

# Simplified JSP page development for Lotus Domino

With custom tags, you can have it all

Skill Level: Introductory

[George Brichacek \(george\\_brichacek@us.ibm.com\)](mailto:george_brichacek@us.ibm.com)  
Technology Advocate  
IBM Software Group

30 Sep 2002

Do you need the powerful collaborative features that Lotus Domino provides, such as industrial-strength messaging, interactive applications, secured access, workflow, and content management? Do you also want to take advantage of J2EE applications and the object-oriented, portable, multi-threading capabilities they provide? You can have it all with JavaServer Pages (JSPs) technology. Lotus Domino 6 provides custom tag libraries that make it easier to build JSP pages that allow you to access, update, create, and delete information in Lotus Domino.

## Section 1. Introduction

### What is this tutorial about?

Do you need the powerful collaborative features that Lotus Domino provides, such as industrial-strength messaging, interactive applications, secured access, workflow, and content management? Do you also want to take advantage of J2EE applications and the object-oriented, portable, multi-threading capabilities they provide? You can have it all with JavaServer Pages (JSPs) technology. The latest release of Lotus Domino -- version 6 -- provides custom tag libraries that make it easier to build JSP pages that allow you to access, update, create, and delete information in Lotus Domino.

The purpose of this tutorial is to help you understand JSP technology and how to use it to access Lotus Domino. We'll begin with a brief overview of JSP pages. We'll then use a news story scenario to walk you through the steps to create a JSP page to access a sample database. The JSP page will use the new Domino custom tag libraries to access the Domino database, and it will select and then create a list of approved news stories. This is a simple example, but it shows you what you need to know without getting sidetracked on other issues.

Because Domino can store large amounts of data, and because it stores data in such a way that it is easily accessible, the news stories are stored in a Domino database. A workflow process could also be implemented within Domino to review and approve the news stories. We'll use the Application Developer configuration of WebSphere Studio to develop and test our JSP page. The JSP page will be hosted on WebSphere Application Server since JSP pages cannot execute on a Lotus Domino server.

## Who should take this tutorial?

This tutorial is intended for developers working in an environment where Lotus Domino is deployed and Domino applications are already developed. It's also intended for developers interested in using Domino 6 to extend their existing applications with collaborative services such as threaded discussions, document libraries, workflow, and content management.

Even though JSP pages can include Java code, Java programming skills are not required to complete this tutorial. We'll be using the Domino custom tags which simplify the JSP development process.

A basic understanding of Application Developer is required. In Application Developer you need to know how to create a Web project, and there should already be a server instance and server configuration established. Refer to the help perspective in Application Developer for more information.

## Software requirements

You need to have the following products installed on your machine in order to complete this tutorial. If you do not already have licensed copies of the software, you can download free trial versions of the products. Be sure to follow the installation instructions provided with the products.

### **Lotus Notes and Domino**

Lotus Notes/Domino is groupware software. Notes is the e-mail, calendaring, group scheduling, Web access and information management client. Domino is the integrated messaging and Web application server providing directory services,

messaging, security, and much more. This tutorial uses Notes/Domino 6. You can download a free trial of the [latest version of Notes/Domino](#).

### **WebSphere Studio Application Developer**

Application Developer is the application development tool for the WebSphere Application Server. We'll use Application Developer Version 4 to develop our Web services application.

### **WebSphere Application Server**

WebSphere Application Server provides a complete set of application services to host your J2EE applications. These services include capabilities for transaction management, security, clustering, performance, availability, connectivity and scalability. This tutorial uses WebSphere Application Server version 4. You can download a free trial of the [latest version of WebSphere Application Server](#).

### **Sample Domino database**

Download [NewsStories.nsf](#). We'll use this database in our scenario.

---

## Section 2. Overview of JSP technology

### What are JSP files?

JavaServer Pages (JSP) files are similar to HTML files, but they provide the ability to display dynamic content within Web pages. The page is compiled at the time it is requested on the Web application server. JSP technology was developed by Sun Microsystems to separate the development of dynamic Web page content from static HTML page design. The result of this separation means that the HTML page design can change without the need to change the dynamic Web page.

### Why should you use JSP technology?

Here are some of the advantages of using JSP technology:

#### **Separation of static and dynamic content**

This reduces the complexity of Web site development and makes the site easier to maintain. HTML developers focus on design and JSP developers focus on dynamic content.

#### **Platform independence**

JSP technology is Java based and platform independent. JSP pages can run

on nearly any Web application server, including WebSphere Application Server. They can be developed on any platform and, since the output of a compiled JSP file is HTML, they can be viewed by any browser.

### Component reuse

Using JavaBeans and Enterprise JavaBeans, JSP pages leverage the inherent reusability offered by these technologies. By sharing these components, Web site development is much faster.

### Scripting and custom tags

JSP pages support both embedded JavaScript and custom tags. JavaScript is typically used at the browser client for field validation and other page-level functionality. Custom tags reduce the necessity to embed large amounts of Java code in JSP pages.

## How JavaServer Pages work

JSP pages are made available by having their contents (HTML tags, JSP tags, scripts, and custom tags) compiled into a servlet by the Web application server on the server side. The following image illustrates both the initial invocation of a JSP page and subsequent invocations.

## What does a JSP page look like?

A JSP page looks like a standard HTML page, with additional tags that the JSP engine processes and strips out. Typically, the JSP elements create text that is inserted into the results page.

The following JSP page is very simple. It prints the day of the month and the year, and welcomes you in the morning or in the afternoon, depending on the time of day.

```
<html>
<title>Date Display</title>
<body>

<!-- D I R E C T I V E S -->
<%@ page language = "java" %>
<%@ page import = "java.util.*" %>
<%@ page contentType = "TEXT/HTML" %>

<!-- S C R I P T L E T S -->
<H3>
<% if (Calendar.getInstance().get(Calendar.AM_PM) == Calendar.AM)
    {%>
    How are you this morning,
    <%} else {%>
    How are you this afternoon,
    <% }
%>
```

```

WebSphere 4 User ?
</H3>
<HR>

<!-- A C C E S S I N G   I M P L I C I T   O B J E C T S -->
<% out.println("Here is the <b>Date Display JSP</b>"); %>

<!-- D E C L A R A T I O N S -->
<%!
private int calledCount = 0;
private String getDate(GregorianCalendar gcalendar) {
    StringBuffer dateStr = new StringBuffer();
    dateStr.append(gcalendar.get(Calendar.DATE));
    dateStr.append("/");
    dateStr.append(gcalendar.get(Calendar.MONTH) + 1);
    dateStr.append("/");
    dateStr.append(gcalendar.get(Calendar.YEAR));
    return (dateStr.toString());
}
private int incrementCounter() {
    return (++calledCount);
}
%>

<H1> Today's Date is: <%= getDate(new GregorianCalendar()) %> </H1>
<H1> This page has been called: <%= incrementCounter() %> time(s)</H1>

</body>
</html>

```

Let's break down this example and look at some of its components.

## Components of a JSP page

JSP tags are typically implemented as standard or customized tags and they have an XML tag syntax. Tags that the JSP engine does not recognize are passed on with the results page. Typically, these will be HTML tags. For example:

```

<html>
<title>Date Display</title>
<body>

```

The components of a JSP page are:

- **Directives**

Directives are page-specific instructions to the JSP parser. These do not produce any visible output. There are three Directives:

- *Include* (includes a file of text or code when the JSP page is translated)
- *Page* (defines attributes that apply to an entire JSP page)
- *Taglib* (defines a tag library and prefix for the custom tags used in the

## JSP page)

In the following example, the JSP directives indicate the location of some Java programming language extensions -- the language and content type -- that are to be accessible from this page. Directives are enclosed in `<%@` and `%>` markers.

```
<%@ page language = "java" %>
<%@ page import = "java.util.*" %>
<%@ page contentType = "TEXT/HTML" %>
```

- **Declaration**

Declarations initialize variables and methods. In the following example, `incrementCounter()` increments the `calledCount` variable by one every time it's called. Declarations are not supposed to be used for variables that are specific to each users request. They are to be used for all user requests.

```
<%!
private int calledCount = 0;
private String getDate(GregorianCalendar gcalendar) {
    StringBuffer dateStr = new StringBuffer();
    dateStr.append(gcalendar.get(Calendar.DATE));
    dateStr.append("/");
    dateStr.append(gcalendar.get(Calendar.MONTH) + 1);
    dateStr.append("/");
    dateStr.append(gcalendar.get(Calendar.YEAR));
    return (dateStr.toString());
}
private int incrementCounter() {
    return (++calledCount);
}
%>
```

- **Scriptlet**

A scriptlet is a small script that performs functions not supported by tags. Scriptlets are also used to tie everything together. The native scripting language for JSP software is based on the Java programming language. The scriptlet in the following example determines whether it is AM or PM and greets the user accordingly.

```
<% if (Calendar.getInstance().get(Calendar.AM_PM) == Calendar.AM)
    {%>
    How are you this morning,
    <%} else {%>
    How are you this afternoon,
    <% }
%>
WebSphere 4 User ?
```

- **Comments**

Comments can be Hidden (they document the JSP page but are not sent to the client), or they can be HTML (they generate a comment that can be

viewed in the HTML source file).

```
<%-- S C R I P T L E T S --%>
```

- **Expression**

An expression generates output, or contains an expression or variable. The JSP engine evaluates anything between `<%=` and `%>` markers as an expression. The expression is highlighted in bold in the following example:

```
<H1> Today's Date is: <%= getDate(new GregorianCalendar()) %> </H1>  
<H1> This page has been called: <%= incrementCounter() %> time(s)</H1>
```

---

## Section 3. Accessing Domino

### How to access Domino with a JSP page

Lotus Domino provides a Java API. The Java API consists of a set of classes (sometimes called back-end classes) that JSP pages can use to access information within Domino databases. These back-end classes allow a programmer to start at the session class, locate and open Domino databases, manipulate data within these databases, run agents, move between documents within a view, and look at fields within a document.

### Controlling access

Security is one of the strengths of a Domino database. When you build JSP pages that access Domino data, you can take advantage of this powerful security model. There are three levels of Domino security:

1. **Server-level security**  
Access is controlled by having the user log on to the server.
2. **Database-level security**  
Conditional access to the database is set up based on user information as defined in the Access Control List (ACL).
3. **Document-level security**

Document access is controlled by defining the names of users who are allowed to read or author that document.

## Domino custom tag libraries

Since JSP pages use tags and scriptlets written in the Java programming language, you can write Java code to access a Domino server R4.6 or higher right within the JSP page. However, writing Java is not the easiest thing to do. This is where JSP tag libraries come into play. JSP tag libraries define declarative, modular functionality that can be reused by any JSP page. Tag libraries reduce the necessity to embed large amounts of Java code in JSP pages by moving the functionality provided by the tags into tag implementation classes. In doing so, tag libraries make authoring JSP pages easier -- both for the Web page author and for the tools that expose the functionality encapsulated by the library to the author.

With Domino 6, we have two Domino JSP tag libraries. Both comply to the JSP 1.1 and Java Servlet 2.2 specifications developed by Sun Microsystems. They are:

- `domtags.tld` -- the collaboration tags for accessing standard, back-end objects in a Domino database
- `domutil.tld` -- the utility tags for performing tasks that are common to all J2EE Web containers

---

## Section 4. Set up the development environment

### Overview

There are a few steps you must perform before you can use the tags in a Web application. We'll use Application Developer for these steps.

Launch Application Developer and create a new Web project. In our example, the project name will be `DominoJSPProject` and the enterprise application project name is `DominoJSPEAR`.

### Import the `domtags.jar` file

After creating the Web project, you need to import the `domtags.jar` file into the new project. Select the `\webApplication\WEB-INF\lib` folder, then select **File =>**

**Import.** Select **File system** to import resources from the local file system. Browse the directory `C:\Lotus\Domino\Data\domino\java` for the file `domtags.jar`. Click **Finish**.

The Web project should look like:

## Import the tag library files

It doesn't really matter where you import the tag library files, but by default, they are stored in a `tlds` folder. If the folder does not exist, you need to create it. Select the `WEB-INF` folder and create a new folder under it named `tlds`. Select the new folder and then **File => Import**. Select **File system** to import resources from the local file system. Browse the directory `C:\Lotus\Domino\Data\domino\java` for the files `domtags.tld` and `domutil.tld`. Click **Finish**.

The Web project should look like:

## Update the web.xml file

We need to define the Domino tag library files in the `web.xml` file. Double-click the file `web.xml`, which is in the `WEB-INF` folder. Select the `References` tab along the bottom of the frame. Select **JSP tag libraries**. Finally, select the `Add` button. For the `URI name`, enter `domtags.tld`. Select the `location line`, then select the drop-down arrow button and select the `domtags.tld` file in the `WEB-INF\tlds` folder.

Do the same procedure for the `domutil.tld` file.

The `web.xml` frame should look like this:

## Update the classpath

We need to add either of the following Domino Objects for Java JAR files to the classpath in `Application Developer`:

- `NOTES.JAR` - Use this if you are setting up the application for local access (Lotus Domino and `Application Developer` on the same machine).
- `NCSO.JAR` - Use this if you are setting up the application for remote (CORBA/DIIOP) access (Lotus Domino and another Web application server, like `WebSphere`, on different machines).

In a test environment, typically you'll have Lotus Domino and `Application Developer` installed on the same machine. We're assuming that this is the configuration for this

tutorial. Therefore, we'll add the `Notes.jar` file to the Java Build Path in Application Developer. Select the `DominoJSPProject` folder, right-click, and then select **Properties**. Select **Java Build Path**, then the Libraries tab. Click the Add External JARs button. Look in `C:\Lotus\Domino` for the `Notes.jar` file to add it to the classpath. When finished, click **OK** on the properties dialog box.

The Java Build Path in the properties dialog box should look like:

## Final notes

Our development environment should now be all set up. Application Developer should be ready to add Domino JSP custom tags. For our example, Lotus Domino 6 server should be installed on the same machine as Application Developer. There shouldn't be any problem with HTTP ports. Application Developer will use port 8080 and Lotus Domino will use port 80. The demo Domino database, `NewsStories.nsf`, should be deployed on the Domino server.

---

## Section 5. Create the JSP page

### Core Domino JSP tags

The following core tags serve as the primary means of accessing Domino data when used in a JSP page. We will only be using the session and view tags to access our news stories database.

#### **session**

Defines the environment that the core collaboration tags run in. Use the session tag when multiple core tags are included on the same JSP page. This initializes and tears down the Domino session one time only, providing improved performance.

#### **db**

Provides a database context for all enclosed tags. If you are including several core tags that are running off the same database in a page, you can wrap them in this tag to increase scalability.

#### **document**

Represents a Domino document or database record.

#### **form**

Provides a schema for document creation and access. This tag should contain tags that build the JSP interface, such as the formula, item, richtext, select, and textarea tags, and the action and validation tags.

**ftsearch**

Enables a full text search on a page. The result is a list of documents that fit the search criteria.

**mailto**

Sends a mail message via Domino mail.

**runagent**

Runs a specified agent on the server. You can specify agents that run on the server, such as agents that send mail or create a folder or a document. You cannot specify an agent that displays information in the browser to the user.

**view**

Displays a list of documents to select from. It displays the summary information of a subset of documents in a database. You can use the key and ftsearch attributes to further qualify the subset.

## Create a new JSP file

Let's get started. Create a new JSP file in our project. Select the webApplication folder, right-click and select **New => JSP File**. For the file name, enter `DominoNewsStories` and then click **Finish**.

## Change the page title

We'll change the page title to something more descriptive to document our new JSP file and to improve maintainability.

To change the page title, right-click on the JSP file in design mode and select **Attributes...** This will open the attribute dialog for the JSP file. Change the page title to something like `Domino News Stories`. Keep the attribute dialog box open and go to the next section.

## Select a tag type of page directive

Next we need to add a page directive and identify Java as the scripting language. The attributes dialog box should still be displayed, if not, right-click on the JSP file in design mode and select **Attributes...** to open the attribute dialog for the JSP file.

Make sure that Tag: at the top of the dialog box says *Page properties*.

Select the right arrow button to scroll the tabs to the right. Select the JSP Tags tab. In the Tag Type drop down, select **JSP Directive - page**.

## Add a page directive

Select the Add... button to add the page directive. In the language field enter `Java`. Then click **OK**. This will add: `<%@ page language="java" %>` to our JSP page. Back on the Attributes dialog, select **OK** to return to the JSP design screen.

---

## Section 6. Add Domino JSP tags

### Incorporating the tag libraries

We need to add a taglib directive. This tells the Java parser where to find the tag library descriptor (tld) files `domtags` and `domutil`.

Right-click on the JSP file in design mode and select **Attributes...**, which opens the attribute dialog for the JSP page. Make sure that Tag: at the top of the dialog box says *Page properties*. Select the right arrow button to scroll the tabs to the right. Select the JSP Tags tab. In the Tag Type drop down, select **JSP Directive - taglib**.

### Add two taglibs

Select the Add... button to add the taglib directive. In the URI field select the down arrow button and select **domtags.tld**. The tag prefix should be filled in with *domino*. Then press **OK**.

Perform the same procedure to add `domutil.tld` taglib.

### Let's review

Let's see what we've done so far. We set up Application Developer to support the Domino JSP tag libraries, `domtags.tld` and `domutil.tld`. We added a page directive and two taglib directives. You can't see these JSP tags in our JSP page, because the JSP file is in design mode. If you select the Source tab, you will see the JSP

tags.

## Now we're ready

We're finally ready to start adding Domino JSP tags. For this news story demo, we'll be accessing a specific view in a Domino database. However, this could be any Domino database and any view you might have. Using JSP pages, you can integrate information from any Domino databases with other business applications not running Domino, as long as the other applications are executing on an application server that supports the execution of JSP files.

The demo database should already be deployed to a Domino server. The demo Domino database name is `NewsStories.nsf` and in our example, the Domino server name is `geo2001`.

Go back to design mode. Select the Design tab.

Change the text *Place DominoNewsStories.jsp's content here* to `Domino News Stories`. Play around with Application Developer and make the text a Header 1.

Place the cursor below the text. Along the top, select **JSP => Insert Custom....** You should see the Insert Custom Tag dialog. The two tag libraries are listed in the left-hand column and the Domino tags are listed in the right-hand column. Select the session tag and then select the Insert button.

Before we close the dialog, we'd like to include a view within this session, so select **View** and then select the Insert button. Finally, select the Close button.

## Add session attributes

You should see two icons that represent the session and view tags. Select the first one, session icon, and right-click and select **Attributes...**, the attributes dialog box will appear.

Select the down arrow after Name:. You'll see all the possible attributes for the session tag. The attributes we need to add are *user* and *password*. The values will depend on your Domino server. Enter a valid user that has the proper access to the database. In our example, the values are `Joe Admin` and `password`. Click **OK**.

## Add view attributes

You should see two icons that represent the session and view tags. Select the second one, view icon, and right-click. Select **Attributes...**, the attributes dialog box

will appear.

Select the down arrow after Name:. You'll see all the possible attributes for the view tag. The attributes we need to add are *dbname* and *viewname*. The values should be `NewsStories.nsf` and `AllNewsStoriesView`. Click **OK**.

## Switch to source mode

Now we'll add the function to loop through all the documents in the view and select column values. It's easier to do this by entering the tags directly into the source. Switch to the source view by selecting the Source button.

After the `<domino:view>` tag and before the `</domino:view>` tag, enter the following tags:

```
<domino:viewloop>
  <domino:ifcategoryentry>
    <H2><domino:viewitem name="Category" /></H2>
  </domino:ifcategoryentry>
  <domino:ifdocumententry>
    <H3><domino:viewitem name="Subject" /></H3>
    <domino:viewitem name="Short Description" />
  </domino:ifdocumententry>
</domino:viewloop>
```

Let's examine what this code is doing. The `<domino:viewloop>` tag loops through all the documents in the view. Our view has categories, so we check if the entry in the view is a category using the `<domino:ifcategoryentry>` tag. If it is a category, we place H2 tags around it.

The `<domino:viewitem name="Category">` tag selects the view column data. The name attribute is the name of the column in the view.

The `<domino:ifdocumententry>` tag will check if the next entry in the view is a document. If it is, then we place H3 tags around it.

Just as with the category entry, we select the view column data. This is done with a `<domino:viewitem name="Subject">` tag. The name attribute is the name of the column in the view. Place this tag in between H3 tags.

Last, we place the short description on the page using the `<domino:viewitem name="Short Description">` tag.

**Note:** Application Developer has a code complete feature to assist you in adding HTML and JSP tags by hand. To use it, begin typing the tag you want to add and then hold down the CTRL key and hit the space bar. A pop-up window will appear in which you can continue typing the tag name or scroll through the list of options.

Make sure you save the JSP file. Select **File => Save**. The complete JSP file should look like this:

```
<HTML>
<HEAD>
<TITLE>Domino News Stories</TITLE>
<%@page language="Java" session="true" isThreadSafe="true" isErrorPage="false"%>
<%@taglib uri="domtags.tld" prefix="domino"%>
<%@taglib uri="domutil.tld" prefix="domutil"%>
</HEAD>
<BODY>
<H1>Domino News Stories</H1>
<domino:session user="Joe Admin" password="password">
  <domino:view dbname="NewsStories.nsf" viewname="AllNewsStoriesView">
    <domino:viewloop>
      <domino:ifcategoryentry>
        <H2><domino:viewitem name="Category"/></H2>
      </domino:ifcategoryentry>
      <domino:ifdocumententry>
        <H3><domino:viewitem name="Subject"/></H3>
        <domino:viewitem name="Short Description"/>
      </domino:ifdocumententry>
    </domino:viewloop>
  </domino:view>
</domino:session>
</BODY>
</HTML>
```

---

## Section 7. Run the JSP file on the server

### Set up and start the WebSphere V4.0 test environment

Before we start the WebSphere test server, we need to add the `Notes.jar` file to the server's classpath. In the Server Configuration view, select **WebSphere v4.0 Test Environment**. Right-click and select **Open**. Select the Paths tab. Click the Add External JARs... button and add the `Notes.jar` file which should be in the `C:\Lotus\Domino` directory.

In the Servers view, select the server instance **WebSphere v4.0 Test Environment**, right-click and **Start** the server.

A couple of things you should note. You should see `C:\Lotus\Domino\Notes.jar` at the end of the classpath statement. Then, you should see *Server Default Server open for e-business* message in the server console.

## Launch the JSP page

In the navigator view, select **DominoNewsStories.jsp**, right-click and select **Run on Server**. If everything goes as planned, you should see the following page:

---

## Section 8. Summary

### What we've learned

That's all the time we have for now. You now understand what JSP technology is. You can set up Application Developer to support the development of a JSP page using the Domino custom tag library. And, you can develop a JSP page in Application Developer that will access Lotus Domino 6.

Just think of all the possibilities for creating JSP pages that will access Domino. Our example was just reading a view in Domino. You can create JSP forms to update and/or create documents in Domino. You can create e-mail messages and have Domino route them. You can create workflow applications with e-mail notifications, reminders, and more. You can create document libraries for storing your news stories, product catalogs, or customer information. The possibilities are endless.

# Resources

## Learn

- See the [Lotus Developer Domain](#) for more information on the Domino server.
- Find more information about [Application Developer](#) or [WebSphere Application Server](#).
- [Building a J2EE application with Lotus Domino and WebSphere Application Server](#)
- [Introduction to JavaServer Pages technology](#)
- The following IBM Redbooks provide more in-depth coverage of JSP pages and Domino development:
  - [Lotus Domino Release 5.0: A Developer's Handbook](#)
  - [Domino and WebSphere Together Second Edition](#)
  - [Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java](#)
  - [Design and Implement Servlets, JSPs, and EJBs for IBM WebSphere Application Server](#)
- For a great book on Web programming, read *Domino 5 Web Programming with XML, Java, and JavaScript* by Randall A. Tamura
- 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.

## Discuss

- [Participate in the discussion forum for this content](#).

# About the author

## George Brichacek

George Brichacek is a Technology Advocate with IBM Lotus Software, IBM Software Group, focused on application development and enterprise integration. He has presented and developed several memorable demos for the Lotus Masters Broadcast, as well as for customers, and for internal field enablement. He works with Lotus

product management to bring the latest and greatest technologies on Lotus application development and enterprise integration to you. During his spare time, he developed and supported a Lotus Notes utility called CalPrint. George resides in the Chicago area. You can reach him at [george\\_brichacek@us.ibm.com](mailto:george_brichacek@us.ibm.com).