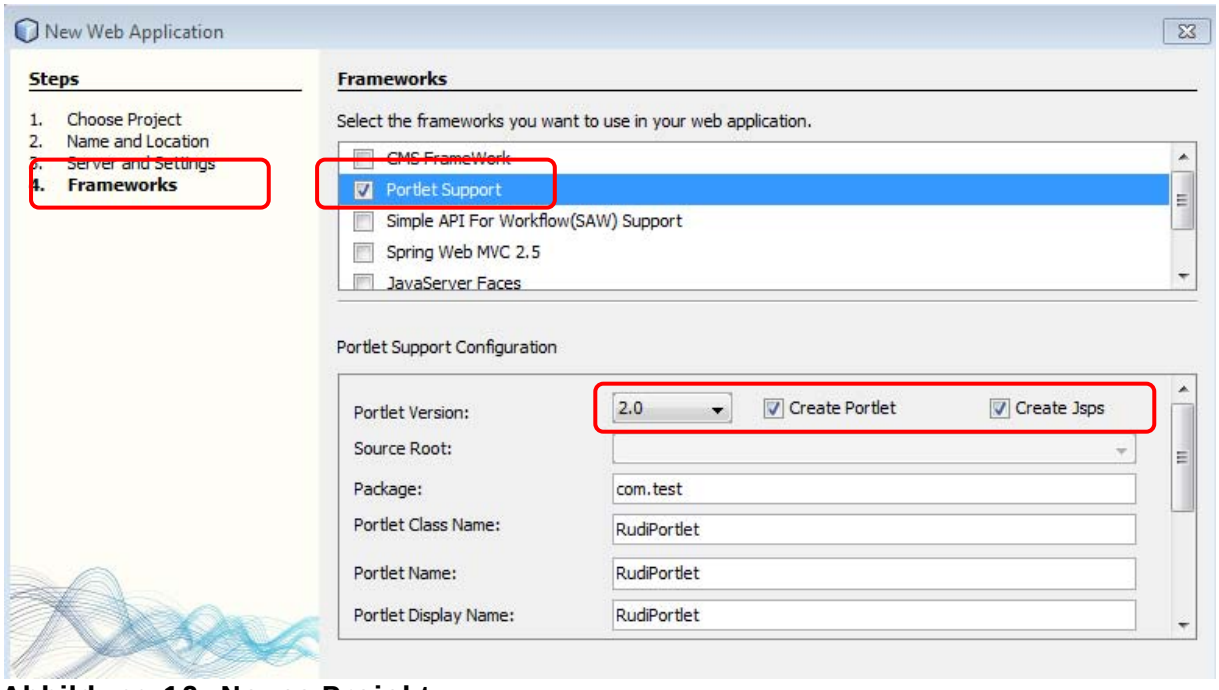

Abbildung 12: Neues Projekt

Die IDE erzeugt eine deploy bereite Web(bzw. Portlet)-Applikation, auf die „Hello World“ Art. Notwendige Bibliotheken sind schon eingebunden. 3 JSPs für die drei unterschiedlichen Modi sind schon vorhanden und die Configuration Files sind auch schon da.

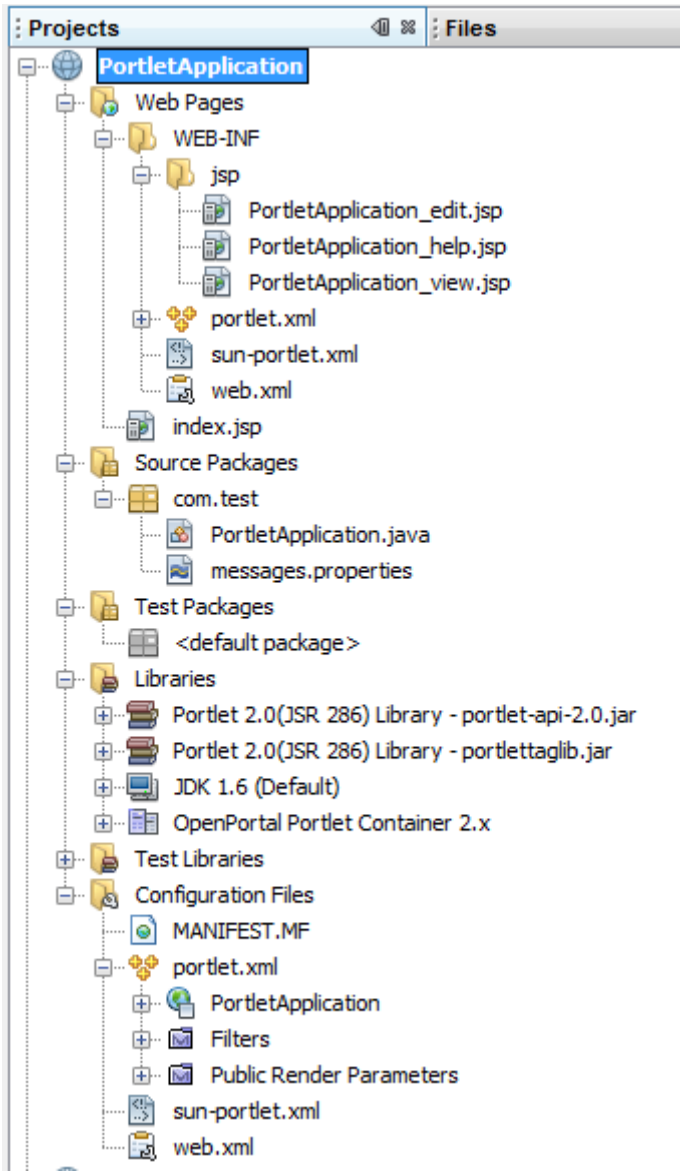

Abbildung 13: Struktur der Webapplikation

1.4.3 Build & Deploy

Das Builden läuft über den IDE => „Rechtsklick – Clean&Build (oder nur Build)“. Deployen kann man über die vom Portlet Container Driver dazu zur Verfügung gestellte Oberfläche (siehe Abbildung 14).

Abbildung 14: Porlet Container Driver - Admin Oberfläche

1.4.4 Page-Flow

Da ein Portlet (JSR 168) in viele Portale eingebettet werden kann , muss man bei Verlinkungen beachtet, dass man die dazu vorgesehene Bibliothek benützt. Es war leider nicht möglich direkt von JSP zu JSP zu verlinken. Daher wird im Beispiel ein RequestParameter definiert , die die URL beinhaltet. Auf diesen Parameter wird dann zugegriffen und auf die gewünschte URL weitergeleitet. (über den RequestDispatcher)

1.4.4.1 Beispiel:

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@ page import="javax.portlet.*"%>
<%@ taglib uri="http://java.sun.com/portlet_2_0"
prefix="portlet"%>

<portlet:defineObjects />
<%PortletPreferences prefs = renderRequest.getPreferences();%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Fachinfo1</h1>
        <p>
            seite 1

            <portlet:renderURL var="aURL">
                <portlet:param name="goto" value="/WEB-
INF/jsp/view2.jsp" />
            </portlet:renderURL>
            <a href="<%=aURL.toString()%>">zur Hilfe Seite
2</a>

            <br />
            a picture:
            <br />
            <img src="<%=
renderResponse.encodeURL(renderRequest.getContextPath() +
"/image/exampel.jpg") %>" alt="ein example Bild" />
        </p>
    </body>
</html>
```

Im Beispiel wird ein renderURL erzeugt und diese dem Link als HREF übergeben. Der Grund dafür ist, dass ja nur der Portlet nochmals gerendert werden soll und nicht das gesamte Portal. Wenn man auf den Link klickt, tritt ein Render Event auf, dadurch wird einer der RenderFunktionen doView(), doEdit() oder doHelp() aufgerufen (je nach Modi).
Im Beispiel wird nur der View Modus verwendet => es wird doView() aufgerufen.

```
public void doView(RenderRequest request,RenderResponse
response) throws PortletException,IOException {
        response.setContentType("text/html");

        gotoRenderAction = request.getParameter("goto");

        view = (gotoRenderAction == null ? "/WEB-
INF/jsp/view.jsp" : gotoRenderAction);
         PortletRequestDispatcher dispatcher =
```

```
getPortletContext().getRequestDispatcher(view);
         dispatcher.include(request, response);
    }
```

Der RequestParameter goto wird ausgelesen und der Variable gotoRenderAction zugewiesen. Danach wird überprüft ob diese null ist und je nach dem wird mit dem RequestDispatcher weitergeleitet.

1.4.5 Image

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@ page import="javax.portlet.*"%>
<%@ taglib uri="http://java.sun.com/portlet_2_0"
prefix="portlet"%>

<portlet:defineObjects />
<%PortletPreferences prefs = renderRequest.getPreferences();%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Fachinfo1</h1>
        <p>
            seite 1
            <portlet:renderURL var="aURL">
                <portlet:param name="goto" value="/WEB-
INF/jsp/view2.jsp" />
            </portlet:renderURL>
            <a href="<%=aURL.toString()%>">zur Hilfe Seite
2</a>
            <br />
            a picture:
            <br />

            <img src="<%=
renderResponse.encodeURL(renderRequest.getContextPath() +
"/image/example.jpg") %>" alt="ein example Bild" />

        </p>
    </body>
</html>
```

Falls man ein Image einbinden will muss dies über den ContextPath geschehen (siehe oben). Dieser zeigt, wenn man die Ordnerstruktur betrachet auf den „web“ Ordner (siehe Abbildung 15).

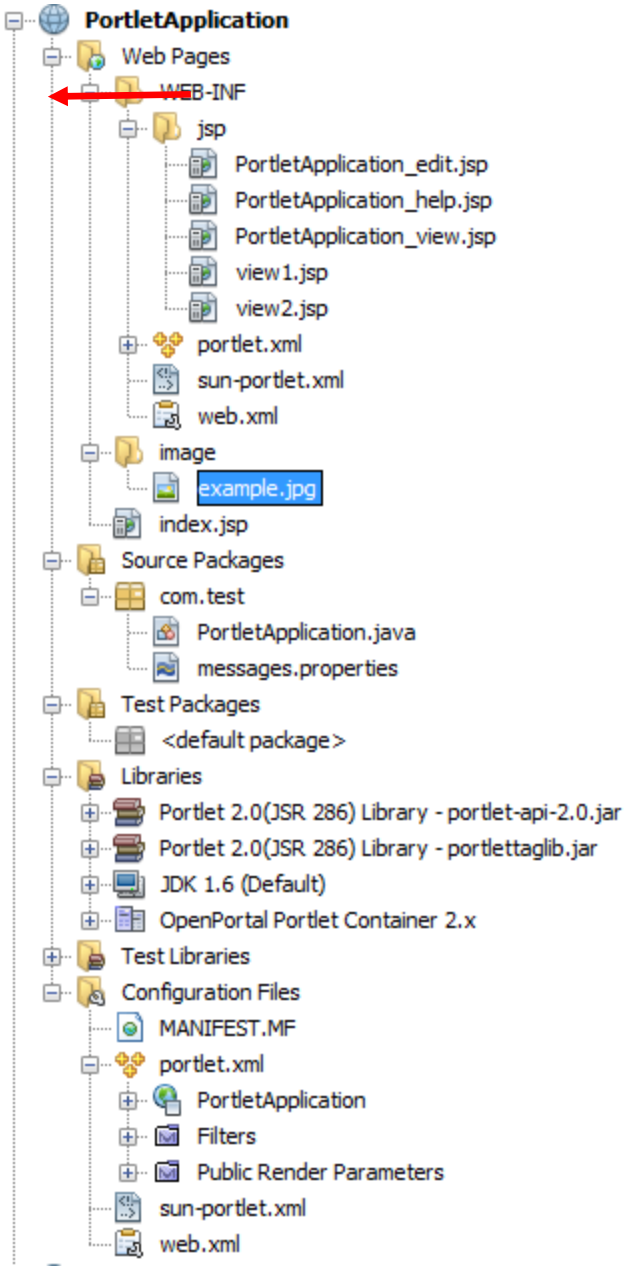

Abbildung 15: Ordnerstruktur

1.4.6 Iframe

Eine Alternative um eine Page-Flow zu bieten ist die gesamte „Hilfe-Seite“ mittels IFrame einzubinden.

```
<iframe src="<%=url%>"
        frameborder="<%=frameborder%>"
        height="<%=height%>"
        scrolling="<%=scrolling%>"
        width="<%=width%>">
</iframe>
```

2 Portlet Tag Library

[PLT.22 Portlet Tag Library ; Java™ Portlet Specification Seite 97]

Die Portlet Tag Library erlaubt es dem JSP, welche in einem Portlet inkludiert sind, Portlet spezifische Elemente zu verwenden, wie RenderRequest und RenderResponse. Außerdem stellt sie dem JSP Portlet Funktionalitäten, wie erstellen eines Portlet URLs, zur Verfügung.

JSP Seiten die diese Tag Library verwenden, müssen diese in einem „taglib“ deklarieren:

```
<%@ taglib uri=”http://java.sun.com/portlet” prefix=”portlet” %>
```

2.1 defineObjects Tag

Der defineObjects Tag legt folgende variablen im JSP fest:

- RenderRequest renderRequest
- RenderResponse renderResponse
- PortletConfig portletConfig

Der defineObjects Tag muss kein Attribut enthalten oder ein Inhalt im Body:

```
<portlet:defineObjects />
```

2.2 actionURL Tag

Dieser Tag erstellt eine URL, welche auf das aktuelle Portlet zeigt, und dort ein Action Request triggert und diese mit Parametern versorgt. Parameter können mit einem param Tag zwischen actionURL Start- und Endtag übergeben werden.

Beispiel:

```
<portlet:actionURL windowState=”maximized” portletMode=”edit”>
     <portlet:param name=”action” value=”editStocks”/>
</portlet:actionURL>
```

2.3 renderURL Tag

Dieser Tag erstellt eine URL, welche auf das aktuelle Portlet zeigt, und dort ein Render Request triggert und diese mit Parametern versorgt. Parameter können mit einem param Tag zwischen actionURL Start- und Endtag übergeben werden.

Beispiel:

```
<portlet:renderURL portletMode=”view” windowState=”normal”>
     <portlet:param name=”showQuote” value=”myCompany”/>
     <portlet:param                              name=”showQuote”
value=”someOtherCompany”/>
</portlet:renderURL>
```

2.4 param Tag

Dieser Tag definiert ein Parameter, welcher in einem actionURL oder renderURL verwendet werden kann. Sie muss nichts im Body beinhalten. Folgende Attribute müssen angegeben werden:
name (String) - Name des Parameter die in einem URL verwendet wird.
value (String) - Wert des Parameter die in einem URL verwendet wird.

Beispiel:
```
<portlet:param name=”myParam” value=”someValue”/>
```

3 Medien einbinden

Java Entwickler machen oft den Fehler und binden z.B. ein Bild (in die JSP) folgendermaßen ein:

```
<img src="/<%= request.getContextPath()%>/images/logo.gif"/>
```

Das ist aber nach den JSR-168 Standard inkorrekt, richtig wäre es:
```
<img src="<%=
renderResponse.encodeURL(renderRequest.getContextPath()+
"/images/logo.gif") %>"/>
```

Man sollte immer die URL Rewriting APIs verwenden.

Ein Pfad im nachhinein zu erweitern kann auch leicht zu Fehler führen, folgendes Beispiel demonstriert zuerst den korrekten und danach den inkorrekten Fall:
```
<@= renderResponse.encodeURL(renderRequest.getContextPath()+
"/images/logo.gif")@>
```

Dieser Code generiert folgendes HTML-Fragment:
```
<img
src="http://localhost:7001/PortalWebApp/images/logo.gif;PORTAL
_TAU=W3f6FbmLLcgZq9Fpv1JHLs5rrJG8Lgj2nnDVJqdfShhRGFnsqCKZ!-
545815275"/>
```

was korrekt ist. Hingegen dieser Code:
```
<@= renderResponse.encodeURL(renderRequest.getContextPath()+
"/images/")+"logo.gif"@>
```

generiert ein falsches HTML-Fragment:
```
<img
src="http://localhost:7001/PortalWebApp/images/logo.gif;PORTAL
_TAU=W3f6FbmLLcgZq9Fpv1JHLs5rrJG8Lgj2nnDVJqdfShhRGFnsqCKZ!-
545815275"/>
```

da dieser nicht mehr auf das File zeigt!

4 PageFlow

Eine Verlinkung von einer JSP-Seite zu einer anderen innerhalb eines Portlets muss über ein renderURL realisiert werden:

```
<portlet:renderURL var="aURL">
     <portlet:param name="goto" value="/WEB-INF/jsp/view2.jsp"
/>
</portlet:renderURL>
<a href="<%=aURL.toString()%>">zur Seite 2</a>
```

Im Beispiel wird ein renderURL erzeugt und diese dem Link als HREF übergeben. Der Grund dafür ist, dass ja nur der Portlet nochmals gerendert werden soll und nicht das gesamte Portal. Wenn man auf den Link klickt, tritt ein Render Event auf, dadurch wird einer der RenderFunktionen doView(), doEdit() oder doHelp() aufgerufen (je nach Modi).
Im Beispiel wird nur der View Modus verwendet => es wird doView() aufgerufen.

```
public    void    doView(RenderRequest    request,RenderResponse
response) throws PortletException,IOException {
        response.setContentType("text/html");

        gotoRenderAction = request.getParameter("goto");

        view    =    (gotoRenderAction    ==    null    ?    "/WEB-
INF/jsp/view.jsp" : gotoRenderAction);
         PortletRequestDispatcher dispatcher =

getPortletContext().getRequestDispatcher(view);
         dispatcher.include(request, response);
    }
```

Der RequestParameter goto wird ausgelesen und der Variable gotoRenderAction zugewiesen. Danach wird überprüft ob diese null ist und je nach dem wird mit dem RequestDispatcher weitergeleitet. Wichtig ist es auch, immer den ContentType zu setzen.

5 Rechtliche Hinweise / Impressum

5.1 Fragen und Anregungen

Sollten Sie weitergehende Fragen oder Anregungen zum Produkt haben, so wenden Sie sich bitte an den Support. Wir stehen Ihnen auch gerne für Fragen bezüglich unseres Reseller-Programms und Partner-Programms zur Verfügung. Zugriff auf die erweitere Online-Demo des hyper Content Management Servers können sie ebenfalls über den Support beantragen.

hyperCMS Support:
support@hypercms.com
http://www.hypercms.com

5.2 Impressum

Verantwortlich für den Inhalt:

hyperCMS
Content Management Solutions GmbH
Rembrandtstr. 35/6
A-1020 Wien – Austria

office@hypercms.com
http://www.hypercms.com

5.3 Rechtliche Hinweise

Vorliegendes Benutzerhandbuch basiert auf der zum Zeitpunkt der Verfassung des Dokumentes verfügbaren Programmversion.

Der Hersteller behält sich Programmänderungen und –Verbesserungen vor.

Fehler und Irrtümer vorbehalten.