This tutorial explains both the benefits and the methodology for integrating a Domino application into WebSphere Portal utilizing the Domino custom JSP tags. It provides an overview of the JSP technology and outlines the advantages, as well as the limitations of using the Domino custom JSP tag libraries. Finally, it explains both the theoretical background and covers step-by-step instructions on how to build a portlet application that accesses a Domino Journal database leveraging the Lotus Domino Toolkit for WebSphere Studio.

Section 1. Before you start

About this tutorial

Integrating an existing Lotus Domino application into WebSphere Portal can be a challenging task. This tutorial discusses one of the integration options, using the Domino custom JSP tag libraries.

The first half of this tutorial discusses the need for integrating existing Domino infrastructure into the WebSphere Portal environment. It also introduces the Domino
custom JSP tags as one option for integrating a Lotus Domino application into WebSphere Portal and discusses its advantages, as well as its limitations. The second half covers a step-by-step description of how to develop an example portlet application that utilizes the Domino custom JSP tag libraries.

The topics covered in this tutorial include:

- Reasons for integrating an existing Domino infrastructure into WebSphere Portal
- Domino custom JSP tag libraries as one option for integration
- Overview about J2EE and JSP technology
- Building of an example portlet application using the Domino custom JSP tag libraries

Prerequisites

This tutorial is recommended for Domino developers who seek to be rapidly productive in enhancing their Web applications by using J2EE technologies. It is also suited for J2EE developers who want to incorporate Domino capabilities without advanced knowledge of the Domino object model. Both developer groups can extend their knowledge about portalizing a Lotus Domino application by using the Domino custom JSP tag libraries.

Before taking this tutorial it is recommended that you are familiar with the following IBM software products:

- IBM WebSphere Studio Application Developer
- Lotus Domino Designer

System requirements

To complete the steps as demonstrated in this tutorial you will need the following IBM software products:

- IBM WebSphere Studio Application Developer version 5.1.2 or higher.
- Lotus Domino Designer version 6.5.2
- Lotus Domino Server version 6.5.1 or higher
Make sure you install the following plug-ins after WebSphere Studio has been successfully installed.

- **IBM Portal Toolkit** version 5.0.2.2
- **Lotus Domino Toolkit for WebSphere Studio** version 1.3
  The Lotus Domino Toolkit for WebSphere Studio is included with Lotus Domino Designer.

Download the source code used for the sample application: DominoJspJournal.zip or DominoJspJournal.war.

---

### Section 2. Introduction

**Why integrate Domino into WebSphere Portal?**

The history of Domino reveals why Domino integration into a portal environment is so significant. Over the years, many organizations have utilized Domino (and continue to use it) for their internal application development. The spectrum of these applications range from document stores to complex workflow applications and often contain critical data and business processes. Developing a Domino Web application that utilizes these existing applications can be challenging. Some of the issues to deal with often include both scaling and performance issues for Domino HTTP access. Additionally, delivery of personalized content increases the number of requirements for an implementation. This makes it difficult to rely solely on the Domino infrastructure to provide a Web-enabled work place.

WebSphere Portal is designed to integrate the entire software portfolio offered by IBM including an organization's Lotus infrastructure. Relying on open standards WebSphere Portal provides a comprehensive framework that enables the integration of Lotus Domino assets with a single, secure portal infrastructure. More importantly, it also utilizes advanced collaborative capabilities of Lotus to achieve high levels of productivity. WebSphere Portal provides flexible, global and personalized data presentation based on profiles and business rules and helps to support a global operating organization.

When you consider the strong collaboration and document-based capabilities of Lotus Domino, combined with the features of a WebSphere Portal environment, it is
easy to understand the potential benefits and the demand for a combination of both. During the following pages you will be introduced to the Domino custom JSP tags and how they can help you to integrate your Domino application into WebSphere Portal.

Section 3. Integration options

General-purpose portlets

There are several ways to integrate your Domino application within WebSphere Portal. The first option is to use general-purpose portlets that are shipped with WebSphere Portal or are available from third-party sources. These portlets provide the possibility to access your Domino application. The usage of an existing portlet requires the least amount of development time and skill, but may be limited in terms of functional capabilities and customization.

Java API for LCC and Domino

Another method (and by far the most advanced option) for integrating your Domino application into Portal, is to utilize the Java API for Lotus Collaborative Components and Domino. Using Java provides the best way to overcome certain integration limitations, such as session management, customization and object pooling. Keep in mind however, this approach also requires extensive Java knowledge and increases the development time in most cases.

Domino custom JSP tag libraries

An intermediate solution is the utilization of the Domino custom JSP tag libraries. These tag libraries allow WebSphere Portal to access the rich set of Domino collaborative capabilities. They provide a quick and easy technique to incorporate complex Lotus Domino applications by adding Domino custom JSP tags to a JSP page. While there may be performance limitations for heavily used portal application, the tag libraries allow for quickly and easily exposing Domino data and functionality in a portlet. The tag libraries are primarily designed to attract Domino developers who seek to be rapidly productive in enhancing their Web applications by using J2EE technologies. Additionally, they also allow J2EE developers to incorporate
Domino capabilities without advanced knowledge of the Domino object model.

Integrating Lotus Domino applications with JSP tag libraries therefore yields shorter development time than that for a custom Java development option. It is sufficient to extend your current Domino application by transactional and personalization services. Additionally, this approach may be helpful to relieve some of the load which Lotus Domino would otherwise handle as a Web application server. Keep in mind that the underlying technology has certain performance-related limitations, so the option for using Domino custom JSP tag libraries is probably the one to choose if the functionality of general-purpose portlets is not enough to fulfill the integration needs, and if there is an underlying motivation to understand how Java technologies can help to leverage your Domino application within WebSphere Portal.

Section 4. J2EE and JSP overview

J2EE platform

Java 2 platform, Enterprise Edition (J2EE) is a comprehensive set of synchronized specifications for designing, developing, deploying, and managing multi-tier distributed server-based applications. J2EE describes the runtime environment for a J2EE application. This environment includes application components, containers, and resource manager drivers. The elements of this environment communicate with a specified set of standard services. The modular and standard-based architecture therefore lowers development risks and handles many application details automatically.

J2EE is based on containers that provide clear separation of business components and enterprise services. This enables the developer to focus on business logic rather than developing infrastructure code, e.g. Java Servlets and JavaServer Pages (JSP). It also makes it possible to simplify Web development by providing a flexible infrastructure for component communication and session management within a Web application that is integrated with the Web server. By representing a set of technologies that many vendors can implement and extend, J2EE and its standards guarantee interoperability and compatibility among the J2EE application vendors. This also ensures the portability of J2EE applications and components. J2EE enables applications to connect to external applications through technologies and standards that let you easily expose business logic to Web applications. J2EE is capable of delivering data to a set of different clients offering the basis of a
multi-channel e-business infrastructure.

Separation of tiers

The main advantage that J2EE offers from a design point of view is the separation of tiers that compose an application. While data, business logic, and presentation objects within a Domino application are held within a single file, J2EE builds on several containers representing different logical tiers.

Client tier:
This tier contains the end-user clients, which interact with the services provided by server-side components. Clients can include mark-up language pages and applets running in browsers, Java applications, and other applications that are interoperable with Java.

Middle tier:
This tier consists of the Web container and the EJB container. Both are J2EE runtime components. The EJB container holds the business logic and is designed to provide seamless access to relational databases and enterprise information systems with transactional integrity. Along with the data storage and access facilities, it is possible to embed the business logic of the application within the components of the EJB container since they offer high performance and availability mechanisms. The Web container is designed to construct presentation components and to control the presentation logic. It is based on technologies that offer high performance, extendible presentation framework, e.g. Java Servlets and JavaServer Pages. The
container is also a host to other technologies, such as Web Services and XML-XLST applications.

**Enterprise information system tier:**
This tier contains resources such as databases and other information systems. Its is considered to be the provider of data access and storage.

The Web container and the EJB container are the main blocks that construct a J2EE application server. The WebSphere Portal allows the construction of Portlet applications that reside on the Web container, so that the focus remains on this container. This is important as you discover the options the Custom JSP tag libraries have to offer for integrating a Domino application into WebSphere Portal.

The portlet components extend the Servlet component and can present its information using JSP pages. Because you are going to add the Domino custom tags into these JSP pages, it is necessary to explore the JSP technology in more detail.

**JavaServer Pages**

A JavaServer Page is a single component of the J2EE Web container. It is very similar to an HTML page, but provides the possibility to include and display dynamic content within Web pages. JSP technology was developed to separate the development of dynamic Web content from static Web page design. The result of this is that the presentation design can be altered, without the need to change the underlying dynamic page content. This is useful in the development life-cycle, because the Web page designers do not have to know how to create the dynamic content. Instead, they simply have to know where to place the dynamic content within the page.

To facilitate embedding of dynamic content, JSPs use a number of tags that enable the page designer to insert the properties of a JavaBean object and Enterprise JavaBeans. Therefore JSPs leverage the advantages offered by these technologies, e.g. the inherent reusability. Also JavaScript is supported by JSPs and can be used to add page-level functionality to the JSP. Combined with the portability and platform independence, these features describe the advantages of the JSP technology over most of the other options for creating dynamic Web content.

**Invoking a JavaServer Page**

A JSP file is stored on the file system in text format. On the first invocation of the file, i.e. the first HTTP request to the page or the first request of the page after changing
the underlying JSP file, the contents of the file are parsed by the JSP engine and translated into servlet source code by the application server. The generated servlet is responsible for rendering the static contents of the JSP file at design time, as well as the dynamic components of the Web page. The source code is compiled into a servlet class file. The servlet is instantiated and executed. The rendered content specified by the original JSP file is returned through the output stream of the servlet response object.

Any subsequent request to the page will invoke the generated servlet to serve the content to the client. The servlet remains in service until the application server is stopped, the underlying JSP file is altered causing a recompilation, or the servlet is manually unloaded.

Section 5. Domino custom JSP tag libraries

Overview

A JSP tag library is a collection of custom tags that encapsulate Java code through the use of tags. The library defines a declarative and modular functionality that can be reused by any JSP file. The tags are defined in an XML file called the Tag Library Descriptor file or TLD. It is used during the parsing and compiling of the JSP and defines which Java classes and methods should be interpreted. The collection of tags provides the possibility to include Java content to the JSP file, without getting
into the details of the Java code.

In the case of the Domino custom JSP tags, these can be used on your JSP file to include Lotus Domino elements. They enable either a user without Java knowledge to incorporate Java content or to allow a user without knowledge about Lotus Domino back-end objects to access Lotus Domino objects within the JSP file. Therefore the developer is able to extend his Web application to leverage the complex Lotus Domino interactions by utilizing tags. Since WebSphere Portal is based on the J2EE architecture, it allows the inclusion of JSP files, which can be used to expose Lotus Domino application within the WebSphere Portal infrastructure. The Lotus Domino tag libraries are:

- domtags.tld
- domutil.tld

The first TLD comprises collaboration tags for accessing standard, back-end objects in the Domino data repository. The second consists of utility tags for performing tasks that are common to all J2EE Web containers. The Domino JSP tags can be divided into four groups:

- Data access tags
- Data input tags
- Process control tags
- Utility tags

**Domino custom JSP tags**

The JSP tags include:

**Data access tags:** Data access tags provide the possibility to gain access to the most important objects in the Lotus Domino object hierarchy.

**Data input tags:** These tags allow the input of information from a JSP file to the Lotus Domino application.

**Process control tags:** These tags are used to obtain the state or the properties of the Lotus Domino application. They allow the JSP to modify the presentation of the portlet based on the result of the queried state or property.

**Utility tags:** The utility tags allow controlling the flow of the presentation logic on the
portlet based on Lotus Domino conditions and expressions.

For further descriptions of the tags, refer to the help files that can be accessed in the WebSphere Application Studio Developer help. You'll see a reference of the tags, including examples how to use the most common tags.

Domino custom JSP tags

There are a few issues you should consider when using the Domino custom JSP tags to incorporate Lotus Domino applications into the WebSphere Portal infrastructure:

- The use of the `domino:attachment` tag is not supported in the WebSphere Portal environment. The applet or control running in the browser is not able to perform a validation handshake mechanism after a file is uploaded. This is caused by the way the portal creates its responses.

- To expose Rich Text in WebSphere Portal, turn off the cross site scripting defense of the Portal by editing the ConfigService.properties file located in the `<wp_root>/shared/app/config/services` directory. Set `security.css` to `false`.

- Use the duration attribute of the `domino:session` tag in order to increase the performance of your application. There exists a DominoSessionManager that caches existing sessions. To share a session between a JSP and a JSP that is included by the first, consider the following: It is necessary to use the same `domino:session` tag or to dismiss the `domino:session` tag and all session attributes from all Domino top level tags in the latter JSP file.

- To prevent the Domino server from running out of resources, it is recommended to decrease the timeout value of the DIIOP session on the Domino server. Domino JSP tags can re-establish a timed out session, but the lower value will help to decrease the load on the Domino server.

- In the upcoming sample application, only Domino views with a small number of documents are used. In a real implementation, pagination is necessary. For simplicity reasons the session management issues described above are not addressed in the implementation example.
Section 6. Sample application

Overview

The following sections focus on the creation of a sample application that utilizes the Domino custom JSP tags. The Domino application exposed within a portlet is a Journal database based on the Personal Journal (R6) template that Lotus Domino Designer provides.

The functionality of the portlet consists of connecting to the server hosting the Domino application, displaying the entries of the views, and displaying information of selected entries. The flow of the application is described as follows:
1. Log in to the application, i.e. provide information needed to connect to the Domino server and the database.

2. Display the views.

3. Expand a selected view and display its entries.

4. Select a specific entry and show detailed information of the selected entry.

5. At every step within the application you can go back to the page before or log out, which enables you to connect to another database.

The following sections describe the steps for creating the sample application.

---

Section 7. Create a portlet project

Steps

1. Open WebSphere Studio Application Developer.
2. Switch to the J2EE Hierarchy View.

3. Right-click and select **New > Project...**

4. Select **Portlet Development**, select **Portlet Project**, then click **Next**.
5. Name the project DominoJspJournal and click Next.
6. On the J2EE Settings Page, accept the default settings and click **Next**.

7. On the Features Page accept the default settings and click **Next**.

8. On the Portlet Settings Page accept the default settings and click **Next**.

9. On the Event Handling Page deselect **Add form sample** and click **Next**.
10. On the Single-Sign-On Page accept the default settings and click **Next**.

11. On the Miscellaneous Page accept the default settings and click **Finish**.

12. In the Confirm Perspective Switch Dialogue click **No**.

13. Switch to the Project Navigator and expand the folder DominoJspJournal. The structure created by the creation wizard consists of:
   - Web Deployment Descriptor
   - Portlet Deployment Descriptor
   - Java Source folder
• Web Content folder
The Web Content folder contains the dominojspjournal/jsp/html folder in which the JSP files for the project are situated. The Java Source folder contains the dominojspjournal package that holds the sample portlet class that was created by the wizard. Before you start to implement the various components, you have to add support for the Domino custom JSP tag libraries to the application.

Section 8. Add support for Domino custom tag libraries

Create folder hierarchy

To use the Domino custom tags in your application:

1. Copy domtags.jar from the Notes/data/domino/java directory to the WebContent/WEB-INF/lib directory for the application.
2. Copy NSCO.jar from the Notes/data/domino/java directory to the WebContent/WEB-INF/lib directory for the application.
3. Create a new folder named *tld* in the WebContent/WEB-INF directory of your application.
4. Copy domtags.tld and domutil.tld from the Notes/data/domino/java directory to the WebContent/WEB-INF/tld directory of your application.
5. Verify that your folder hierarchy has the following structure:
Add libraries to web.xml

To create the necessary tag library XML tags in the Web Deployment Descriptor of your application:

1. Switch to the J2EE Hierarchy view.
2. Expand the folder Web Modules and double-click on DominoJournal to open the Web Deployment Descriptor.
3. Switch to References, click the JSP tag libraries tab.
4. Click **Add**, select **WebContent/WEB-INF/tld/domtags.tld**, and click **Finish**.

5. Click **Add**, select **WebContent/WEB-INF/tld/domutil.tld**, and click **Finish**.

6. Switch to Source.

7. Verify that the following XML tags have been added to your Web Deployment Descriptor:

   `<taglib>
     <taglib-uri>/WEB-INF/tld/domutil.tld</taglib-uri>
     <taglib-location>/WEB-INF/tld/domutil.tld</taglib-location>
   </taglib>

   `<taglib>
     <taglib-uri>/WEB-INF/tld/domtags.tld</taglib-uri>
     <taglib-location>/WEB-INF/tld/domtags.tld</taglib-location>
   </taglib>`

8. Save the changes and close the Web Deployment Descriptor.

---

Section 9. Modify the portlet Java source
Adding a SessionBean

To hold user-specific data, add a SessionBean to the project. The use of this SessionBean allows you to store data that is related to the user, through the user interaction with the application. Even though the user might switch to another WebSphere Portal page or might interact with other portlets, you are still able to display the same information once the user returns to your application.

1. Switch to the Project Navigator.
2. Right-click the dominojspjournal package contained in the Java Source folder of your application.
3. Select New > Class.
4. Name the class DominoJspJournalSessionBean.
5. Click Finish.
6. Open the file DominoJspJournalSessionBean.java in an editor.
7. Insert the following code:

   ```java
   private String jspPage = null;
   private String username = null;
   private String password = null;
   private String hostname = null;
   private String dbname = null;
   private String selectedView = null;
   private String uidDoc = null;
   ```

8. Select the inserted code and right-click into the window of the editor.
10. Click Select All.
11. Click OK.
12. Add the following constructor to the java file:
The following error should be displayed on the tasks view after saving the file:

```
DominoJspJournalPortlet.INDEX_JSP cannot be resolved.
```

In order to correct the error and to include support for our SessionBean we have to modify the

```
DominoJspJournalPortlet.java file.
```

Modifying the DominoJspJournalPortlet


2. Replace the existing declaration by:

   ```java
   // session data
   public static final String SESSION_BEAN = "dominojspjournal.DominoCapJournalSessionBean";
   public static final String REDIRECT = "dominojspjournal.DominoCapJournalPortletRedirect";
   public static final String Controller = "dominojspjournal.DominoCapJournalPortletController";
   public static final String Context = "dominojspjournal.DominoCapJournalPortletContext";
   public static final String Index = "dominojspjournal.DominoCapJournalPortletIndex";
   public static final String Dice = "dominojspjournal.DominoCapJournalPortletDice";
   public static final String Diary = "dominojspjournal.DominoCapJournalPortletDiary";
   public static final String View = "dominojspjournal.DominoCapJournalPortletView";

   // jsp root directory
   public static final String VIEW_JSP_ROOT = "/dominojspjournal/jsp/DominoJspJournalPortlet";

   // names of JSP files
   public static final String VIEW_JSP_INDEX = "View.jsp";
   public static final String VIEW_JSP_IFrames = "Iframes.jsp";
   public static final String VIEW_JSP_FULL_DOC = "FullDoc.jsp";
   public static final String VIEW_JSP_BROWSE предприятие = "Browse.jsp";
   public static final String VIEW_JSP_BROWSE_Document = "BrowseDoc.jsp";
   public static final String VIEW_JSP_BROWSE_Day = "BrowseDay.jsp";
   public static final String VIEW_JSP_BROWSE_Journal = "BrowseJournal.jsp";
   public static final String VIEW_JSP_BROWSE_Entry = "BrowseEntry.jsp";

   // actions
   public static final String ACTION_LOGIN = "dominojspjournal.DominoCapJournalPortletActionLogin";
   public static final String ACTIONLogout = "dominojspjournal.DominoCapJournalPortletActionLogout";
   public static final String ACTIONShowView = "dominojspjournal.DominoCapJournalPortletActionShowView";
   public static final String ACTIONShow_Journal = "dominojspjournal.DominoCapJournalPortletActionShowJournal";
   public static final String ACTIONShow_Document = "dominojspjournal.DominoCapJournalPortletActionShowDocument";
   public static final String ACTIONShow_Day = "dominojspjournal.DominoCapJournalPortletActionShowDay";
   public static final String ACTIONShow_Entry = "dominojspjournal.DominoCapJournalPortletActionShowEntry";
   ```

3. Create the method `getSessionBean()` by inserting the following code:
4. Modify the `doView()` method:

```java
/**
 * View spec implementation
 */

public void doView(FortletRequest request, FortletResponse response) throws FortletException, IOException {

    DominoJspJournalSessionBean sessionBean = getSessionBean(request);
    if (sessionBean == null) {
        // Insert routine to handle non-existence of a portlet session
        return;
    }

    String jspName = sessionBean.getJspPage();
    // Invoke the JSP to render
    getPortletConfig().getContext().include("VIEW_JSP_ROOT" + jspName + getJspExtension(request), request, response);
}
```

5. Modify and extend the `actionPerformed()` method:

```java
/**
 * Action spec implementation
 */

public void actionPerformed(FortletRequest request, FortletResponse response) throws FortletException, IOException {

    DominoJspJournalSessionBean sessionBean = getSessionBean(request);
    if (sessionBean == null) {
        // Insert routine to handle non-existence of a portlet session
        return;
    }

    String jspName = sessionBean.getJspPage();
    // Invoke the JSP to render
    getPortletConfig().getContext().include("VIEW_JSP_ROOT" + jspName + getJspExtension(request), request, response);
}
```

---

Section 10. Modify and create the JSP files

Modify and extend the DominoJspJournalPortletView.jsp
Modify and extend the DominoJspJournalPortletView.jsp file. The main task provided by this JSP is to extract the user information needed to connect to the Domino database. The JSP displays a form which can be used to submit the necessary information.


2. Verify that the page directives display the following code:

```html
<%@ page session="false" contentType="text/html" import="java.util.*,
   dominojspjournal.*" %>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>

<portletAPI:init/>
```

This ensures that you can use the JSP tag library that is provided by WebSphere Portal.

3. Create a skeleton of the form:

```html
<DIV style="margin: 8px">
  <DIV style="margin: 10px; margin-bottom: 8px">
    <FORM method="post" action="portletAPI:createReturnURL">
      <portletAPI:PAEAction name="<%=DominoJspJournalPortlet.ACTION_LOGIN%>">
        <portletAPI:encodeNamespace value="dominojspjournal.portletJournalPortletLoginForm"/>
      </portletAPI:PAEAction>
    </FORM>
  </DIV>
</DIV>
```

The tag encodeNamespace used in the form tag ensures uniquely associating the element with the concrete portlet instance and avoids name clashes. The other tags are used to add an action to the URI that points to the current portlet.

4. Insert the submit and reset fields:
5. Insert a field for submitting the username:

```html
<TD>
  <INPUT class="wpsButtonText"
         name="<portletAPI:encodeNamespace
             value='submit' />">
  <INPUT class="wpsButtonText"
         name="<portletAPI:encodeNamespace
             value='reset' />
```

6. Repeat step 5 for the fields:
   - password
   - hostname
   - dbname

   Change the corresponding entries used in the encodeNamespace tag of the Portal JSP tag accordingly.

7. In the field for the password, change the attribute type from text to password.

8. Save and close the file.

10. **Update the `actionPerformed()` method:**

    // handle login action

    // retrieve parameters from login form
    String username = request.getParameter(USER_NAME);
    String password = request.getParameter(PASSWORD);
    String hostname = request.getParameter(HOST_NAME);
    String dbname   = request.getParameter(DB_NAME);

    // set properties in sessionBean
    sessionBean.setUsername(username);
    sessionBean.setPassword(password);
    sessionBean.setHostname(hostname);
    sessionBean.setDbname(dbname);

    // set jsp file to be rendered next
    sessionBean.setJspPage(View_JSP_DB_VIEWS);

**Connect to a Domino database**

1. **Open the Domino View by selecting **Window > Show View > Other...**
2. Expand the folder Domino, select **Domino** and click **OK**.
3. In the Domino View, right-click and select **New database connection**...
4. Insert the name of your Domino host, your username, and the name of your Journal database. Click **Finish**.
5. Enter the password and click OK.


You will recognize subdirectories for Forms, Views, ServerAgents, and Utilities. You will concentrate on the Views and Utilities subdirectory. The Utilities folder allows you to access the whole library of existing Domino custom tags and you can insert any tag by right-clicking on it and selecting Add to Web Page. You will discover this option later on when creating our JSPs that leverage the Domino custom tags.

The Views folder allows you to access information about the Views and its Columns. You can open a View in the Lotus Domino Designer by right-clicking on the View entry and selecting Edit in Domino Designer. By right-clicking on the entry and using the Add to Web page functionality, you can either add all or single columns of the View to the JSP. Drag-and-Drop functionality for adding single tags or a collection of tags is also provided.

Create the JSP displaying the database Views


2. Right-click on the folder html and select New > JSP File.
3. **Enter** `DominoJspJournalPortletDatabaseViews` **in the field File Name**, select **Configure advanced options** and click **Next**.
4. On the Tag libraries page, click **Add**.
5. On the Select a Tag library page, select both entries for the Domino custom JSP tag libraries on the top of the list and click **OK**.

7. On the next page, accept the default settings and click Next.

8. On the next page, accept the default settings and click Finish.


10. Import import="dominojspjournal.*" to the page directives of the file and verify that the head of your file follows:
11. Insert the following code to the body of the JSP:

```html
<table cellspacing="0" cellpadding="0" border="0">
<tr>
<td align="center"><div id="portalAPI" class="portletAPI"><a href="javascript:portalAPI.showReturnURL();">
  <portletAPI:UIAction name="DominoJspJournalPortlet.ACTION_SHOW_SQL_VIEW"/>
  <portletAPI:UIParameter name="DominoJspJournalPortlet.VIEWNAME" value="All Documents"/>
</portletAPI:actorReturnURL>" /></a></div></td>
</tr>
<tr>
<td align="center"><div id="portalAPI" class="portletAPI"><a href="javascript:portalAPI.showReturnURL();">
  <portletAPI:UIAction name="DominoJspJournalPortlet.ACTION_SHOW_SQL_VIEW"/>
  <portletAPI:UIParameter name="DominoJspJournalPortlet.VIEWNAME" value="By Category"/>
</portletAPI:actorReturnURL>" /></a></div></td>
</tr>
<tr>
<td align="center"><div id="portalAPI" class="portletAPI"><a href="javascript:portalAPI.showReturnURL();">
  <portletAPI:UIAction name="DominoJspJournalPortlet.ACTION_SHOW_SQL_VIEW"/>
  <portletAPI:UIParameter name="DominoJspJournalPortlet.VIEWNAME" value="By Diary Date"/> </a></div></td>
</tr>
</table>
```

This creates a table of links. A click on any of the links will invoke the action to show the entries of the specified view.

12. Save and close the JSP file.


14. Update the `actionPerformed()` method:
15. Save and close the Java file.

"All Documents View" JSP

1. Create a new JSP file named DominoJspJournalPortletAllDocs.jsp and add support for the Domino custom Tag libraries as described before.

2. Add import="dominojspjournal.*" to the page directives of the file.

3. Add session="true" to the page directives of the file.
4. To get access to the attributes saved in the SessionBean, insert the following code snippet directly beneath the head of the file:

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1" import="dominojspjournal.*" session="true" %>
<%@ taglib uri="/WEB-INF/tdl/portlet.tld" prefix="portletAPI" %>
<%@ taglib uri="/WEB-INF/tdl/domtags.tld" prefix="domino" %>
<%@ taglib uri="/WEB-INF/tdl/domutil.tld" prefix="domutil" %>
<portletAPI:init />
```

5. Place the cursor beneath the snippet and switch to the Domino View.

6. Expand the folder for the Journal database.

7. Expand Utilities > Domino Custom Tags > Domtags > Alphabetical.

8. Right-click on `session` and select Add tag to Web page. Verify that the inserted code equals:

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1" import="dominojspjournal.*" session="true" %>
<%@ taglib uri="/WEB-INF/tdl/portlet.tld" prefix="portletAPI" %>
<%@ taglib uri="/WEB-INF/tdl/domtags.tld" prefix="domino" %>
<%@ taglib uri="/WEB-INF/tdl/domutil.tld" prefix="domutil" %>
<portletAPI:init />
```

9. Modify the `<domino:session>` tag:

- Change the `host` attribute to `<%=
sessionBean.getHostname()%>

- Change the **user** attribute to `<%= sessionBean.getUsername()%>`.

- Change the **password** attribute to `<%= sessionBean.getPassword()%>`.

- Insert the **duration** attribute and set it to "page".

10. Insert the `<domino:db>` tag and modify its attributes:

- Change the **dbname** attribute to `<%= sessionBean.getDbname()%>`.

11. Insert the `<domino:view>` tag and modify its attributes:

- Change the **viewname** attribute to `<%= sessionBean.getSelectedView()%>`.

12. Insert the following code:

```html
<domino:db>
  <!-- Insert the following code: -->
  
  <!-- This creates the skeleton of the form we will use to select a specific view entry and display its detailed information. -->

  <!-- Extend the form skeleton by the following: -->
```
<table>
  <tr>
    <th>Title</th>
    <th>Modified</th>
  </tr>
  <tr>
    <td><input type="radio" name="viewItem[0]" value="DominoJspJournalPortlet.UIDDOC"/>
    <td><input type="radio" name="viewItem[1]" value="DominoJspJournalPortlet.UIDDOC"/>
  </tr>
</table>

This code fragment creates a table that holds the view entries. Every entry is associated with a radio button. By selecting the entry, the universal ID of the selected document entry is added to the request object and can be extracted by the corresponding action handler within the portlet.

14. Save and close the JSP file.


16. Update the actionPerformed() method:

   ```java
   // handle display entry details action
   String uidDoc = request.getParameter("UIDDOC");
   sessionBean.setUidDoc(uidDoc);
   sessionBean.setJspPage("VIEW_JSP_JOURNAL_ENTRY");
   ```

17. Save and close the Java file.

The "By Category View" JSP

1. Create a new JSP file named DominoJspJournalPortletByCategory.jsp and add support for
the Domino custom Tag libraries as described before.

2. Copy all the code from `DominoJspJournalPortletAllDocs.jsp` into `DominoJspJournalPortletByCategory.jsp`.

3. Exchange the start tag of the form skeleton and the table that displays the view entries by:

   ```html
   <form method="post">
   <portletAD:action name="DominoJspJournalPortlet.ACTION_SHOW_ENTRIES" />
   <portletAD:encoders>
     <portletAD:encoder name="DominoJspJournalPortletByCategory.jsp"/>
   </portletAD:encoders>
   
   <table>
   <tr>
   <th>Categories</th><th>Diary date</th><th>Topic</th><th>Modified</th>
   </tr>
   <tr>
   <td>
   <td>
   <div>
   <input type="radio" name="items:1234567890" value="domino:documentKey"/>
   <input type="radio" name="items:1234567890" value="domino:uid"/>
   </div>
   </td>
   </tr>
   </table>
   
   4. Save and close the JSP file.

"By Diary Date View" JSP

1. Create a new JSP file named `DominoJspJournalPortletByDiaryDate.jsp` and add support for the Domino custom Tag libraries as described before.

2. Copy all the code from `DominoJspJournalPortletAllDocs.jsp` into `DominoJspJournalPortletByDiaryDate.jsp`.

3. Exchange the start tag of the form skeleton and the table that displays the view entries by:
4. Save and close the JSP file.

Journal Entry JSP

1. Create a new JSP file named DominoJspJournalPortletJournalEntry.jsp and add support for the Domino custom Tag libraries as described before.

2. Add import="dominojspjournal.*" to the page directives of the file.

3. Add session="true" to the page directives of the file.

4. To get access to the attributes saved in the SessionBean insert the following code snippet directly beneath the head of the file:
5. Place the cursor beneath the snippet and switch to the Domino View.
6. Expand the folder for the Journal database.
7. Expand Utilities > Domino Custom Tags > Domtags > Alphabetical.
8. Right-click on `session` and select Add tag to Web page. Verify that the inserted code equals:
   ```xml
   <%-- Start:Session-specific code -->%
   <domino:session host="itso-tdcm.can.itso.ibm.com" user="" username="" password="" duration="page">
   <%-- End:Session-specific code -->
   </domino:session>
   `<%-- Start:Session-specific code -->
   ```

   ```java
   catch (lotus.domino.TagLib.DominoTagException e) {
     lotus.domino.NotesException ne = e.getNotesException();
   } catch (Exception e) {
     e.printStackTrace(new java.io.PrintWriter(out));
   }
   ```

9. Modify the `<domino:session>` tag:
   - Change the `host` attribute to `<% sessionBean.getHostname() %>`.
   - Change the `user` attribute to `<% sessionBean.getUsername() %>`.
   - Change the `password` attribute to `<% sessionBean.getPassword() %>`.
   - Insert the `duration` attribute and set it to "page".

10. Insert the `<domino:db>` tag and modify its attributes:
    - Change the `dbname` attribute to `<% sessionBean.getDbname() %>`.
11. Insert the `<domino:document>` tag and modify its attributes:
   • Change the `unid` attribute to ` <%= sessionBean.getUidDoc()%>`. 

12. Insert the following code:

```html
<table>
<tr>
  <td><b>Categories:</b></td>
  <td><domino:item name="Categories"/></td>
</tr>
<tr>
  <td><b>Time Created:</b></td>
  <td><domino:item name="TimeCreated" format="date=medium"/></td>
</tr>
<tr>
  <td><b>Diary Date:</b></td>
  <td><domino:item name="DiaryDate" format="date=medium"/></td>
</tr>
<tr>
  <td><b>Subject:</b></td>
  <td><domino:item name="Subject"/></td>
</tr>
<tr>
  <td><b>Body:</b></td>
  <td><domino:item name="Body"/></td>
</tr>
</table>
```

This code creates a table that displays several details of the selected entry.

Navigating through the different JSPs

To navigate through the different JSPs and to logout, you have to extend our JSPs:

1. Insert the following code into DominoJSPJournalPortletAllDocs.jsp, DominoJSPJournalPortletByCategory.jsp, and DominoJSPJournalPortletByDiaryDate.jsp.
2. Insert the following code into DominoJSPJournalPortletJournalEntry.jsp.

3. Insert the following code into DominoJSPJournalPortletDatabaseViews.jsp.
Section 11. Combine Domino custom JSP tags and Java API

Modifying the DominoJSPJournalPortletDatabaseViews.jsp

To overcome some of the limitations that constrain the use of Domino custom JSP tags, extend the JSPs that are based on these tags through the use of Java. This leads to the JSP-Java-Hybrid option explored during this section:


2. To get access to the attributes saved in the SessionBean insert the following snippet directly beneath the head of the file:

   ```
   <a>
   DominoJspJournalSessionBean sessionBean =
   (DominoJspJournalSessionBean) portletRequest.getPortletSession().getAttribute(DominoJspJournalPortlet.SESSION_BEAN);
   </a>
   ```

3. Replace the code that renders the table including the links for expanding the selected entry by:
4. Insert the following session-specific code:

```html
<% try { %>
    <domino:session id="generalSession"
    host="<%=sessionBean.getHostname()%>"
    user="<%=sessionBean.getUsername()%>"
    password="<%=sessionBean.getPassword()%>"
    duration="page">
    <!-- Start:Session-specific code -->

    <!-- End:Session-specific code -->
    <%-- Start:Session-specific code --%>

    <!-- End:Session-specific code --%>
    </domino:session>
    <% } catch (lotus.domino.taglib.DominoTagException e) { %>
    lotus.domino.NotesException ne = e.getNotesException();
    <% } %>
    <b>Tags Error:</b> <%= e.getMessage() %><br>
    <b>Notes Error:</b> <%= ne.getMessage() %><br>
    <% } catch (Exception e) { %>
    e.printStackTrace(new java.io.PrintWriter(out));
    <% } %>
```

5. Insert code that renders the table body, i.e. code that creates links to...
expand a specific view:

```java
// get views from database
Vector allViews = generalDb.getViews();

// check for existent views
if (allViews.size() > 0) {
    String viewName = null;

    // iterate through views
    Enumeration enum = allViews.elements();
    while (enum.hasMoreElements()) {
        // get name of current view
        View currentView = (View)enum.nextElement();
        viewName = currentView.getName();

        <!-- create table body displaying the name of the views as links -->
        <tr>
            <td>
                <a href="/portletAPI:createReturnURL"
                    portletAPI:URLAction="#-DominoJspJournalPortlet.ACTION_ADMN_SZL_VIEW"></a>
                <a portletAPI:URLParameter name="#-DominoJspJournalPortlet.VIEWNAME">" value="#-viewName"></a>
            </td>
        </tr>
    }
}
```

The first step is to get all the views that consist in the database. Use the ID attribute defined in the domino:db tag to get access to the database object. Afterwards, use Java code to iterate through all of the views and create table rows that include a link to the specific view.

6. Several errors will be shown in tasks view. To solve the problems, add the specific packages to the page directives and verify that the file header equals:

```xml
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<%@ taglib uri="/WEB-INF/tld/domtags.tld" prefix="domino" %>
<%@ taglib uri="/WEB-INF/tld/domutil.tld" prefix="domutil" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"
import="dominojspjournal.*, lotus.domino.*, java.util.*"
%>
<portletAPI:init />
```

At this stage you are only using the Domino custom JSP tags to open the Domino session and to create a database connection. Once you established the connection, retrieve the database object and start using the Java API for LCC and Lotus Domino. This example shows that it is possible to extend the options given by the Domino custom JSP tags with customized Java code, which brings the application certainly on a higher level and leads to a hybrid option that fills the gap between the
use of Domino custom JSP tags for portalizing the Lotus Domino application and the use of the Java API for integrating the Lotus Domino application into a portal environment.

Section 12. Summary

Summary

This tutorial explained both the benefits and the methodology for integrating a Domino application into WebSphere Portal utilizing the Domino custom JSP tags. It provided an overview of the JSP technology and outlined the advantages, as well as the limitations of using the Domino custom JSP tag libraries. Finally, it explained both the theoretical background and covers step-by-step instructions on how to build a portlet application that accesses a Domino Journal database leveraging the Lotus Domino Toolkit for WebSphere Studio.

The theoretical background comprises:

- Why combine Lotus Domino and WebSphere Portal?
- Which integration options can I choose to portalize a Domino application?
- What is J2EE and what are the advantages?
- What is meant by JSP?
- What are the Domino custom JSP tags?

The development of the sample application enclosed:

- How to create a Portlet Project.
- How to add support for the Domino custom JSP tags.
- How to connect to Domino using the Lotus Domino Toolkit.
- How to build JSPs including the Domino custom JSP tags.
- How to bring your application to a higher level by combining Java API for LCC and Domino with the Domino custom JSP tags.
Resources

Learn

- For more detailed information on portalizing Domino applications see the Redbook, "Portalizing Domino Applications for WebSphere Portal".
- For more detailed information on servlet and JSP programming, see the Redbook, "Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java".
- For more detailed information on portlet development see the Redbook, "IBM WebSphere Portal V5 A Guide for Portlet Application Development".
- For more detailed information on WebSphere Portal, check out the developerWorks Web site.
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- Build your next development project with IBM trial software, available for download directly from developerWorks.

About the authors

Thomas Reske
Thomas Reske studies Applied Computer Science at the University for Cooperative Education (Berufsakademie), Stuttgart. He joined the University Education program of IBM Germany in 2002 and has been working on several projects related to Lotus Software products. His areas of expertise include J2EE and J2ME development, WebSphere Portal, and Lotus Software.

John Bergland
John Bergland is a Project Leader at the International Technical Support Organization, Cambridge Center. He manages projects that produce Redbooks about Lotus Software products. Before joining the ITSO in 2003, John worked as an Advisory IT Specialist with IBM Software Services for Lotus (ISSL), specializing in Notes and Domino messaging and collaborative solutions.