



BEA WebLogic Portal™

Interportlet Communication Guide

Version 8.1 Service Pack 3
June, 2004

Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Contents

Using the Interportlet Communication Samples

Samples Description	1-1
Sample 1: Pageflow Portlet to Pageflow Portlet	1-2
How It Works	1-2
Sample 2: Pageflow Portlet to Non-Pageflow Portlet	1-3
How It Works	1-3
Sample 3: Non-Pageflow Portlet to Non-Pageflow Portlet	1-3
How It Works	1-3
Using Backing Files to Add Functionality.....	1-3
Sample 4: Non-Pageflow Portlet to Pageflow Portlet	1-4
How It Works	1-4
Obtaining the Samples	1-4
Download the Samples	1-4
Install the Samples	1-5
Location of Samples.....	1-5
How to Run the Samples	1-6
Open the Portal Application	1-6
Run the Samples	1-6
Using the Samples in Your Portal	1-8

Understanding Backing Files

What are Backing Files?	2-1
-------------------------------	-----

Which Controls Support Backing Files?	2-2
How Backing Files are Executed.	2-2
Other Execution Notes	2-3
Thread Safety with Backing Files	2-3
Creating a Backing File	2-4
Adding a Backing File to a Portlet	2-7

Using the Interportlet Communication Samples

Interportlet communications refers to how an event in one portlet controls some aspect of behavior in another portlet. This section describes four interportlet communication samples available from BEA WebLogic Portal's dev2dev site that show you how interportlet communications can be implemented. It includes information on the following subjects:

- [Samples Description](#)
- [Sample 1: Pageflow Portlet to Pageflow Portlet](#)
- [Sample 2: Pageflow Portlet to Non-Pageflow Portlet](#)
- [Sample 3: Non-Pageflow Portlet to Non-Pageflow Portlet](#)
- [Using Backing Files to Add Functionality](#)
- [Sample 4: Non-Pageflow Portlet to Pageflow Portlet](#)
- [Download the Samples](#)
- [How to Run the Samples](#)
- [Using the Samples in Your Portal](#)

Samples Description

Each sample is comprised of two portlets: a menu portlet and a display portlet, plus a portlet describing the logic behind the selected sample (see [Figure 1-1](#)). The menu portlet appears beneath the selection display portlet to emphasize the point that portlet display order does not

affect the behavior of the sample. The description portlet changes when you select a sample from the menu at the bottom of the portlet header (see [Figure 1-2](#)).

Figure 1-1 Interportlet Communications Sample Portal



Sample 1: Pageflow Portlet to Pageflow Portlet

Sample 1 shows interportlet communications between two pageflow portlets. In this example, both the menu portlet (`menuController.portlet`) and the menu selection display portlet (`selectionReaderController.portlet`) are pageflow portlets.

How It Works

The selection reader portlet listens to the menu portlet. This is achieved by setting the `Listen To` property of the selection reader portlet to the instance label of the menu portlet—in this case, `portlet_pfo_menu_instance`.

Warning: the Listen-To mechanism is only suitable for Single-file portals and not streaming portals.

When the `selectMenuItem` action of the menu's page flow is executed, the same action, if it exists, will subsequently be executed in the selection display page flow. In this simple example the listening page flow sets a pageflow data member that is then bound to the JSP via an invocation of the `netui:label` tag.

Sample 2: Pageflow Portlet to Non-Pageflow Portlet

In this example, the menu portlet (`menuController.portlet`) is a pageflow portlet, but the menu selection display portlet (`selectionReader.portlet`) is not.

How It Works

The `selectMenuItem` action in the menu portlet pageflow places the value of the selected menu item in the outer request, accessible by the selection display portlet. Remember, the `getRequest()` method in a pageflow returns a request scoped to that pageflow. To share the value of the selected item with the non-pageflow portlet, the `selectMenuItem` first accesses the outer request using `ScopedServletUtils.getOuterRequest(scopedRequest)`, and then sets an attribute—`selectedItem`—on this request. The selection display portlet simply retrieves this attribute from the request in its `.jsp`.

Sample 3: Non-Pageflow Portlet to Non-Pageflow Portlet

In this example, neither the menu portlet (`menu.portlet`) nor the menu selection display portlet (`selectionReader.portlet`) are pageflow portlets.

How It Works

The menu portlet uses the portal framework's render taglib to create three anchors, each of which, when clicked, will place a menu selection value into the request, and return to the same `.jsp` (`menu.jsp`) within the portlet. The selection reader `.jsp` grabs the menu selection parameter from the request, and displays it, using some scriptlet.

Using Backing Files to Add Functionality

You should be aware that the first three samples are very simple examples of interportlet communication. You can use backing files in these portlets so that “real work” can be accomplished within the `init()`, `preRender()`, `handlePostback()`, and other portal

framework lifecycle methods. Note that “[Sample 4: Non-Pageflow Portlet to Pageflow Portlet](#),” employs a backing file. For more information on backing files, please refer to [Understanding Backing Files](#).

Sample 4: Non-Pageflow Portlet to Pageflow Portlet

In this example, the menu portlet (`menu.portlet`) is not a pageflow portlet, but the menu selection display portlet (`selectionReaderController.portlet`) is a pageflow portlet.

How It Works

The menu portlet exploits a backing file, `MenuBacking.java`, to explicitly run a particular action in the selection display pageflow portlet. After this action is run, the backing file of the menu portlet explicitly changes the URI of the selection display portlet—to `one.jsp`, `two.jsp`, or `three.jsp`—based on the result of the executed action. The primary steps in accomplishing this form of inter-portlet communication are:

- Retrieve/create a scoped request and response based on the instance label of the pageflow portlet that is to be manipulated based on menu selection.
- Using `PageFlowUtils` and the scoped request, retrieve the appropriate `ActionResolver` for the selection display portlet.
- Set the `<selectedItem>` attribute on the scoped request, so that when the `ActionResolver` runs the pageflow's action, this information will be available to the pageflow.
- Execute the appropriate pageflow action and receive an `ActionResult` object.

Use the URI in the `ActionResult` object to set the current content URI for the pageflow portlet.

Refer to the comments in `MenuBacking.java` for more explicit details. You can find `MenuBacking.java` and other information on backing files in [Understanding Backing Files](#).

Obtaining the Samples

This section describes how to download and install the interportlet communication code samples.

Download the Samples

The samples are available from the BEA WebLogic Portal 8.1 dev2dev site, at:

http://dev2dev.beasys.com/codelibrary/code/interportlet_v1.jsp

Simply click the Interportlet Communication Samples link and follow the download process to obtain the file `ipc_samples.zip`. This zip file contains the directory `interportlets_codes`, which is comprised of the subdirectory `portletToPortlet` and the file `ipc.portal`.

Install the Samples

You can install the samples directly from the download directory or by using BEA WebLogic Workshop.

To install from the download directory:

Using a file extraction tool, such as WinZip, extract both `portletToPortlet` and `ipc.portal` into a web project, such as `sampleportal` (`<WebLogic_Home>/samples/portalApp/sampleportal`).

To install from BEA WebLogic Workshop:

1. Right-click the web project into which you want to install the samples and select Import...
2. Navigate to the download directory and select both `portletToPortlet` and `ipc.portal` (under the `interportlet_codes` directory) and click Import.

Location of Samples

[Table 1-1](#) lists the location of each sample within the directory you choose to store it (for example, `<WebLogic_Home>/portalApp/sampleportal`).

Table 1-1 Location of Interportlet Communication Samples

Sample	Location
Pageflow Portlet to Pageflow Portlet	<code>portletToPortlet/pageFlowsOnly</code>
Pageflow Portlet to Non-Pageflow Portlet	<code>portletToPortlet/pageFlowMenuOnly</code>
Non-Pageflow Portlet to Non-Pageflow Portlet	<code>portletToPortlet/noPageFlows</code>
Non-Pageflow Portlet to Pageflow Portlet	<code>portletToPortlet/pageFlowSelectionDisplayOnly</code>

How to Run the Samples

Since all four samples appear in the same portal, you only need to open that portal to run them all.

Open the Portal Application

You can open the sample either from within an open browser or by using BEA WebLogic Workshop.

To open from a browser:

With BEA WebLogic Server running, enter the URL for `ipc.portal` in the browser address bar; for example, `http://localhost:7001/myWebApp/ipc.portal` (where *myWebApp* is the web application into which you installed `ipc.portal`).

To open from BEA WebLogic Workshop:

1. In the Application tree, locate and double-click to open the file `ipc.portal`.

The portal opens in WebLogic Workshop.

2. If WebLogic Server is not running, open the Tools menu and select WebLogic Server>Start WebLogic Server.

The server startup routine begins; after a few moments, the server will be started and a green dot will appear at the bottom of the IDE, next to the words “Server Running.”

3. Open the Portal menu and select Open Current Portal.

A browser opens with the interportlet communication sample portal displayed.

Run the Samples

To run the samples, click the sample you want to see from the menu below the portal header, as shown in [Figure 1-2](#).

Figure 1-2 Interportlet Communication Sample; Portal Header with Menu

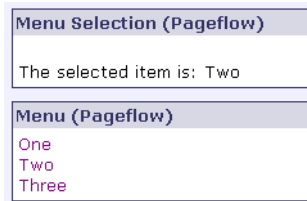
That sample appears in the body of the portal; for example, if you selected sample 1, Pageflow on Menu and on Selection Display, the portal would appear as in [Figure 1-3](#)

Figure 1-3 Interportlet Communication Sample; Sample 1: Pageflows on Menu and Menu Selection Portlets

<div>Menu Selection (Pageflow)</div> <div>The selected item is: none</div> <div>Menu (Pageflow)</div> <div>One</div> <div>Two</div> <div>Three</div>	<div>How I Work</div> <div>Pageflow on Menu and on Menu Selection Display -- How I Work</div> <div>Project Directory: <code>portletToPortlet/pageFlowsOnly</code></div> <div>In this example, both the menu portlet (<code>menuController.portlet</code>) and the menu selection display portlet (<code>selectionReaderController.portlet</code>) are pageflow portlets. The menu portlet was placed beneath the selection display portlet to emphasize the point that portlet display order does not affect the behavior of the sample.</div> <div>The selection reader portlet listens to the menu portlet. This is achieved by setting the "Listen To" property of the selection reader portlet to the instance label of the menu portlet -- in this case, <code>"portlet_pfo_menu_instance."</code></div> <div>When the "selectMenuItem" action of the menu's page flow is executed, the same action, if it exists, will subsequently be executed in the selection display page flow. In this simple example the listening page flow sets a pageflow data member that is then bound to the jsp via an invocation of the <code>netui:label</code> tag.</div>
--	---

To observe interportlet communication with this sample, simply select one of the options in the Menu portlet and view the result in the Menu Selection portlet; for example, if you select Two, the Menu Selection portlet text changes to The selected item is: Two ([Figure 1-4](#)).

Figure 1-4 Menu and Menu Selection Portlets; Item Two Selected



Select additional samples and test them, observing the interplay between the Menu and the Menu Selection portlets.

Using the Samples in Your Portal

You can easily use these samples in your own applications. You will need to do the following:

1. Copy the portlet(s) you want to use from the `portalApp/sampleportal/portletToPortlet` into your project directory.
2. In each sample you want to use, change the instance and action name of the portlet to match your application needs.
3. Update the necessary variable in the request parameters to match your application needs.

Understanding Backing Files

In [Sample 4: Non-Pageflow Portlet to Pageflow Portlet](#) sample shown in [Chapter 1](#), “Using the Interportlet Communication Samples”, a “backing file”—`menuBacking.java`—is used.

Backing files allow you to programatically add functionality to a portlet by implementing (or extending) a Java class, which enables preprocessing (for example, authentication) prior to rendering the portal controls. Backing files can be attached to portals either by using WebLogic Workshop or coding them directly into a `.portlet` file.

This section is primer on backing files. It includes information on the following subjects:

- [What are Backing Files?](#)
- [Which Controls Support Backing Files?](#)
- [How Backing Files are Executed](#)
- [Thread Safety with Backing Files](#)
- [Creating a Backing File](#)
- [Adding a Backing File to a Portlet](#)

What are Backing Files?

Backing files are simple Java classes that implement the `com.bea.netuix.servlets.controls.content.backing.JspBacking` interface or extend the `com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking` interface abstract class. The methods on the interface mimic the controls lifecycle methods

(see “[How Backing Files are Executed](#)”) and are invoked at the same time the controls lifecycle methods are invoked.

Which Controls Support Backing Files?

At this time, the following controls support backing files:

- Desktops
- Books
- Pages
- Portlets

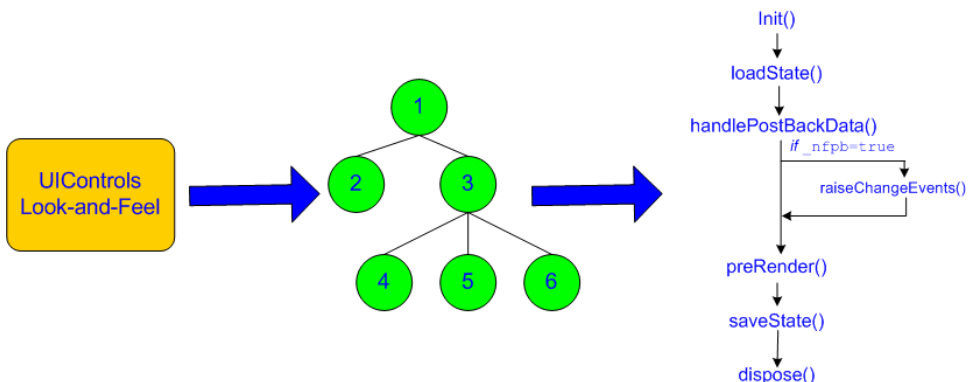
How Backing Files are Executed

All backing files are executed before and after the JSP is called. In its lifecycle, each backing file calls these methods:

- `init()`
- `handlePostBackData()`
 - `raiseChangeEvents()`
- `preRender()`
- `dispose()`

[Figure 2-1](#) illustrates the lifecycle of a backing file.

Figure 2-1 Backing File Lifecycle



On every request, the following occurs:

1. All `init()` methods are called on all backing files on an “in order” basis (that is, in the order they appear in the tree). This method gets called whether or not the control (that is, portlet, page, book, or desktop) is on an active page.
2. Next, if the operation is a postback *and* the control (a portlet, page, or book) is on a visible page, all `handlePostBackData()` methods are called. In other words if portlet is on a page but its parent page is not active, then this method will not get called.
 - If `_nfpb="true"` is set in the request parameter of any `handlePostBackData()` methods called, `raiseChangeEvents()` is called. This method causes events to fire.
3. Next, all `preRender()` methods are called for all controls on an active (visible) page.
4. Next, the JSPs get called and are rendered on the active page by the `<render:beginRender>` JSP tag. Rendering is stopped with the `<render:endRender>` tag.
5. Finally, the `dispose()` method gets called on the backing file.

Note: `roadstead()` and `savviest()`, shown in [Figure 2-1](#) are part of the control lifecycle, not the backing file lifecycle.

Other Execution Notes

If the backing file is part of a floated portlet, when that portlet is floated, only *its* contents are executed.

If a book is embedded within a portlet, then the book would get called; however, if the book is the parent of the portlet then it would not get called as it is not contained within the portlet.

Thread Safety with Backing Files

A new instance of a backing file is created per request, so you don't have to worry about thread safety issues. New Java VMs are specially tuned for short-lived objects, so this is not the performance issues it once was in the past. Also, `JspContent` controls support a special type of backing file that allows you to specify whether or not the backing file is thread safe. If this value is set to `true`, only one instance of the backing file is created and shared across all requests.

Creating a Backing File

As previously discussed, a backing file must be an implementation of `com.bea.netuix.servlets.controls.content.backing.JspBacking` interface or an extension of the `com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking` abstract class. You only need to modify these files as necessary to implement the backing functionality you desire.

Listing 2-1 is the backing file used in the “Sample 4: Non-Pageflow Portlet to Pageflow Portlet” example in Chapter 1, “Using the Interportlet Communication Samples”. In this example, the `AbstractJspBacking` class is extended to provide the backing functionality required by the portlet.

Listing 2-1 Backing File Example

```
package portletToPortlet.pageFlowSelectionDisplayOnly.menu.backing;

import com.bea.netuix.nf.UIControl;
import com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking;
import com.bea.netuix.servlets.controls.page.PageBackingContext;
import com.bea.netuix.servlets.controls.portlet.backing.PortletBackingContext;
import com.bea.netuix.servlets.controls.window.WindowMode;
import com.bea.p13n.management.ApplicationHelper;
import com.bea.wlw.netui.pageflow.ActionResolver;
import com.bea.wlw.netui.pageflow.ActionResult;
import com.bea.wlw.netui.pageflow.PageFlowUtils;
import com.bea.wlw.netui.pageflow.scoping.ScopedRequest;
import com.bea.wlw.netui.pageflow.scoping.ScopedResponse;
import com.bea.wlw.netui.pageflow.scoping.ScopedServletUtils;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.List;
import java.util.ListIterator;

public class MenuBacking extends AbstractJspBacking
{

    public boolean preRender(HttpServletRequest request, HttpServletResponse
        response)
    {
        talkToPageFlowPortlet(request, response);
    }
}
```

```

        return true;
    }

    private void talkToPageFlowPortlet(HttpServletRequest request,
        HttpServletResponse response)
    {
        //First, check to see if we should even do anything.
        if ( request.getParameter("selectedItem") == null )
            return;

        //The servlet context is necessary for getting the scoped request.
        ServletContext context = ApplicationHelper.getServletContext(request);

        //Get the scoped request associated with the pageflow portlet we want to
        //talk to.
        ScopedRequest sRequest =
            ScopedServletUtils.getScopedRequest(request,
                "",
                context,
                "portlet_pfdo_display_
                    instance");

        //Likewise, get the scoped response.
        ScopedResponse sResponse =
            ScopedServletUtils.getScopedResponse(response,
                sRequest);

        //Using the pageflow utilities, get an ActionResolver. This object can have
        //the pageflow execute certain actions.
        ActionResolver resolver = PageFlowUtils.getCurrentActionResolver(sRequest);

        try
        {
            //Set an attribute in the scoped request that the page flow can use.
            sRequest.setAttribute("selectedItem",
                request.getParameter("selectedItem"));

            //Execute the "readSelection" action in the pageflow. The ActionResult
            //object that is returned contains a URI that can be used to update the
            //pageflow portlet's content. Although this method is called "lookup,"
            //the action is actually executed.
            ActionResult ar = resolver.lookup("readSelection",
                context,
                sRequest,
                sResponse,
                "portletToPortlet.pageFlowSelectionDisplayOnly.
                    selectionReader.
                    selectionReaderController");

```

Understanding Backing Files

```
//Now, the final step: Find the control for the page portlet, using
//its label, and update the current content uri for the pageflow
//portlet. To do this, we need to search threw all the window contexts
//until we find the portlet we're looking for.
PageBackingContext pbc =
    PageBackingContext.getPageBackingContext(request);
ListIterator backingContexts =
    pbc.getWindowBackingContexts().listIterator();

while ( backingContexts.hasNext() )
{
    Object nextContext = backingContexts.next();
    if ( nextContext instanceof PortletBackingContext )
    {
        PortletBackingContext portletBC =
            PortletBackingContext(nextContext);
        if ( "portlet_pfdo_display_instance".
            equals(portletBC.getLabel()) )
        {
            //Found it! Update the portlet with the uri from the
            //Action Result.
            WindowMode wm = portletBC.getWindowMode();
            wm.setCurrentContentUri(ar.getURI());
            break;
        }
    }
}

catch ( Exception e )
{
    e.printStackTrace();
}

}
```

You should follow these guidelines when creating a backing file:

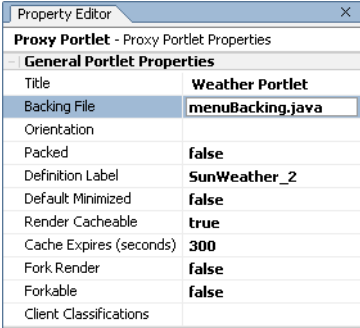
- Ensure `netuix_servlet.jar` is included in the in the project classpath, otherwise compilation errors will occur.
- When implementing the `init()` method, avoid any heavy processing.

Adding a Backing File to a Portlet

You can add a backing file to a portlet either from within WebLogic Workshop or by coding it directly into the file to which you are attaching it. You cannot use the IDE to attach a backing file to a Java Page Flow portlet or to a Struts portlet. Instead, you will need to physically code it into the `.portlet` file, as described in [Listing 2-2](#).

For all other portlet types, simply specify the backing file in the Backing File field under the General Properties section of the Property Editor, as shown in [Figure 2-2](#).

Figure 2-2 Adding a Backing File by Using the IDE



Property Editor	
Proxy Portlet - Proxy Portlet Properties	
General Portlet Properties	
Title	Weather Portlet
Backing File	menuBacking.java
Orientation	
Packed	false
Definition Label	SunWeather_2
Default Minimized	false
Render Cacheable	true
Cache Expires (seconds)	300
Fork Render	false
Forkable	false
Client Classifications	

To add the backing file by coding it into a `.portlet` file, as required for Java Page Flow portlets and Struts portlets, use the `backingFile` parameter within the `<netuix:jspContent>` element, as shown in [Listing 2-2](#).

Listing 2-2 Adding a Backing File to a .portlet File

```
<netuix:content>
  <netuix:jspContent
    backingFile="portletToPortlet.pageFlowSelectionDisplayOnly.menu.
      backing.MenuBacking"
    contentUri="/portletToPortlet/pageFlowSelectionDisplayOnly/menu/
      menu.jsp"/>
  </netuix:content>
```

Understanding Backing Files