STELLENT™

Java Server Page and JavaBean Guide

EJB-EN1-610

© 1996-2002 Stellent, Inc. All rights reserved.
No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system without written permission from the owner, Stellent, Inc., 7777 Golden Triangle Drive, Eden Prairie, Minnesota 55344 USA. The copyrighted software that accompanies this manual is licensed to the Licensee for use only in strict accordance with the Software License Agreement, which the Licensee should read carefully before commencing use of this software.

Stellent, the Stellent logo, Stellent Content Server, Stellent Content Management, Stellent Content Publisher, Stellent Dynamic Converter, and Stellent Inbound Refinery are trademarks of Stellent, Inc. in the USA and other countries.

Adobe, Acrobat, the Acrobat Logo, Acrobat Capture, Distiller, Frame, the Frame logo, and FrameMaker are registered trademarks of Adobe Systems Incorporated.
ActiveIQ is a trademark of ActiveIQ Technologies, Incorporated. Portions Powered by Active IQ Engine.
BEA WebLogic Personalization Server is a trademark of BEA Systems, Inc.
HP-UX is a registered trademark of Hewlett-Packard Company
IBM, Informix, and WebSphere are registered trademarks of IBM Corporation.
Kofax is a registered trademark, and Ascent and Ascent Capture are trademarks of Kofax Image Products.
Linux is a registered trademark of Linus Torvalds.
Microsoft is a registered trademark, and Windows, Word, and Access are trademarks of Microsoft Corporation.
MrSID is property of LizardTech, Inc. It is protected by U.S. Patent No. 5,710,835. Foreign Patents Pending.
Oracle is a registered trademark of Oracle Corporation.
Portions Copyright © 1991-1997 LEAD Technologies, Inc. All rights reserved.
Portions Copyright © 1990-1998 Handmade Software, Inc. All rights reserved.
Portions Copyright © 1988, 1997 Aladdin Enterprises. All rights reserved.
Portions Copyright © 1997 Soft Horizons. All rights reserved.
Portions Copyright © 1999 ComputerStream Limited. All rights reserved.
Portions Copyright © 1995-1999 LizardTech, Inc. All rights reserved.
Red Hat is a registered trademark of Red Hat, Inc.
Sun is a registered trademark, and Solaris, iPlanet, Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc.
Sybase is a trademark of Sybase, Inc.
UNIX is a registered trademark of The Open Group.
Verity is a registered trademark of Verity, Incorporated.

All other trade names are the property of their respective owners.

Table of Contents

Chapter

1

OVERVIEW

INTRODUCTION

The information contained in this guide is based on Stellent™ Content Server 6.1. The information is subject to change as the product technology evolves and as hardware and operating systems are created and modified.

Due to the technical nature of browsers, web servers, and operating systems, Stellent, Inc. cannot warrant compatibility with all versions and features of third-party products.

This chapter contains these topics:

- ❖ About this Guide
- ❖ Stellent Product Distinctions
- ❖ If You Need Assistance

About This Guide

This guide provides information on accessing Stellent services from a Java Server Page. Stellent Content Server core functionality can be accessed from a JSP running in Stellent Content Server, from a JSP through the Stellent Content Server JavaBean, or from a JSP through the Stellent Content Server Enterprise JavaBean (EJB) deployed on your J2EE application server. In addition, the Stellent JSP Tag Libraries are provided to simplify JSP coding and to easily access content server services.

Audience

This guide is intended for application developers who need to access Stellent Content Server functions from a JSP.

Conventions

The following conventions are used throughout this guide:

❖ The notation **<*install_dir*>/** is used throughout this guide to refer to the location on your system where Stellent Content Server product is installed.

❖ Forward slashes (/) are used to separate the directory levels in a path name. A forward slash will always appear after the end of a directory name.

❖ Notes, technical tips, important notices, and cautions use these conventions:

Symbol	**Description**
💡	This is a note. It brings special attention to information.

Symbol	Description
⚙	This is a tech tip. It identifies information that can be used to make your tasks easier.
❗	This is an important notice. It identifies a required step or critical information.
✖	This is a caution. It identifies information that might cause loss of data or serious system problems.

STELLENT PRODUCT DISTINCTIONS

In this guide, the term *content server* is used generically to refer to both the content server and the Collaboration Server. The following table lists the distinctions of these two Stellent content management solutions:

Stellent Content Management Product and Feature Distinction

Product	Description
Stellent Content Server	A fully functional content management system providing end-to-end content management and personalized delivery of that content.
Stellent Collaboration Server	A fully functional content management system providing end-to-end content management and personalized delivery of that content. Additionally, a Stellent Collaboration Server license enables project-level security for collaborative authoring environments.

IF YOU NEED ASSISTANCE

The Stellent family of products is backed by a full range of support options to meet every business need. The service philosophy is to keep your Stellent environment fully operational by providing the best information and solutions available. The Stellent product support team consists of highly trained product engineers who excel at resolving complex technical issues. Every customer inquiry is tracked and managed through automated systems.

Important: The support options that are available for specific systems may vary, depending on the applicable service and maintenance agreements. Please refer to your contract for the support details for your Stellent system.

Support Options

You can choose from the following three support programs offered by Stellent:

❖ **Standard Maintenance and Support Program:** The standard support program is available during standard business hours domestically and internationally (Monday through Friday from 8 am to 5 pm for every time zone). It provides telephone and e-mail support for troubleshooting, bug fixes, call escalation, modifications, enhancements, and updates.

❖ **SDK Developer Support Program:** The SDK support program is available Monday through Friday from 8 am to 5 pm (Central Time in the USA, which is -6 hours from GMT). It provides telephone and e-mail support for customers who wish to use the Software Developer's Kit (SDK) to customize their Stellent systems.

❖ **Extended Support Program:** The extended support program provides the standard support services 24 hours a day and 7 days a week.

Note: Value Added Resellers (VARs) and Original Equipment Manufacturers (OEMs) may have different support programs in place.

Before Contacting Support

When you call or send e-mail, please provide the following information:

- ❖ Nature and severity of the problem.
- ❖ Stellent product and version.
- ❖ Serial number of the registered Stellent product.
- ❖ Operating system and version.
- ❖ Name and telephone number of the person the support engineers should contact if they need to call back.

In addition, depending on the situation, it may be helpful to know the following:

- ❖ Database type and version.
- ❖ Web browser type and version.
- ❖ Web server type and version.

Telephone

Technical support is available from the Support Hotline at 1-888-688-TECH (1-888-688-8324). The Support Hotline is accessible toll-free world-wide.

E-Mail

The Stellent support e-mail address is *support@stellent.com*. It is available for all technical support questions.

Internet

Technical support is also available through the Internet at http://support.stellent.com. You will be prompted for a username and password. To obtain a username and password, contact the Support Hotline at 1-888-688-TECH (1-888-688-8324).

Chapter

2

INTEGRATING JAVA SERVER PAGES

INTRODUCTION

This chapter contains these topics:

- ❖ Conceptual Overview
- ❖ Integration Methods

CONCEPTUAL OVERVIEW

This guide discusses how to access Stellent Content Server core functionality from a Java Server Page using these methods:

- ❖ From a JSP running in Stellent Content Server.
- ❖ From a JSP through the Stellent Content Server JavaBean.
- ❖ From a JSP through the Stellent Content Server Enterprise JavaBean (EJB) deployed on your J2EE application server.

Additionally, this guide provides the Stellent JSP Tag Libraries to simplify JSP coding and to easily access content server services. By using the Stellent JSP Tag Library, application developers can focus on presentation issues rather than being concerned with how to access Stellent Content Server services.

INTEGRATION METHODS

This section details the methodologies that can be employed to access content server core functionality from a Java Server Page and includes these topics:

- ❖ Running a JSP in Stellent Content Server.
- ❖ Running the Stellent JavaBean on a JSP Server
- ❖ Running the Stellent EJB on a J2EE App Server.

Note: Refer to *Installing and Configuring Stellent Portlets on WebLogic Portal Server* for information on integrating Stellent Portlets with BEA WebLogic Portal Server to access Stellent Content Server.

Note: Refer to *Stellent EJB Integration for WebLogic Personalization Server* for information on integrating a deployed Stellent ContentServerBean EJB with BEA WebLogic Personalization Server and configuring a portal to access Stellent Content Server.

Note: Refer to *Installing and Configuring Stellent Portlets on WebSphere Portal Server* for information on integrating Stellent Portlets with IBM WebSphere Portal Server to access Stellent Content Server.

Running a JSP in Stellent Content Server

You can access content server core functionality from a JSP running in content server. With this configuration, you can execute content server services and IdocScript predefined variables used on a JSP checked into a content server instance.

This configuration uses the Apache Jakarta Tomcat Servlet/JSP Server running inside Stellent Content Server in a seamless integration with the Stellent Tomcat Integration JavaBean. This JavaBean, customized for Tomcat Server, provides the public class *ServerBean* as the main entry point to interact with Stellent Content Server from a JSP running inside content server.

Important: JSP pages can only execute IdocScript functions when the JSP page is being served on the content server. JSP pages served on a separate JSP server do not have this functionality. In those cases, checking a JSP page into the content server provides revision control but will not provide dynamic execution of IdocScript functions on the presentation tier (JSP server).

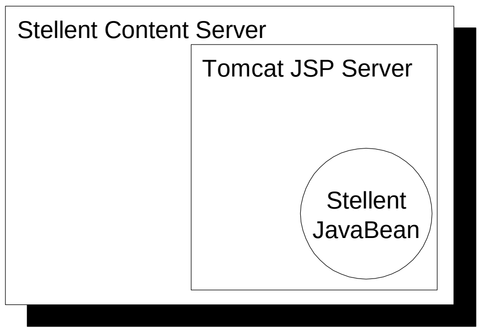


Figure 2-1 Access services from a JSP running in Stellent Content Server.

Note: For additional information, see the chapter *Running a JSP in Stellent Content Server*.

Note: Refer to the JavaDoc API in the */docs* sub-directory on the Stellent Content Server CD-ROM for information on the Class/Interface, Field, and Method descriptions of the ServerBean.

Running the Stellent JavaBean on a JSP Server

Stellent Content Server core functionality can be accessed from a JSP through the Stellent Content Server JavaBean. The Stellent Content Server JavaBean is called *IdcServerBean* and is accessed directly from a Java application or Java Server Page.

The IdcServerBean provides a direct entry point to interact with the content server. This direct integration method using the IdcServerBean eliminates the need for a socket-based interface while still enabling access to all Stellent Content Server core capabilities.

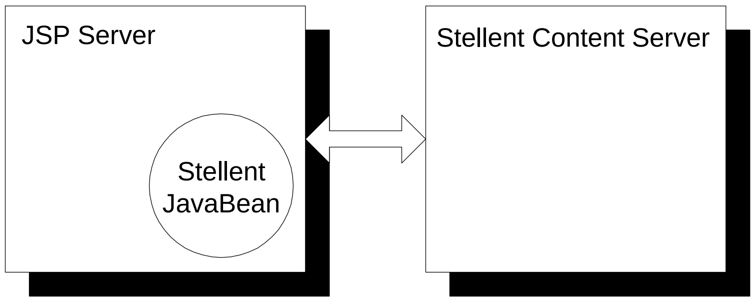


Figure 2-2 Access services from a JSP through the Stellent Content Server JavaBean

Note: For additional information, see the chapter *Running the Stellent JavaBean on a JSP Server*.

Note: Refer to the JavaDoc API in the */docs* sub-directory on the Stellent

Content Server CD-ROM for information on the Class/Interface, Field, and Method descriptions of the IdcServerBean.

Running the Stellent EJB on a J2EE App Server

Stellent Content Server core functionality can be accessed from a JSP through the Stellent Content Server Enterprise JavaBean (EJB) deployed on your J2EE application server.

The Stellent Content Server Enterprise JavaBean (EJB) is called *ContentServerBean.* The ContentServerBean EJB can be deployed in any J2EE compliant Enterprise JavaBean container such as WebLogic, WebSphere, or EAServer.

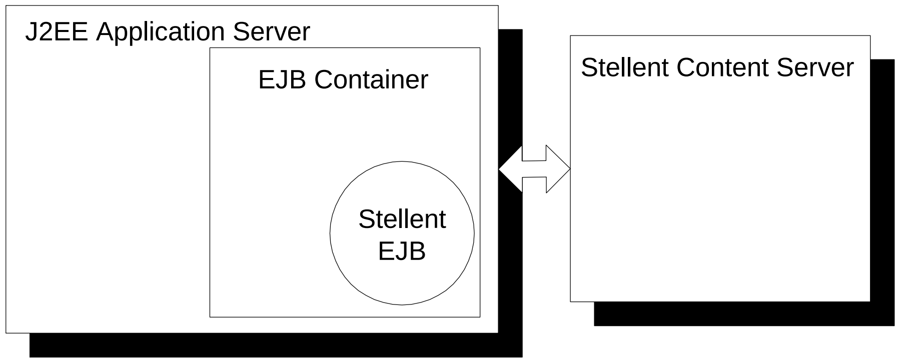


Figure 2-3 Access services from a JSP through the Stellent Content Server Enterprise JavaBean (EJB) deployed on your J2EE application server.

Note: For additional information, see the chapter *Running the Stellent EJB on a J2EE App Server*.

Note: Refer to the JavaDoc API in the */docs* sub-directory on the Stellent Content Server CD-ROM for information on the Class/Interface, Field, and Method descriptions of the ContentServerBean EJB.

Chapter

3

RUNNING A JSP IN STELLENT CONTENT SERVER

INTRODUCTION

This chapter contains these topics:

- ❖ Understanding the Architecture
- ❖ Enabling Stellent JSP Support
- ❖ Disabling Stellent JSP Support
- ❖ Troubleshooting
- ❖ Administration Notes

UNDERSTANDING THE ARCHITECTURE

Running a Java Server Page in Stellent Content Server involves the use of Apache Jakarta Tomcat Servlet/JSP Server in a seamless integration with the Stellent Tomcat Integration JavaBean to access the content and content management functions within Stellent Content Server. This JavaBean, customized for Tomcat Server, provides the public class *ServerBean* as the main entry point to interact with Stellent Content Server from a Java Server Page.

The Apache Jakarta Tomcat Server is a free, open-source server of Java Servlet and Java Server Pages that is run inside of the content server when the feature is enabled. The integration of Tomcat Server with Stellent Content Server provides the benefit of increased performance for content delivery.

Running a Java Server Page in Stellent Content Server enables developers to access and modify Stellent Content Server content, ResultsSets, personalization and security definitions, and predefined variables and configuration settings through Java Server Pages rather than through standard Stellent component architecture. Stellent services and IdocScript functions can also be executed from JSP pages which reside as executable content in the content server.

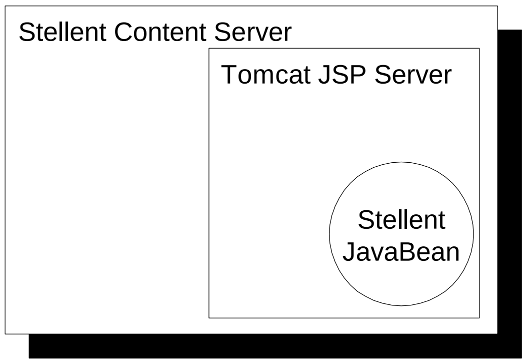


Figure 3-1 Access services from a JSP running in Stellent Content Server.

ENABLING STELLENT JSP SUPPORT

Prerequisites

These components are required:

- ❖ Java 2 SDK 1.3.1
- ❖ Stellent Content Server 6.1
  Refer to the *Stellent Content Server Installation Guide*.
- ❖ Database: SQL or Oracle
  Refer to the *Stellent Content Server Installation Guide*.

Important: You cannot use JSP support with a Microsoft Access database because Access works only with Microsoft's JVM. Refer to the content server installation guide and online help for information on database configurations.

Note: Prior to the release of Stellent 5.1, Stellent JSP support was implemented as a component. Starting with Stellent 5.1, this function has been integrated into the content server application. This documentation applies only to the Stellent Content Server version 5.1 through version 6.

Configuration

Follow these steps to enable and configure JSP support:

- ❖ Create security group.
- ❖ Create the Java environment.
- ❖ Enable JSP support and security groups.

Create Security Group

In Stellent Content Server, create a new security group to be used for JSP pages (called “jsp” in the steps below). This security group should be restricted to developers.

Note: This step is not required, but is recommended for developer convenience. Any security groups to be enabled for JSP must be specified in step 6.

1. Display the User Admin screen.
2. Select **Security—Permissions by Group**.
3. Click **Add Group**.
4. Enter **jsp** as the group name, enter a description, and click **OK**.
5. Select the *Admin* role and click **Edit Permissions**.
6. Assign Admin permission to the *Admin* role and any developer roles and click **OK**.
7. Select each non-admin role, click **Edit Permissions**, assign Read permission, and click **OK**.
8. Click **Close** on the Permissions By Group screen.
9. Close the User Admin screen.

Create Java Environment

1. If you do not have the Java 2 SDK installed on your system, run the installation program, which is located in the 3rdParty/<*platform*>/ directory.
   - ❖ **Windows:** j2sdk-1_3_1_01-win.exe
   - ❖ **Solaris:** j2sdk-1_3_1_01-solsparc.sh
   - ❖ **HP-UX:** sdk_13101os11.depot

- **Linux:** j2sdk-1_3_1_01-linux-i386-rpm.bin

Note: You can install the SDK to any directory you choose, but you will need to copy the installed Java 2 SDK directory to the <install_dir>/shared/ directory.

2. Copy the Java 2 SDK directory to the *<install_dir>/shared/* directory.

   For example:

   ```
   C:/stellent/shared/
   ```

3. Rename the Java 2 SDK directory to *j2sdk*.
4. Open the *<install_dir>/bin/intradoc.cfg* file in a text-only editor.

   For example:

   ```
   C:/stellent/bin/intradoc.cfg
   ```

5. Edit the JvmCommandLine entry to define the path to the v1.3 JVM and ensure that the comment tag is removed from the entry (see figure 3-2).

   For example:

   ```
   JvmCommandLine=<jvm_path> -cp $CLASSPATH $STARTUPCLASS
   ```

Note: The <*jvm_path*> entry should define the complete directory path of your JVM, such as <*install_dir*>/shared/j2sdk/bin/java.

6. Save and close the *intradoc.cfg* file.

```
#Additional Variables
JvmCommandLine=C:/stellent/shared/j2sdk/bin/java -cp
$CLASSPATH $STARTUPCLASS
```

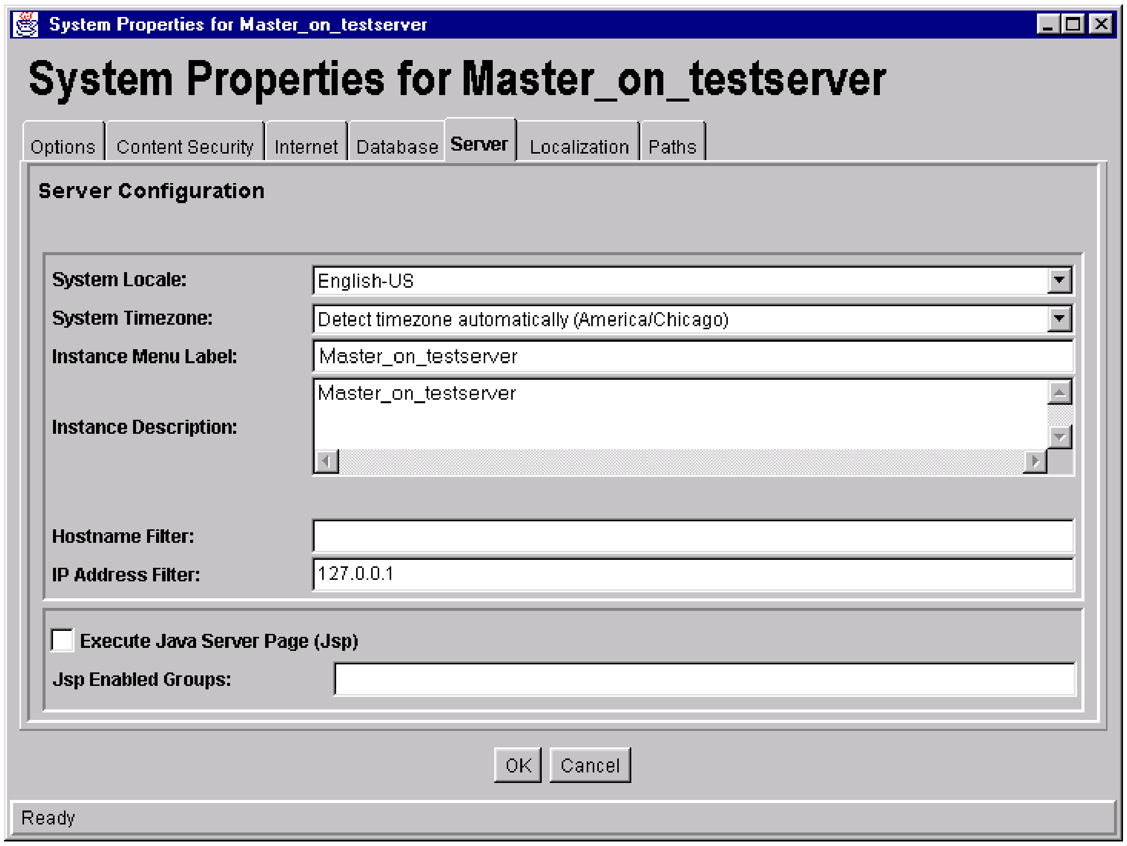
Figure 3-2 JvmCommandLine entry.

Enable JSP Support and Security Groups.

JSP support and security groups can be enabled using System Properties or by editing the configuration file. It is recommended that JSP support be enabled using System Properties

System Properties

The System Properties can be used to enable JSP support and define security groups (see figure 3-3).



System Properties for Master_on_testserver

Server Configuration

System Locale:	English-US
System Timezone:	Detect timezone automatically (America/Chicago)
Instance Menu Label:	Master_on_testserver
Instance Description:	Master_on_testserver
Hostname Filter:	
IP Address Filter:	127.0.0.1

☐ Execute Java Server Page (Jsp)

Jsp Enabled Groups:

Ready

Figure 3-3 System Propertiesr

Follow these steps to enable JSP support and define security groups using System Properties:

1. .Select **Start—Programs—Stellent Content Server—*instance_name*—Utilities—System Properties**.
2. Log in as the system administrator.
3. Enable the *Execute Java Server Page* checkbox.
4. Enter the security groups to be enabled in the *Jsp Enabled Groups* field. For example:: jsp,group1
5. Click **OK**.
6. Close System Properties.
7. Restart the content server.

Edit the Configuration File

Follow these steps to enable JSP support and define security groups by editing he configuraton file:

1. Open the *<install_dir>/config/config.cfg* configuration file in a text-only editor (see figure 3-4), or display the General Configuration page of the Admin Server.

   For example:

   ```
   C:/stellent/config/config.cfg
   ```
2. Edit the file by adding the **IsJspServerEnabled** parameter and setting it to `true`.

   ```
   IsJspServerEnabled=true
   ```
3. Enter the security groups to be enabled for JSP.

   ```
   JspEnabledGroups=jsp,group1
   ```

4. **Optional:** Enter the search sequence for the index page. The default value is `index.html,index.htm,index.jsp`.

   ```
   JspDefaultIndexPage=index.html,index.htm,index.jsp
   ```

5. **Optional:** Define the query that will find the web application archiver (.war file) in the content server. The default value is `dExtension <Matches> war`

   ```
   JspAdminQuery=dExtension <Matches> war
   ```

6. **May Be Required:** If the content server is running on Windows 2000/NT with SQL Server, open the <*install_dir*>/config/config.cfg file in a text editor and change the **JdbcDriver** entry to:

   ```
   JdbcDriver=sun.jdbc.odbc.JdbcOdbcDriver
   ```

```
#Added JSP Variables
IsJspServerEnabled=true
JspEnabledGroups=jsp
JspDefaultIndexPage=index.html,index.htm,index.jsp
JspAdminQuery=dExtension <matches> war
```

Figure 3-4 Configuration file with JSP variables defined.

7. Save and close the *config.cfg* file, or click **Save** on the General Configuration page of the Admin Server.

8. Restart the content server.

Loading Example Pages

Example pages can be loaded into the content server by checking in the .war file or by using the Batch Loader.

Check In .war File

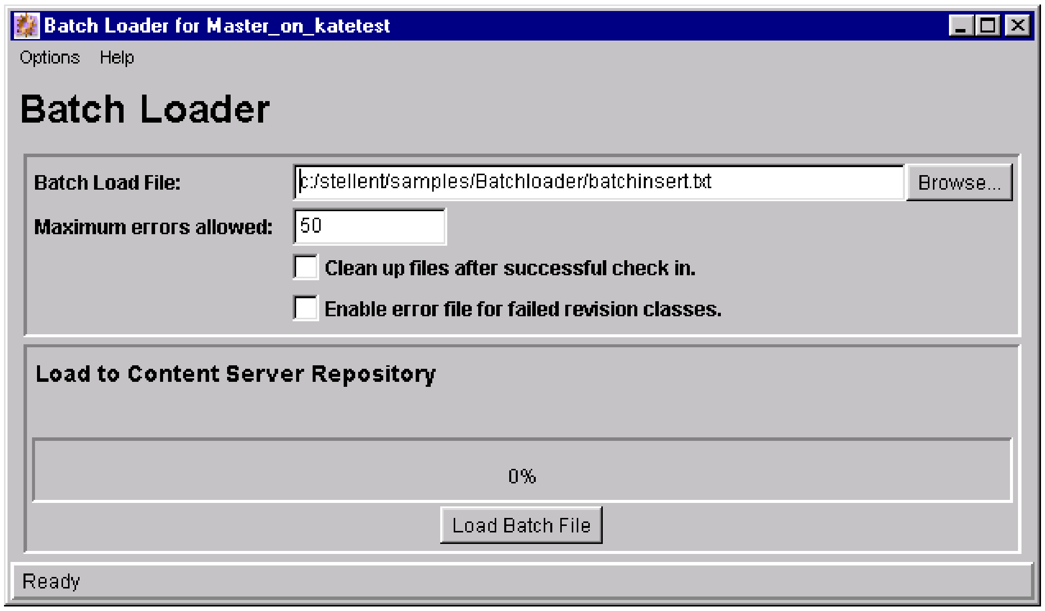
Load example pages into the content server by checking in the .war file:

1. Check the <*install_dir*>/samples/JspServer/tomcatexample.war file into the content server.
2. Start the example from the Administration page.

   The index page links to the other sample JSP pages, which you can use as a starting point for customizing the content server.

Batchload JSP Files

The Batch Loader can be used to check the example JSP files into the content server (see figure 3-5).


Batch Loader for Master_on_katetest
Options
Help
Batch Loader

Batch Load File: c:/stellent/samples/Batchloader/batchinsert.txt

Maximum errors allowed: 50

[ ] Clean up files after successful check in.

[ ] Enable error file for failed revision classes.

Browse...
Load to Content Server Repository
0%
Load Batch File
Ready

Figure 3-5 Batch Loader

Follow these steps to check files into the content server:

1. .Select **Start—Programs—Stellent Content Server—*instance_name*—Utilities—Batch Loader**.

2. Log in as the system administrator.
3. Click **Browse** and navigate to the <*install_dir*>/samples/JspServer/ directory.
4. Select *jspbatchinsert.txt* and click **Open**.
5. Click **Load Batch File**.
6. When batch loading is complete, close the Batch Loader screen.
7. In a web browser, enter the following URL:

   ```
   <SCSRootURL>/groups/jsp/documents/adacct/index.jsp
   ```

   The index page links to the other sample JSP pages, which you can use as a starting point for customizing the content server.

DISABLING STELLENT JSP SUPPORT

JSP support can be disabled using System Properties or by editing the configuration file. It is recommended that JSP support be disabled using System Properties

System Properties

Follow these steps to disable JSP support using System Properties:

1. .Select **Start—Programs—Stellent Content Server—*instance_name*—Utilities—System Properties**.
2. Log in as the system administrator.
3. Clear the *Execute Java Server Page* checkbox.
4. Click **OK**.
5. Close System Properties.

6. Restart the content server.

Edit the Configuration File

Follow these steps to disable JSP support by editing he configuraton file:

1. In the <*install_dir*>/config/config.cfg file, change the **IsJspServerEnabled** setting to `false`:

   ```
   IsJspServerEnabled=false
   ```

2. Restart the content server.

The Tomcat provider will be disabled automatically.

TROUBLESHOOTING

❖ If the JSP Server is started correctly, “Apache-Tomcat 4.0” appears in the content server output. If the content server has been started as a service, you can check the output from the Admin Server by clicking the instance button and then clicking **View Server Output**.

❖ If the following error message appears when the content server is started from console window, you are trying to use JSP support with an Access database:

  ```
  Unable to load provider class intradoc.dao.DaoWorkspace
  for SystemDatabase. Unable to instantiate java class
  code for ’intradoc.dao.DaoWorkspace’ at location
  ’intradoc.dao.DaoWorkspace’ with default location at
  ’intradoc.dao.DaoWorkspace’.com/ms/com/ComException
  ```

  You cannot use JSP support with a Microsoft Access database because Access works only with Microsoft’s JVM. Refer to the content server installation guide and online help for information on database configurations.

Administration Notes

If you have a standard installation of Stellent, reference the Stellent home page at (http://<stellent_dir>/stellent).

Note: Refer to the *Stellent Content Server System Administration Guide* for additional information,

Note: Refer to the JavaDoc API in the */docs* sub-directory on the Stellent Content Server CD-ROM for information on the Class/Interface, Field, and Method descriptions of the ServerBean.

Note: This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

Chapter

4

Running the Stellent JavaBean on a JSP Server

Introduction

This chapter provides the procedures needed to run the Stellent JavaBean on a JSP server and contains these topics:

❖ Understanding the Architecture

❖ Calling Services from a JSP

❖ Sample JSP Code

UNDERSTANDING THE ARCHITECTURE

Stellent Content Server core functionality can be accessed from a JSP through the Stellent Content Server JavaBean. The Stellent Content Server JavaBean is called *IdcServerBean* and is accessed directly from a Java application or Java Server Page.

The IdcServerBean provides a direct entry point to interact with the content server. This direct integration method using the IdcServerBean eliminates the need for a socket-based interface while still enabling access to all Stellent Content Server core capabilities.

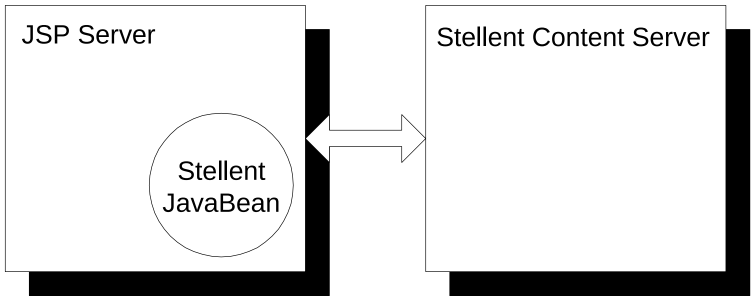


Figure 4-1 Access services from a JSP through the Stellent Content Server JavaBean

Note: Refer to the JavaDoc API in the */docs* sub-directory on the Stellent Content Server CD-ROM for information on the Class/Interface, Field, and Method descriptions of the IdcServerBean.

CALLING SERVICES FROM A JSP

Stellent Content Server services can be called from a Java Server Page through the Stellent Content Server JavaBean (*IdcServerBean)* or by using the Stellent Content Server Tag Library to reference the Stellent Content Server JavaBean.

Note: For additional information on content server services, refer to the *IdcCommand—Java Command Utility Reference Guide*.

Prerequisites

These components are required:

❖ Java 2 SDK 1.3.1

❖ Stellent Content Server 6.1
Refer to the *Stellent Content Server Installation Guide*.

❖ Database: SQL or Oracle
Refer to the *Stellent Content Server Installation Guide*.

❖ Compatible JSP Server.

❖ The idcserverbean.jar file. This file contains the Stellent Content Server JavaBean and the Stellent Content Server Tag Library.

Configuration

Follow these steps to enable and configure JSP support:

Note: This section provides the basic procedures required to access content server services form a JSP page. Because each JSP server configuration is different, your particular integration may be different.

Follow these steps to access JSP support:

1. In your JSP Server, locate the WEBAPPS directory.
2. Within the WEBAPPS directory, create a *WEB-INF* directory
3. Within the WEB-INF directory, create a *lib* sub-directory.
4. Place the idcserverbean.jar file in the *lib* sub-directory. This file contains the Stellent Content Server JavaBean and the Stellent Content Server Tag Library.
5. Create a web application descriptor file (web.xml) and place it in the WEB-INF directory.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
    2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
</web-app>
```

Figure 4-2 Web Application Descriptor File

6. Edit the web application descriptor file to define the complete web location (HttpWebRoot), the content server connection information (IdcReference), and the content server user (IdcUser).

```
<context-param>
    <param-name>HttpWebRoot</param-name>
    <param-value>http://localhost/stellent/</param-value>
</context-param>
<context-param>
    <param-name>IdcReference</param-name>
    <param-value>socket:localhost:4444</param-value>
</context-param>
<context-param>
    <param-name>IdcUser</param-name>
    <param-value>guest</param-value>
</context-param>
```

Figure 4-3 Edited web.xml File—Stellent Content Server Values

7. Edit the web application descriptor file to define the Uniform Resource Identifier (URI) and the location of the Tag Library Descriptor (.tld) file for the Stellent Content Server Tag Library.

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

Figure 4-4 Edited web.xml File—Stellent Tag Library Values

8. Save the web application descriptor file.

9. If required, edit the tag library properties file (see Figure 4-5) to reference the Uniform Resource Locator (URL) and the port number used by Stellent Content Server. These parameters are defaulted to *localhost* and *4444* respectively. This file is named *taglib.properties* and is located in the web application /WEB-INF/classes directory.

```
stellentURL=localhost
stellentPort=4444
```

Figure 4-5 Tag Library Properties File

10. Stop and start the JSP server.

Returning Data from a Service Call

The IdcServerBean can be used to execute a service call and then either return a file, return an html string, or return an hda string that can be parsed back into a databinder.

However, a parsing error occurs because the search results are being returned as html, which the databinder cannot parse. If you set 'IsJava=1' then you will get an hda string, which can then be parsed (see Figure 4-6).

```
IdcServerBean.init("http://localserver/stellent",
"socket:localserver:4444", "sysadmin");
// Returns content server connection information and logged
in user information.
IdcServerBean.putLocal("IdcService", "PING_SERVER");
IdcServerBean.putLocal("IsJava", "1");
```

Figure 4-6 Executing PING_SERVER service.

In general, if you are having problems with the IdcServerBean, try creating a LWDataBinder object, put all your local data into it, then call:

```
bean.executeService(binder, false);
String str = bean.getResponseString();
System.out.println(str);
```

This will do a debug dump of the string you are getting back from the content server. If it is html, xml, or another format, it will become immediately apparent and this error message will be returned: "Unable to parse response.”

SAMPLE JSP CODE

This section provides example JSP code and includes two sample searches:

❖ Search for Titles

❖ Search for Keyword

Note: Also see the *Example JSP Page* (page 5-70).

Search for Titles

This search generates a list of titles that the guest user can access (public) and that are checked in as ADDC.

```
<%@ taglib uri = "stellent.tld" prefix="scs" %>
<%
String userName = "guest";
String queryText = “dDocType <Matches> `ADDC`";
%>
<scs:search queryText="<%=queryText%>"
idcUser="<%=userName%>" />
<ul>
  <scs:resultSet name="SearchResults">
    <li><scs:getValue name="dDocTitle"/>
  </scs:resultSet>
</ul>
```

Figure 4-7 Sample Code for JSP—Search for Titles

Search for Keyword

This search retrieves content items that contain the defined keyword in the text.

```
<%
String userName = "guest";
// Get text for full text search query.
String searchText = request.getParameter("SEARCH_TEXT");
%>
<form method="GET" action="?SEARCH_TEXT=<%=searchText%>">
  <input name="SEARCH_TEXT" type="text"><input
type="submit" value="Go">
</form>
<ul>
<%
// Seaches for keyword
if(searchText!=null && !searchText.equals("")){
%>
<scs:search queryText="<%=searchText%>"
idcUser="<%=userName%>" />
<scs:resultSet name="SearchResults">
    <li> <scs:getValue name="dDocTitle"/>
</scs:resultSet>
<%
}
%>
```

Figure 4-8 Sample Code for JSP—Search for Keyword

ADMINISTRATION NOTES

If you have a standard installation of Stellent, reference the Stellent home page at (http://<stellent_dir>/stellent).

Note: For additional information, refer to the *Stellent Content Server System Administration Guide*.

Note: This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

Chapter

5

STELLENT JSP TAG LIBRARIES

INTRODUCTION

The Stellent JSP Tag Library is provided to simplify Java Server Page development. By using the Stellent JSP Tag Library, application developers can focus on presentation issues rather than being concerned with how to access Stellent Content Server services.

This chapter contains these topics:

- ❖ Understanding the JSP Tag Library
- ❖ Stellent IdcTag Library
- ❖ Stellent Content Server Tag Library
- ❖ Example JSP Page

UNDERSTANDING THE JSP TAG LIBRARY

The Stellent Tag Libraries

Stellent provides these two tag libraries to help developer write JSP files that access Stellent Content Server core services:

- Stellent IdcTag JSP Tag Library. This tag library contains five tags and is used on Java Server Pages served by Tomcat Server running inside Stellent Content Server in a seamless integration with the Stellent Tomcat Integration JavaBean.
- Stellent Content Server Tag Library. This tag library contains eleven tags and is used on Java Server Pages served by a JSP Server to access content server services through the Stellent Content Server JavaBean.

Note: For additional information on content server services, refer to the *IdcCommand—Java Command Utility Reference Guide*.

Understanding Tags

A JSP Tag is used on a JSP page to invoke a custom action. These custom actions encapsulate recurring tasks so that they can be reused across more than one application. A collection of custom tags is called a JSP Tag Library.

Tags from the Stellent JSP Tag Library reference a Stellent JavaBean as a wrapper to access the Stellent Content Server core functionality. Depending on the integration methodology used, this can be a JavaBean seamlessly integrated with Tomcat Server, a JavaBean providing a direct entry point to interact with the content server, or an Enterprise JavaBean (EJB) deployed on an application server.

Associated Reference Files

These associated files reference important naming and directory information:

- ❖ taglib.properties
- ❖ web.xml

The tag library properties file (see Figure 5-1) is a text file that stores the Uniform Resource Locator (URL) and the port number used by Stellent Content Server. These parameters are defaulted to *localhost* and *4444* respectively. This file is named *taglib.properties* and is located in the web application /WEB-INF/classes directory.

```
stellentURL=localhost
stellentPort=4444
```

Figure 5-1 Tag Library Properties File

The web application descriptor file (see Figure 5-2) is an XML file that defines the Uniform Resource Identifier (URI) and the location of the Tag Library Descriptor (.tld) file for the tag library. This file is named *web.xml* and is located in the web application /WEB-INF directory.

```
<web-app>
 <taglib>
    <taglib-uri>/stellent.tld</taglib-uri>
    <taglib-location>/WEB-INF/stellent.tld
    </taglib-location>
 </taglib>
 </web-app>
```

Figure 5-2 Web Application Descriptor File

Declaring Tag Libraries

To use the Stellent JSP Tag Library from a Java Server Page, the tag library must be referenced using one of these coding methods:

- Call the Tag Library Descriptor of an unpackaged JSP Tag Library from a Java Server Page. For example:
- <%@ taglib uri="/WEB-INF/stellent-taglib.tld" prefix="scs" %>
- Call the JAR file containing the JSP Tag Library from a Java Server Page. For example:

  <%@ taglib uri="/WEB-INF/stellent-tags.jar" prefix="scs" %>
- Call the JSP Tag Library by reference. This requires editing the web application descriptor and defining a short name for the Java Server Page to use for referencing this JSP Tag Library. For example:

  <%@ taglib uri="SCSTags" prefix="scs" %>

The Uniform Resource Identifier (URI) attribute uniquely identifies the tag library. The URI can define the JSP Tag Library directly or define the JSP Tag Library by reference. If the URI is defined by reference, it must be mapped to an absolute location in the *taglib-location* element of a web application deployment descriptor (see Figure 5-2). The prefix attribute defines the prefix that distinguishes tags provided by a given tag library from those provided by other tag libraries.

Tech Tip: Defining a reference to the Tag Library Descriptor file from the web application descriptor provides the convenience of not being required to change coded Java Server Pages at a later time if you decide to JAR the JSP Tag Library.

Calling the Tag Library

The Uniform Resource Identifier (URI) and the location of the Tag Library Descriptor (.tld) file is defined in the *web.xml* file (see Figure 5-2). The Tag Library Descriptor (.tld) file defines each tag and the associated attribute.

To use the Stellent JSP Tag Library, code this entry on the selected Java Server Pages:

```
<%@ taglib uri="/stellent-taglib.tld" prefix="scs" %>
```

This entry points the JSP container to the location of the Stellent JSP Tag Library.

Calling the Tag Library by Reference

Defining a reference to the *taglib* descriptor from the web applications descriptor provides the convenience of not being required to change coded Java Server Pages at a later time if you decide to JAR the JSP Tag Library.

To call the Stellent JSP Tag Library by reference, the web application descriptor must be edited and a short name for the Stellent JSP Tag Library must be provided. This short name will be used on the Java Server Page to reference the Stellent JSP Tag Library.

Follow these steps to call the JSP Tag Library by reference:

1. In a text-only editor, open the *web.xml* file found in the web application /WEB-INF directory.
2. Define a short name for the JSP page to use for referencing the Stellent Tag Library (in this example, we have used SCSTags). Add the following lines just before the </web-app> entry:

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

Notice that the stellent-taglib.tld file is referenced rather than a JAR file.

3. Save the web.xml file.
4. Code this entry on the selected Java Server Pages:

```
<%@ taglib uri="SCSTags" prefix="scs"%>
```

This entry points the JSP container to the location of the Stellent Tag Library. If the Tag Library Descriptor (.tld) file is packaged as a JAR file (for example, stellent-tags.jar), the tag library location entry could be changed to read */WEB-INF/stellent-tags.jar* without impacting the code on each Java Server Page.

STELLENT IDCTAG LIBRARY

The Stellent IdcTag JSP Tag Library is a collection of Tags that help the developer write JSP files against Stellent Content Server. The Stellent IdcTag Library is used on Java Server Pages served by the Apache Jakarta Tomcat Servlet/JSP Server running inside Stellent Content Server in a seamless integration with the Stellent Tomcat Integration JavaBean. This JavaBean, customized for Tomcat Server, provides the public class *ServerBean* as the main entry point to interact with Stellent Content Server from a Java Server Page.

Note: For additional information, see the chapter *Running a JSP in Stellent Content Server*.

Note: Refer to the JavaDoc API in the */docs* sub-directory on the Stellent Content Server CD-ROM for information on the Class/Interface, Field, and Method descriptions of the ServerBean.

The Stellent IdcTag Library contains five tags. Each TAG defines a *tagclass*, *bodycontent*, and *attributes*. Each tag attribute has three subelements: *name*, *required*, and *rtexprvalue*. The NAME element defines the parameter name, the REQUIRED element defines whether the attribute is required. The RTEXPRVALUE element is optional and indicates whether the value for the attribute can be dynamically calculated at run time.

- ❖ The Stellent IdcTag Library descriptor filename is **idctaglib.tld**.
- ❖ The Stellent IdcTag Library reference shortname is **idc**.
- ❖ The Stellent IdcTag JSP Tag Library is compliant with Sun Microsystems tag library specifications 1.1 and JSP specifications 1.2.

These are the available Stellent IdcTag JSP Tag Library tags:

Tag Name	Description
LocalEnv	Retrieve local or environment data (page 5-8).
ResInc	Evaluate an Resource Include (page 5-9).
ResultSet	Retrieve data from ResultSet (page 5-10).
Script	Evaluate an IdocScript (page 5-12).
Service	Execute a Stellent Content Server service (page 5-13).

LocalEnv

Description

Retrieve local or environment data.

- Tagclass: idctag.LocalEnv
- Bodycontent: (empty)

Tag Attributes

These are the required and optional parameters:

Name	Description
name	The key for the value to look for. ❖ required=true ❖ rtexprvalue=true

ResInc

Description

Evaluate a Resource Include.

- ❖ Tagclass: idctag.ResInc
- ❖ Bodycontent: When inc attribute is not defined, tag body will be evaluated. Tag body can contain jsp.

Tag Attributes

These are the required and optional parameters:

Name	**Description**
inc	Defines the Resource Include to be evaluated. ❖ required=false ❖ rtexprvalue=true

ResultSet

Description

Retrieve data from ResultSet.

- ❖ Tagclass: idctag.ResultSet
- ❖ Bodycontent: When fieldname and fieldIndex attributes are not defined, tag body will be evaluated. Tag body can contain jsp and IdocScript.

Tag Attributes

These are the required and optional parameters:

Name	Description
name	Name of the ResultSet. ❖ required=true ❖ rtexprvalue=true
fieldName	Name of the field to retrieve value from. ❖ required=false ❖ rtexprvalue=true
FieldIndex	Retrieve the value from this index. ❖ required=false ❖ rtexprvalue=true
IsDate	This is a date value. This attribute should be used with fieldIndex. ❖ required=false ❖ rtexprvalue=true

Name	**Description**
Iterations	Defined iterations that body would be evaluated. Note: This should be used without fieldname and fieldIndex attributes. Type: Int. ❖ required=false ❖ rtexprvalue=true
AutoIteration	Defined if ResultSet will be iterated till the end while repeated evaluate the body for each new row in ResultSet. Type: Boolean. ❖ required=false ❖ rtexprvalue=true

Script

Description

Evaluate an IdocScript.

- ❖ Tagclass: idctag.Script
- ❖ Bodycontent: When script attribute is not defined, tag body will be evaluated. Tag body can contain jsp.

Tag Attributes

These are the required and optional parameters:

Name	Description
script	Script to be evaluated. ❖ required=false ❖ rtexprvalue=true
hasIdcTag	Default as false, which indicates the entire content of script is a single IdocScript. ❖ required=false ❖ rtexprvalue=false

Service

Description

Execute a content server service.

- ❖ Tagclass: idctag.Service
- ❖ Bodycontent: When service attribute is not defined, tag body will be evaluated. Default service format is ’&’ separated name=value pair. Tag body can contain jsp.

Tag Attributes

These are the required and optional parameters:

Name	**Description**
service	Defines a service. ❖ required=false ❖ rtexprvalue=true
sepChar	Defines the separation character. Default as ’&’ ❖ required=false ❖ rtexprvalue=false
EscChar	Defines the escape character. ❖ required=false ❖ rtexprvalue=false

STELLENT CONTENT SERVER TAG LIBRARY

The Stellent Content Server Tag Library is a collection of tags to help developers write JSP files that access Stellent Content Server core functionality. The Stellent Content Server Tag Library is used on Java Server Pages served by a JSP Server to access content server services through the Stellent Content Server JavaBean. The Stellent Content Server JavaBean is called *IdcServerBean* and provides a direct entry point to interact with the content server.

Note: The Stellent Content Server Tag Library is bundled with the Stellent Content Server JavaBean in the *idcserverbean.jar* file.

Note: For additional information, see the chapter *Running the Stellent JavaBean on a JSP Server*.

Note: Refer to the JavaDoc API in the */docs* subdirectory on the Stellent Content Server CD-ROM for information on the Class/Interface, Field, and Method descriptions of the IdcServerBean.

Each TAG in the Stellent Content Server Tag Library defines a *tagclass*, *bodycontent*, and *attributes*. Each tag attribute has three subelements: name, required, and rtexprvalue. The NAME element defines the parameter name, the REQUIRED element defines whether the attribute is required. The RTEXPRVALUE element is optional and indicates whether the value for the attribute can be dynamically calculated at run time.

- The Stellent Content Server Tag Library descriptor filename is **stellent.tld**.
- The Stellent Content Server Tag Library reference shortname is **scs**.
- The Stellent Content Server Tag Library version 1.0b and 1.1b are compliant with Sun Microsystems tag library specifications 1.1 and JSP specifications 1.2.

Stellent Content Server Tag Library version 1.1b:

Tag Name	**Description**
checkin	Takes a multipart encoded request from a previous JSP page and forwards it on to the content server (page 5-18).
checkinList *(New for 1.1b)*	Obtains a list for the content currently checked out by the user (page 5-20).
checkinSelPage *(New for 1.1b)*	Obtain the metadata needed to fill in a checkin page (page 5-22).
checkout *(New for 1.1b)*	Checks out a content item by 'id' (page 5-24).
docInfo	Downloads the content information for a specific item (page 5-25).
docPage	Downloads the system metadata information for a particular content server (page 5-27).
getFile *(Version 1.0b)*	Retrieves content checked into the content server. **DEPRECATED:** Use the GetFileServlet to download binary files (page 5-29).
getHtmlConversion *(New for 1.1b)*	Obtains content checked in to the content server (page 5-33).
getValue	Retrieves the values of the properties object that corresponds to the value of propertiesName (page 5-36).
metadataField *(New for 1.1b)*	Used when looping over the result set 'DocMetaDefinition' to display a custom metadata field (page 5-38).

Tag Name	Description
optionList	Iterator tag used after a service tag executes to loop over an option list (page 5-39).
pne	Used to load or alter the PNE data for the user (page 5-40).
resultSet	Iterator tag used after a service tag executes a call against the content server, such as *search*, *docInfo*, or *workflowQueue* (page 5-45).
search	Performs a search and places the resultset into the page context (page 5-46).
service	Extends the ServiceTag class to connect to the content server (page 5-49).
subscribe *(New for 1.1b)*	Used to subscribe a specific user to a specific content item (page 5-53).
subscriptionList *(New for 1.1b)*	Obtains a list of all content that the user is currently subscribed to (page 5-55).
undoCheckout *(New for 1.1b)*	Used to undo a check out of a content item (page 5-56).
unsubscribe *(New for 1.1b)*	Used to unsubscribe a specific user to a specific content item (page 5-57).
userProfile *(New for 1.1b)*	Obtains a list of general information about the user (page 5-59).
workflowApprove *(New for 1.1b)*	Used to approve content that is in a workflow (page 5-60).
workflowQueue	Downloads the workflow in queue for the specified user (page 5-62).

Tag Name	**Description**
workflowReject *(New for 1.1b)*	Used to reject content that is in a workflow (page 5-64).
workflowRejectMail *(New for 1.1b)*	Executes a call against the content server to reject a content item that is inside a workflow (page 5-66).
workflowStepInfo *(New for 1.1b)*	Executes a call against the content server and returns the current workflow step information for the content item (page 5-68).

checkin

Description

Takes a multipart encoded request from a previous jsp page and forwards it on to the content server. This tag can be used to check in new content, check in previously checked out content, or perform any other file transfer service that the content server uses (CHECKIN_NEW, CHECKIN_SEL, etc.).

- ❖ Tagclass: idcbean.taglib.CheckinTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	Description
fileService	The name of the IdcService to use. Defaults to 'CHECKIN_NEW' ❖ required=true ❖ rtexprvalue=true
idcUser	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true

Name	Description
debug	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true

checkinList

Description

Obtains a list for the content currently checked out by the user, and places it into the pageContext as the LWResultSet 'CHECKIN_LIST'. It also places name-value pairs from the result into a Properties object named 'LocalData'.

- ❖ Tagclass: idcbean.taglib.CheckinListTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  idcbean.data.LWResultSet: CHECKIN_LIST
  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	Description
userOnly	If TRUE, only the content checked out by this user is returned. ❖ required=false ❖ rtexprvalue=true
idcUser	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true

Name	**Description**
debug	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true

checkinSelPage

Description

This tag can be used in a similar fashion to the docInfo or docPage tags to obtain the metadata needed to fill in a checkin page. The data from the response, including result sets and option lists, is stored in the binder 'CheckinSel' for retrieval on the JSP pages.

❖ Tagclass: idcbean.taglib.CheckinSelPageTag

❖ Bodycontent: empty

❖ Stores in pageContext:

  idcbean.data.LWResultSet: DOC_INFO
  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	Description
id	The 'dID' value for the content item. This is the generated content item revision ID. ❖ required=true ❖ rtexprvalue=true
idcUser	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true

Name	Description
debug	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true

checkout

Description

This tag can be used to check out a content item by ’id’. It will throw an error if the ’id’ does not point to the most recent revision.

❖ Tagclass: idcbean.taglib.CheckoutTag

❖ Bodycontent: empty

❖ Stores in pageContext:

  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	Description
id	The ’dID’ value for the content item. This is the generated content item revision ID. ❖ required=true ❖ rtexprvalue=true
idcUser	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true

docInfo

Description

This tag will download the content information for a specific item. If the value for 'dID' is specified, it will obtain the metadata info for that specific revision. If the 'dDocName' is passed, then the content information for the most recent revision is returned. In both cases, the resultSets for 'REVISION_HISTORY', 'WF_INFO', and 'DOC_INFO' are placed into the page attributes, and 'DOC_INFO' is made the active result set. These result sets can be looped over with the 'resultSet' tag.

- ❖ Tagclass: idcbean.taglib.DocInfoTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  idcbean.data.LWResultSet: DOC_INFO
  idcbean.data.LWResultSet: REVISION_HISTORY
  idcbean.data.LWResultSet: WF_INFO
  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	Description
id	The 'dID' value for the content item. This is the generated content item revision ID. ❖ required=false ❖ rtexprvalue=true

Name	**Description**
docName	The 'dDocName' value for the content item. This is the content item identifier (Content ID). **Note:** Do not confuse the Content ID (dDocName) with the internal content item revision identifier (dID). The *dID* is a generated reference to a specific rendition of a content item. ❖ required=true ❖ rtexprvalue=true
idcUser	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true

docPage

Description

This tag will download the system metadata information for a particular content server, including all custom option lists, all content types, all doc formats, all default security accounts, and optionally all security groups, authors, and security roles. This tag is mostly useful when executed at the beginning of a form to check in or search for content, so that the user knows which fields and options are available.

- ❖ Tagclass: idcbean.taglib.DocPageTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  idcbean.data.LWDataBinder: DocPage
  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	**Description**
includeFields	Only obtain these values for the custom metadata fields. ❖ required=false ❖ rtexprvalue=true

Name	Description
omitFields	Obtain all except these values for the custom metadata fields. ❖ required=false ❖ rtexprvalue=true
idcUser	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true

getFile

Description

DEPRECATED: Use the GetFileServlet to download binary files. The API is exactly the same, any of the parameters listed below can be passed in the URL, except for 'idcUser'. That must be in the security principal or in the session attributes as "IdcUser".

This tag can be used to obtain content checked in to the content server. These files can be the primary, alternate, or web viewable file, or a dynamically converted html file based on one of them. The dynamic conversion can only be performed on files of certain types and the conversion templates need to be properly set up for those file types.

- ❖ Tagclass: idcbean.taglib.GetFileTag
- ❖ Bodycontent: empty

Tag Attributes

These are the required and optional parameters:

Name	Description
id	The 'dID' value for the content item. This is the generated content item revision ID. ❖ required=false ❖ rtexprvalue=true

Name	Description
docName	The 'dDocName' value for the content item. This is the content item identifier (Content ID). **Note:** Do not confuse the Content ID (dDocName) with the internal content item revision identifier (dID). The *dID* is a generated reference to a specific rendition of a content item. ❖ required=false ❖ rtexprvalue=true
fileName	The suggested name for this file, if the user is prompted by the browser to download the file ❖ required=false ❖ rtexprvalue=true
getDocInfo	Defaults to FALSE. If TRUE, this tag will determine the proper 'fileName' and 'mimeType' for this item with a preliminary DOC_INFO call. ❖ required=false ❖ rtexprvalue=true

Name	Description
revisionSelection Method	The revision selection method. If the dID value is not specified, this is which revision of the content to obtain. Defaults to 'Latest'. If present, docName must be present. The value of this variable is the method used to compute a dID from the specified dDocName. Its value may be Specific, Latest, or LatestReleased. If the value is Specific, the dDocName is ignored, and dID is required and is used to get a rendition. If the value is Latest, the latest revision of the content item is used to compute the dID. If the value is LatestReleased, the latest released revision of the content item is used to compute the dID. ❖ required=false ❖ rtexprvalue=true
rendition	The content item rendition. This parameter specifies the rendition of the content item and can be set to *Primary*, *Web*, or *Alternate*. If rendition is not present, it defaults to *Primary.* If the value is *Primary*, the primary rendition of the selected revision is returned. If the value is *Web*, the web viewable rendition of the selected revision is returned. If the value is *Alternate*, the alternate rendition of the selected revision is returned. ❖ required=false ❖ rtexprvalue=true

Name	Description
doHtmlConversion	The HTML conversion option. Defaults to FALSE. If set to TRUE, it will attempt to translate this content to an html document. ❖ required=false ❖ rtexprvalue=true
mimeType	The mime type for the content item. This is used so the browser can properly handle it. ❖ required=false ❖ rtexprvalue=true
idcUser	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true

getHtmlConversion

Description

This tag can be used to obtain content checked in to the content server. These files can be the primary, alternate, or web viewable file, or a dynamicly converted html file based on one of them. The dynamic conversion can only be performed on files of certain types - and the conversion templates need to be properly set up for those file types.

- ❖ Tagclass: idcbean.taglib.GetHtmlConversionTag
- ❖ Bodycontent: empty

Tag Attributes

These are the required and optional parameters:

Name	**Description**
id	The ’dID’ value for the content item. This is the generated content item revision ID. ❖ required=false ❖ rtexprvalue=true

Name	Description
docName	The 'dDocName' value for the content item. This is the content item identifier (Content ID). **Note:** Do not confuse the Content ID (dDocName) with the internal content item revision identifier (dID). The *dID* is a generated reference to a specific rendition of a content item. ❖ required=false ❖ rtexprvalue=true
revisionSelection Method	The revision selection method. If the dID value is not specified, this is which revision of the content to obtain. Defaults to 'Latest'. If present, docName must be present. The value of this variable is the method used to compute a dID from the specified dDocName. Its value may be Specific, Latest, or LatestReleased. If the value is Specific, the dDocName is ignored, and dID is required and is used to get a rendition. If the value is Latest, the latest revision of the content item is used to compute the dID. If the value is LatestReleased, the latest released revision of the content item is used to compute the dID. ❖ required=false ❖ rtexprvalue=true

Name	**Description**
rendition	The content item rendition. This parameter specifies the rendition of the content item and can be set to *Primary*, *Web*, or *Alternate*. If rendition is not present, it defaults to *Primary*. If the value is *Primary*, the primary rendition of the selected revision is returned. If the value is *Web*, the web viewable rendition of the selected revision is returned. If the value is *Alternate*, the alternate rendition of the selected revision is returned. ❖ required=false ❖ rtexprvalue=true
coreContentOnly	Defaults to ’true’, if false the content server UI is wrapped around the converted html. ❖ required=false ❖ rtexprvalue=true
idcUser	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true

getValue

Description

This is a simple tag that can be used to pull values from the LocalData, or from a ResultSet of a previous content server call. By default, it pulls the values out of the properties object that corresponds to the value of propertiesName, which defaults to 'LocalData'.

If multiple services are called on the same JSP, and you want to obtain LocalData from each of them, you should set the 'propertiesName' parameter to something besides 'LocalData' for both the service and localData tags.

If the value is not found in the local data, it will then look for the value in the current row of the active result set. This is most useful when inside a resultSet iterator tag. Otherwise, if a result set has only one row, or if it is being iterated manually, you can also pass a value for 'resultSetName' and pull the values from the current row.

- ❖ Tagclass: idcbean.taglib.GetValueTag
- ❖ Bodycontent: empty

Tag Attributes

These are the required and optional parameters:

Name	**Description**
name	The key for the value to look for. ❖ required=false ❖ rtexprvalue=true

Name	Description
properties	The name of the properties object in the page context to look first for the value. Defaults to ’LocalData’. ❖ required=false ❖ rtexprvalue=true
resultSet	The name of the LWResultSet object to look for the value in. This defaults to the currently active result set, which is set when one begins looping with the ’resultSet’ iterator. ❖ required=false ❖ rtexprvalue=true
optionList	HHH ❖ required=false ❖ rtexprvalue=true
binder	The name of the LWDataBinder object (which holds properties and named result sets) to look in for the value. Rarely set. ❖ required=false rtexprvalue=true

metadataField

Description

This tag is used exclusively when looping over the result set ’DocMetaDefinition’ to display a custom metadata field.

- ❖ Tagclass: idcbean.taglib.MetadataFieldTag
- ❖ Bodycontent: empty

Tag Attributes

These are the required and optional parameters:

Name	**Description**
extraTagDef	Extra html to put inside the tag, such as ’onChange()’ events. This is also pulled out of the LocalData on a per metadata basis. For example, if you set "xComments:extraTagDef" to "maxlength=100" in the LocalData, it will be placed inside the tag when ’dName’ equals ’xComments’. ❖ required=false ❖ rtexprvalue=true

optionList

Description

This is an iterator tag that can be used after a service tag executes a call against the content server, to loop over an option list. Option lists are returned by very few service calls. They are typically used to format check in and search pages with the appropriate options for custom metadata fields, or for the content server lists 'docAccounts,' 'securityGroups', 'docAuthors', 'docTypes', or 'roles'.

- ❖ Tagclass: idcbean.taglib.OptionListTag
- ❖ Bodycontent: JSP

Tag Attributes

These are the required and optional parameters:

Name	Description
name	The name of the option list to iterate over ❖ required=true ❖ rtexprvalue=true
binder	The name of the LWDataBinder object to pull the Vector out of, if it was not placed directly into the page context. ❖ required=false ❖ rtexprvalue=true
maxRows	The maximum number of rows to iterate over. ❖ required=false ❖ rtexprvalue=true

pne

Description

This tag is used to load or alter the PNE data for the user, for such things as saved queries, personal Urls, and some info about how the user likes the portal to be displayed. The PNE data for user ’sysadmin’ can be found in the file:

contentserver/data/users/profiles/sy/sysadmin/pne_portal.hda

The data in this file is loaded up on most content server pages, and determines what the saved queries are, the personal URLs, and which links to show/hide in the navigation sidebar. See the ’p13n’ sample JSPs for how to use this tag.

- ❖ Tagclass: idcbean.taglib.PneTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  idcbean.data.LWResultSet: SavedQueries
  idcbean.data.LWResultSet: PersonalURLS
  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	Description
action	An integer representing the action to perform load (0), add a row (1), delete a row (2), or update a name-value pair (3). This attribute is specific to the tag itself. ❖ required=false ❖ rtexprvalue=true
request	Name-Value pairs separated by new line characters for adding new name-value pairs, or sometimes altering rows. ❖ required=false ❖ rtexprvalue=true
resultSet	The name of the result set to load or modify. ❖ required=false ❖ rtexprvalue=true
urlTitle	The user-provided “title” or link name for the saved URL. This attribute is specific to saving URLs (personal URL parameter). ❖ required=false ❖ rtexprvalue=true

Name	Description
website	The URL path. This attribute is specific to saving URLs (personal URL parameter). ❖ required=false ❖ rtexprvalue=true
queryTitle	The user provided “title” or name to give this query. This attribute is specific to saving searches (saved query parameter). ❖ required=false ❖ rtexprvalue=true
queryText	The Verity-specific query text. This attribute is specific to saving searches (saved query parameter). ❖ required=false ❖ rtexprvalue=true
sortOrder	The sort order. Allowed values are ASC (ascending) and DES (descending). This attribute is specific to saving searches (saved query parameter). ❖ required=false ❖ rtexprvalue=true
sortField	The name of the metadata field to sort on. For example, dDocTitle, dInDate, dOutDate. This attribute is specific to saving searches (saved query parameter). ❖ required=false ❖ rtexprvalue=true

Name	Description
resultCount	The number of results per page. This attribute is specific to saving searches (saved query parameter). ❖ required=false ❖ rtexprvalue=true
fromPageUrl	The parent page, typically null. This attribute is specific to saving searches (saved query parameter). ❖ required=false ❖ rtexprvalue=true
ftx	Set to '1' if a full text search is performed. This attribute is specific to saving searches (saved query parameter). ❖ required=false ❖ rtexprvalue=true
searchProviders	If using Enterprise Search, this is a list of content servers to search. This attribute is specific to saving searches (saved query parameter). ❖ required=false ❖ rtexprvalue=true
securityGroup	The security group that this query belongs to (for example, PUBLIC or SECURE), typically null. This attribute is specific to saving searches (saved query parameter). ❖ required=false ❖ rtexprvalue=true

Name	**Description**
idcUser	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true

resultSet

Description

This is an iterator tag that can be used after a service tag executes a call against the content server, such as ’search’, ’docInfo’, or ’workflowQueue’. It can be used in conjunction with the ’getValue’ tag to print values out to the JSP page, or in conjunction with Java code to manually pull values out of the LWResultSet object in the pageContext attributes.

❖ Tagclass: idcbean.taglib.ResultSetTag

❖ Bodycontent: JSP

Tag Attributes

These are the required and optional parameters:

Name	Description
name	The name of the result set to iterate over. ❖ required=true ❖ rtexprvalue=true
binder	The name of the LWDataBinder object to pull the LWResultSet out of, if it was not placed directly into the page context. ❖ required=false ❖ rtexprvalue=true
maxRows	The maximum number of rows to iterate over. ❖ required=false ❖ rtexprvalue=true

search

Description

This tag is used to execute a search and place the resultset 'SearchResults' into the page context. In the event of a multi-page search result, it will also calculate the values for 'prevStartRow', 'prevEndRow', prevPageNumber', 'nextStartRow', 'nextEndRow', and 'nextPageNumber', needed for multi-page navigation.

- ❖ Tagclass: idcbean.taglib.SearchTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  idcbean.data.LWResultSet: SearchResults
  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	**Description**
queryText	The Verity specific query text. ❖ required=true ❖ rtexprvalue=true
sortOrder	The sort order. Allowed values are ASC (ascending) and DES (descending). ❖ required=false ❖ rtexprvalue=true

Name	Description
sortField	The name of the metadata field to sort on. For example, dDocTitle, dInDate, dOutDate. ❖ required=false ❖ rtexprvalue=true
sortSpec	For sorting over multiple fields. Requires the MultiSort component to be installed. ❖ required=false ❖ rtexprvalue=true
resultCount	The number of results per page. This attribute is specific to saving searches (saved query parameter). ❖ required=false ❖ rtexprvalue=true
pageNumber	The current page number. ❖ required=false ❖ rtexprvalue=true
startRow	The first row in the search results. ❖ required=false ❖ rtexprvalue=true
endRow	The last row in the search results. ❖ required=false ❖ rtexprvalue=true

Name	**Description**
idcUser	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true

service

Description

This is the generic ServiceTag class that all tag library classes extend in order to connect to the content server. By itself it can do a great deal, but it is not particularly user-friendly. It can be used to make a generic call to the content server. By default it stores the LocalData from the request into the pageContext as a Properties object named 'LocalData'. If a value for 'resultSet' is specified, it will also save the correspondingly named LWResultSet object to the page context. If a value for 'binder' is specified, the local data and all result sets will be stored in a LWDataBinder object with that name. Alternatively, the entire string response from the request will be dumped to the page if 'isJava' is set to false.

It should be noted that most tags extend the class for this tag. As a result, all of the TLD parameters for the 'service' tag can be specified for other tags in the TLD. All tags that have the "inherited from the 'service' tag" comment in the TLD can have their XML definition modified to include any of the attributes for this tag. This is only useful for very fancy calls with the taglibraries. However, extremely fancy calls should probably be done directly with the IdcServerBean instead of a tag.

- ❖ Tagclass: idcbean.taglib.ServiceTag
- ❖ Bodycontent: JSP

Tag Attributes

These are the required and optional parameters:

Name	Description
idcUser	For connecting to the content server. The user name used to connect to the content server. This is pulled from the ServletRequest. If not yet validated, it will connect as the value specified in the init parameter ’IdcUser.’ Setting this to ’guest’ is recommended. ❖ required=false ❖ rtexprvalue=true
password	For connecting to the content server. The password used to connect to the content server. If set, this password will be validated by the content server before executing the request as that user. ❖ required=false ❖ rtexprvalue=true
idcReference	For connecting to the content server. The reference string that points to a specific content server. Defaults to ’socket:localhost:4444’. This value is loaded from the xml as the init parameter ‘IdcReference.’ ❖ required=false ❖ rtexprvalue=true

Name	Description
request	For sending data. The name-value pairs, separated by new line characters, which get loaded up into a Properties object for the request. ❖ required=false ❖ rtexprvalue=true
isJava	For sending data. Defaults to TRUE. If set to FALSE, it will not attempt to parse the response into a LWDataBinder, and will instead dump the response to the html page. ❖ required=false ❖ rtexprvalue=true
initFromRequest	For sending data. If set to TRUE, the name-value pairs from the URL will be loaded into the data for the request. ❖ required=false ❖ rtexprvalue=trur
properties	For displaying/saving data. The name of the Properties object to save the name-value pairs from the result to in the page context. Defaults to ’LocalData’. ❖ required=false ❖ rtexprvalue=true

Name	Description
resultSet	For displaying/saving data. The name of the LWResultSet from the result to save in the page context. ❖ required=false ❖ rtexprvalue=true
binder	For displaying/saving data. The name of the LWDataBinder to save the entire result to in the page context. ❖ required=false ❖ rtexprvalue=true
debud	For displaying/saving data. If set to TRUE, the response will be dumped to the standard output. ❖ required=false ❖ rtexprvalue=true
suppressError	For displaying/saving data. If set to TRUE, the tag will throw an exception if a content server error occurs. Otherwise the error is just logged. ❖ required=false ❖ rtexprvalue=true

subscribe

Description

This tag is used to subscribe a specific user to a specific content item. Any time a new revision of that content is checked in, the user will recieve notification.

- ❖ Tagclass: idcbean.taglib.SubscribeTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	Description
id	The 'dID' value for the content item. This is the generated content item revision ID. ❖ required=true ❖ rtexprvalue=true

Name	Description
docName	The 'dDocName' value for the content item. This is the content item identifier (Content ID). **Note:** Do not confuse the Content ID (dDocName) with the internal content item revision identifier (dID). The *dID* is a generated reference to a specific rendition of a content item. ❖ required=true ❖ rtexprvalue=true
email	The email address to send notifications to, defaults to the user's current email address, if known. ❖ required=false ❖ rtexprvalue=true
idcUser	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true

subscriptionList

Description

Obtains a list of all content that the user is currently subscribed to.

- ❖ Tagclass: idcbean.taglib.SubscriptionListTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  idcbean.data.LWResultSet: SUBSCRIPTION_LIST
  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	**Description**
idcUser	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true

undoCheckout

Description

This tag can be used to undo a check out of a content item.

- ❖ Tagclass: idcbean.taglib.CheckoutTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	**Description**
id	The ’dID’ value for the content item. This is the generated content item revision ID. ❖ required=true ❖ rtexprvalue=true
idcUser	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true

unsubscribe

Description

This tag is used to unsubscribe a specific user to a specific content item.

- ❖ Tagclass: idcbean.taglib.UnsubscribeTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	**Description**
id	The ’dID’ value for the content item. This is the generated content item revision ID. ❖ required=true ❖ rtexprvalue=true

Name	Description
docName	The 'dDocName' value for the content item. This is the content item identifier (Content ID). **Note:** Do not confuse the Content ID (dDocName) with the internal content item revision identifier (dID). The *dID* is a generated reference to a specific rendition of a content item. ❖ required=true ❖ rtexprvalue=true
idcUser	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true

userProfile

Description

Obtains a list of general information about the user, such as email address, full name, and locale, as well as security info such as roles, and predefined accounts (if enabled).

- ❖ Tagclass: idcbean.taglib.UserProfileTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	**Description**
idcUser	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true

workflowApprove

Description

This tag is used to approve content that is in a workflow. The content items in the workflow can be obtained via the workflowQueue tag (see page 5-62 for additional information).

❖ Tagclass: idcbean.taglib.WorkflowApproveTag

❖ Bodycontent: empty

❖ Stores in pageContext:

java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	Description
workflowName	The name of the workflow that this content item is in. ❖ required=true ❖ rtexprvalue=true
id	The ’dID’ value for the content item that is being approved or rejected with this tag. This is the generated content item revision ID. ❖ required=true ❖ rtexprvalue=true

Name	**Description**
idcUser	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true

workflowQueue

Description

This tag is used to download the workflow in queue for the specified user, and save it in the result set ’WorkflowInQueue’, which is made the active result set and can be looped over with the resultSet tag. The data in the workflow queue represents content items that require this user’s attention.

- Tagclass: idcbean.taglib.WorkflowQueueTag
- Bodycontent: empty
- Stores in pageContext:

  idcbean.data.LWResultSet: WorkflowInQueue
  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	**Description**
approvedItems	If set to FALSE, only the items that have not been acted upon by this user will be in the list. Otherwise an item will stay in the queue until the step has been completed by all reviewers, and this user’s last action will be stored in ’wfQueueActionState.’ ❖ required=false ❖ rtexprvalue=true

Name	Description
contributionItems	Set to FALSE if you do not want to see items in the list which require contribution. ❖ required=false ❖ rtexprvalue=true
reviewItems	Set to false if you do not want to see items in the list which require review (approve/reject). ❖ required=false ❖ rtexprvalue=true
idcUser	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true

workflowReject

Description

This tag is used to reject content that is in a workflow. The content items in the workflow can be obtained via the workflowQueue tag (see page 5-62 for additional information).

- ❖ Tagclass: idcbean.taglib.WorkflowRejectTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	**Description**
workflowName	The name of the workflow that this content item is in. ❖ required=true ❖ rtexprvalue=true
id	The 'dID' value for the content item that is being approved or rejected with this tag. This is the generated content item revision ID. ❖ required=true ❖ rtexprvalue=true

Name	Description
rejectMessage	The message to email to the author regarding the rejected content containing the reason for rejection. ❖ required=false ❖ rtexprvalue=true
idcUser	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the ’service’ tag. ❖ required=false ❖ rtexprvalue=true

workflowRejectMail

Description

This tag will execute a call against the content server and reject a content item that is inside a workflow. The 'DOC_INFO' result set is stored in the page context for retrieval on JSP pages.

- ❖ Tagclass: idcbean.taglib.WorkflowRejectMailTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  idcbean.data.LWResultSet: DOC_INFO
  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	Description
workflowName	The name of the workflow that this content item is in. ❖ required=true ❖ rtexprvalue=true
id	The 'dID' value for the content item that is being rejected. This is the generated content item revision ID. ❖ required=true ❖ rtexprvalue=true

Name	**Description**
idcUser	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true

workflowStepInfo

Description

This tag will execute a call against the content server and return the current workflow step information for the content item. All result sets for this request will be stored in the binder 'WorkflowStepInfo'.

- ❖ Tagclass: idcbean.taglib.WorkflowStepInfoTag
- ❖ Bodycontent: empty
- ❖ Stores in pageContext:

  idcbean.data.LWDataBinder: WorkflowStepInfo
  java.util.Properties: LocalData

Tag Attributes

These are the required and optional parameters:

Name	**Description**
workflowStepName	The name of the workflow step. ❖ required=true ❖ rtexprvalue=true
workflowName	The name of the workflow that this content item is in. ❖ required=false ❖ rtexprvalue=true

Name	**Description**
id	The 'dID' value for the content item in the workflow. This is the generated content item revision ID. ❖ required=true ❖ rtexprvalue=true
docName	The 'dDocName' value for the content item in the workflow. This is the content item identifier (Content ID). **Note:** Do not confuse the Content ID (dDocName) with the internal content item revision identifier (dID). The *dID* is a generated reference to a specific rendition of a content item. ❖ required=true ❖ rtexprvalue=true
idcUser	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true
debug	Inherited from the 'service' tag. ❖ required=false ❖ rtexprvalue=true

EXAMPLE JSP PAGE

This section provides a complete Java Server Page that includes code to retrieve content item information.

Note: Also see the *Sample JSP Code* (page 4-8).

```
<%@ taglib uri = "/WEB-INF/stellent.tld" prefix="scs" %>
<%@ page errorPage="errorpage.jsp" %>

<html><head><title>
Doc Info Test Page
</title></head>
<body>
<h1>Doc Info Test Page</h1>
<pre>
You will need to modify the values for ’dID’ and ’dDocName’
used in the samples below to instead reflect what exists in
your content server.
<%
    String id = request.getParameter("dID");
    String docName = request.getParameter("dDocName");
if (id != null)
    {
%>
Obtain a specific revision:
<scs:docInfo id="<%=id%>" />
ID                : <scs:getValue name="dID" />
Title            : <scs:getValue name="dDocTitle"/>
Type             : <scs:getValue name="dDocType"/>
Author           : <scs:getValue name="dDocAuthor"/>
Is Checked Out   : <scs:getValue name="dIsCheckedOut"/>
Checked out by   : <scs:getValue name="dCheckoutUser"/>
<%
    }
    else if (docName != null)
    {
%>
Obtain the most recent revision:
<scs:docInfo docName="<%=docName%>" />
Name             : <scs:getValue name="dDocName" />
Title            : <scs:getValue name="dDocTitle"/>
Type             : <scs:getValue name="dDocType"/>
Author           : <scs:getValue name="dDocAuthor"/>
Is Checked Out   : <scs:getValue name="dIsCheckedOut"/>
Checked out by   : <scs:getValue name="dCheckoutUser"/>
<%  }   %>
Display the previous revisions. Note that the getValue tags
need to specify which result set to get the values from,
since this data is in multiple result sets and the loca
data:
</pre>
<table border="2">
    <tr>
        <td>Revision</td>
        <td>Release Date</td>
        <td>Status</td>
    </tr>
<scs:resultSet name="REVISION_HISTORY">
    <tr>
        <td><scs:getValue resultSet="REVISION_HISTORY"
name="dRevLabel"/></td>
        <td><scs:getValue resultSet="REVISION_HISTORY"
name="dInDate"/></td>
        <td><scs:getValue resultSet="REVISION_HISTORY"
name="dStatus"/></td>
    </tr>
</scs:resultSet>
</table>
</body>
</html>
```

Chapter

6

RUNNING THE STELLENT EJB ON A J2EE APP SERVER

INTRODUCTION

The information contained in this guide is based on Stellent™ Content Server 6.1 and ContentServerBean 6.1. The information is subject to change as the product technology evolves and as hardware and operating systems are created and modified.

This chapter contains these topics:

- ❖ Understanding the Architecture
- ❖ Deployment Prerequisites
- ❖ Basic Deployment Steps

UNDERSTANDING THE ARCHITECTURE

Stellent Content Server core functionality can be accessed from a JSP through the Stellent Content Server Enterprise JavaBean (EJB) deployed on your J2EE application server. The Stellent Content Server Enterprise JavaBean (EJB) is called *ContentServerBean*. The ContentServerBean EJB can be deployed in any J2EE compliant Enterprise JavaBean container such as WebLogic, WebSphere, or EAServer.

The ContentServerBean EJB provides the functionality to call Stellent services from within a J2EE Enterprise JavaBean environment and provides a gateway to all Stellent services. The ContentServerBean EJB is called from a J2EE environment application server, but executes the function completely in the Stellent environment.

The ContentServerBean EJB provides robust connectivity to Stellent Content Server using J2EE compliant EJB architecture. This enables integration of Stellent Content Server into extended multiple-tier applications.

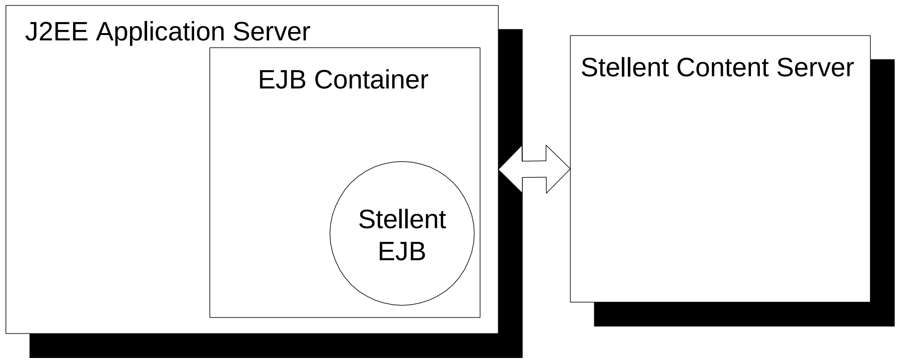


Figure 6-1 Access services From a JSP through the Stellent Content Server Enterprise JavaBean (EJB) deployed on your J2EE application server.

Note: Refer to the JavaDoc API in the */docs* sub-directory on the Stellent Content Server CD-ROM for information on the Class/Interface, Field, and Method descriptions of the ContentServerBean EJB.

DEPLOYMENT PREREQUISITES

System Requirements

The following table describes the system requirements for deploying the ContentServerBean.

Category	**Requirements**
Content Management System	Stellent Content Server 6.1
Java Environment	Required: • Java 2 SDK 1.3 • Java 2 SDK 1.3 and Java HotSpot Server VM, v.2.0 (for BEA WebLogic 5.1 installation)
Deployment Server	You can deploy the ContentServerBean on any J2EE compliant application server supporting EJB 1.1. This document provides deployment instructions for the following servers: • BEA WebLogic® Server v.5.1 with Service Pack v.6 • BEA WebLogic® Server v.6.0 • IBM Websphere® Server v.4.0 • Sybase EAServer® Enterprise Application Server v.3.6.1

Software and Documentation

The ContentServerBean software and documentation is included on the Stellent Content Server CD-ROM in the *extras/J2ee* folder.

Of particular interest:

❖ The ContentServerBean documentation is located in the *extras/J2ee/docs* sub-directory.

❖ The ContentServerBean JAR file (*IdcEjb.jar)* is located in the *extras/J2ee/ejb* sub-directory.

❖ The ContentServerBean Test Client is located in the *extras/J2ee/ejb* sub-directory.

Note: Refer to *Installing and Configuring Stellent Portlets on WebLogic Portal Server* for information on integrating Stellent Portlets with BEA WebLogic Portal Server to access Stellent Content Server.

Note: Refer to *Stellent EJB Integration for WebLogic Personalization Server* for information on integrating a deployed Stellent ContentServerBean EJB with BEA WebLogic Personalization Server and configuring a portal to access Stellent Content Server.

Note: Refer to *Installing and Configuring Stellent Portlets on WebSphere Portal Server* for information on integrating Stellent Portlets with IBM WebSphere Portal Server to access Stellent Content Server.

BASIC DEPLOYMENT STEPS

The deployment of the ContentServerBean EJB requires several basic steps that are common to all application servers. While these steps may vary depending on the specific application server, these basic steps are still performed.

Important: Before deploying the ContentServerBean ensure that the correct Java environment exists, and both the application server and Stellent Content Server are installed.

These basic steps are required for generation and deployment:

1. Copy the ContentServerBean to a location on your primary system.
2. Compile the Java classes and interfaces.
3. Generate the container classes. This creates the container-generated EJBObject implementing the Remote interface.
4. Run RMI against the Remote classes. This generates the Stub and Skeleton structure of the Remote Method Invocation architecture.
5. JAR the re-compiled ContentServerBean along with the associated files (such as the deployment descriptor).
6. Deploy the re-compiled ContentServerBean to the application server.

Generally, an application server provides a utility or wizard to automate steps 2-6 or steps 3-5. The application server utility or wizard may perform these steps with a series of dialog screens or automatically when invoked. For example, each basic step in generation and deployment may be performed in this manner:

- ❖ Step two is often performed by the utility or wizard using the SDK `javac` Java compiler or a similar complier.
- ❖ Step three is performed by the utility or wizard by referencing the SDK `javax` extension classes to extend `EJBObject` for the Remote session interface and to extend `EJBHome` for the Home session interface.

- Step four is performed by the utility or wizard referencing the SDK `rmic` Remote Method Invocation compiler.
- Step five is often performed by the utility or wizard using the Sun `jar.exe` utility or a similar JAR utility.
- Step six is often performed by the utility or wizard but may involve manually updating several application server specific files. For example, updating a file to reference the re-compiled ContentServerBean or to ensure that the JNDI name is properly defined.

Note: The `javax` extension classes, the `javac` compiler, the `rmic` compiler, and the `jar.exe` utility are all provided with Java 2 SDK 1.3.

Chapter

7

EJB DEPLOYMENT ON BEA WEBLOGIC 6.0

INTRODUCTION

This chapter provides information required for the deployment of the ContentServerBean using BEA WebLogic® Server as the J2EE compliant Enterprise JavaBean container.

This chapter contains these topics:

- ❖ Installation Prerequisites
- ❖ ContentServerBean Deployment
- ❖ Administration Notes

INSTALLATION PREREQUISITES

These components must be installed prior to the installation of the Stellent ContentServerBean EJB:

❖ Stellent Content Server 6.1.
Refer to the *Stellent Content Server Installation Guide*.

❖ WebLogic Server 6.0 with Service Pack 2

Note: Refer to *Installing and Configuring Stellent Portlets on WebLogic Portal Server* for information on integrating Stellent Portlets with BEA WebLogic Portal Server to access Stellent Content Server.

Note: Refer to *Stellent EJB Integration for WebLogic Personalization Server* for information on integrating a deployed Stellent ContentServerBean EJB with BEA WebLogic Personalization Server and configuring a portal to access Stellent Content Server.

CONTENTSERVERBEAN DEPLOYMENT

These are the required installation steps for the ContentServerBean deployment on BEA WebLogic Server:

❖ Preliminary Setup

❖ Prepare the ContentServerBean for Deployment

❖ Start the WebLogic Server

Preliminary Setup

You must add the ContentServerBean related folders to your system.

Complete the following steps:

1. Insert the Stellent CD-ROM.
2. Copy the *J2ee* folder to a location on your primary system. This folder is located on the CD-ROM within the *extras* directory.

   For example:

   ```
   C:/stellent/J2ee/
   ```

Note: The ContentServerBean JAR file (*IdcEjb.jar)* and the Test Client are located in the *J2ee/ejb* sub-directory.

Prepare the ContentServerBean for Deployment

You must copy the ContentServerBean JAR file to the WebLogic Admin Domain Name *applications* sub-directory for automatic deployment upon server startup.

Complete the following steps:

1. Locate the ContentServerBean JAR file (*IdcEjb.jar)* within the *J2ee/ejb* sub-directory.

   For example:

   ```
   C:/stellent/J2ee/ejb/IdcEjb.jar
   ```

Note: It is assumed that the J*2ee* folder was copied from the Stellent CD-ROM to a location on your primary system.

2. Copy the *IdcEjb.jar* file into the *applications* sub-directory within the WebLogic Admin Domain Name directory.

   For example (if defaults were accepted on installation):

   ```
   C:/bea/wlserver6.0/config/mydomain/applications/IdcEjb.j
   ar
   ```

Note: The *applications* sub-directory acts as an automatic deployment directory for EJB JAR files. WebLogic Server automatically deploys any valid EJB JAR file that resides in the *applications* sub-directory upon startup of the server.

Start the WebLogic Server

Run the WebLogic Server from the command prompt (or from the *Start* menu on Windows). The message *WebLogic Server Started* is displayed when the server is started.

Startweblogic file location (if defaults were accepted on installation):

```
C:/bea/wlserver6.0/config/mydomain/startweblogic.cmd
```

For Windows:

```
startServer.cmd
```

For UNIX:

```
startServer.sh
```

WebLogic Server automatically loads the EJB jar file, runs the classes and interfaces through the Java compiler, generates WebLogic Server container classes for the EJB, runs each EJB container class through the RMI compiler to create a client-side stub and a server-side skeleton, and finally, deploys the EJB to the container.

The deployment of the ContentServerBean is complete.

Note: For information on testing the EJB, see the appendix *Testing the ContentServerBean EJB*.

View Log File

The WebLogic Server generates a log file named *weblogic.log* to the *config/mydomain/logs* directory. View this log file to verify that the ContentServerBean has successfully deployed.

Log file location (if defaults were accepted on installation):

```
C:/bea/wlserver6.0/config/mydomain/logs/weblogic.log
```

Log file example:

```
<EJB Deploying file: IdcEjb.jar>
<EJB Deployed EJB with JNDI name IdcEjb.>
<Deployed : IdcEjb>.
```

View WebLogic Server Console

Follow these steps to view the active ContentServerBean from the Weblogic Server Console.

1. Launch the Weblogic Server Administration Console.
2. In the right pane, locate the J2EE heading, and click the EJB link.

   A list of the EJB deployments for the server displays in the right-hand pane.
3. Click the IdcEjb link under the Application entry.

   The directory path and deployment status for IdcEjb displays in the right-hand pane.

ADMINISTRATION NOTES

If you have a standard installation of Stellent, reference the Stellent home page at:

```
http://<stellent_dir>/stellent
```

Note: For additional information, refer to the *Stellent Content Server System Administration Guide*.

Note: For information on using a content server instance located remotely at Stellent corporate headquarters see the appendix *Using the Remote Stellent Content Server*.

Chapter

8

EJB DEPLOYMENT ON BEA WEBLOGIC 5.1

INTRODUCTION

This chapter provides information required for the deployment of the ContentServerBean using BEA WebLogic® Server as the J2EE compliant Enterprise JavaBean container.

This chapter contains these topics:

❖ Installation Prerequisites

❖ ContentServerBean Deployment

❖ Administration Notes

INSTALLATION PREREQUISITES

These components must be installed prior to the installation of the ContentServerBean:

❖ Stellent Content Server 6.1.
Refer to the *Stellent Content Server Installation Guide*.

❖ Java 2 SDK 1.3 (Do not use SDK 1.3.1 or greater)

❖ Java HotSpot Server VM 2.0

❖ WebLogic Server 5.1

❖ WebLogic Service Pack 6

Important: WebLogic Server 5.1 requires the installation of Java HotSpot Server VM 2.0 with Java 2 SDK 1.3. HotSpot Server 2.0 does not install over SDK 1.3.1.

Important: If WebLogic Service Pack 6 is not integrated with your version of WebLogic Server, you must acquire the ZIP file for the Service Pack from BEA and the Service Pack must be properly installed. This includes updating the `classpath` and the `weblogic.class.path` properties of the *wlconfig* file. Reference the BEA WebLogic Server 5.1 documentation for complete instructions.

For example:

Set the classpath (one line)

```
C:\weblogic\bin>wlconfig -classpath
C:\weblogic\weblogic510sp6boot.jar;C:\weblogic\myserver\
serverclasses
```

Set the weblogic classpath (one line)

```
C:\weblogic\bin>wlconfig -
Dweblogic.class.path=C:\weblogic\weblogic510sp6boot.jar;
C:\weblogic\license;C:\weblogic\classes;C:\weblogic\myse
rver\serverclasses;C:\weblogic\lib\weblogicaux.jar
```

CONTENTSERVERBEAN DEPLOYMENT

These are the required installation steps for the ContentServerBean deployment on BEA WebLogic Server:

- ❖ Preliminary Setup
- ❖ Prepare the ContentServerBean for Deployment
- ❖ Re-Compile the ContentServerBean
- ❖ Edit the EJB Properties File
- ❖ Edit the Startweblogic File
- ❖ Start the WebLogic Server

Preliminary Setup

You must add the ContentServerBean related folders to your system.

Complete the following steps:

1. Insert the Stellent CD-ROM.
2. Copy the *J2ee* folder to a location on your primary system. This folder is located on the CD-ROM within the *extras* directory.

   For example:

   ```
   C:/stellent/J2ee/
   ```

Note: The ContentServerBean JAR file (*IdcEjb.jar)* and the Test Client are located in the *J2ee/ejb* sub-directory.

Prepare the ContentServerBean for Deployment

You must copy the ContentServerBean JAR file to the *weblogic/classes* directory and create a new WebLogic sub-directory.

Complete the following steps:

1. Locate the ContentServerBean JAR file (*IdcEjb.jar)* within the *J2ee/ejb* sub-directory.

   For example:

   ```
   C:/stellent/J2ee/ejb/IdcEjb.jar
   ```

Note: It is assumed that the J*2ee* folder was copied from the Stellent CD-ROM to a location on your primary system.

2. Copy the *IdcEjb.jar* file to the *weblogic/classes* directory.

   For example:

   ```
   C:/weblogic/classes/IdcEjb.jar
   ```

3. Within the WebLogic *classes* directory, create a *lib* sub-directory.

   For example:

   ```
   C:/weblogic/classes/lib/
   ```

Re-Compile the ContentServerBean

Important: Do not use the BEA WebLogic Server "EJB Deployer Tool" to deploy the ContentServerBean. Follow the instructions provided in this document.

Use the WebLogic *ejbc* utility to re-compile the *IdcEjb.jar* file into a JAR file for use by the WebLogic server. This utility is named `ejbc.exe` and is located in the *weblogic/bin* directory.

To re-compile the ContentServerBean perform these steps:

1. From a command prompt, navigate to the *weblogic/classes* directory.
2. Run the WebLogic *ejbc* utility by specifying these entries on the command prompt:

   ❖ Define the complete path to the *weblogic.ejbc* utility.

   ❖ Set the compiler option and define the complete path to the *javac* compiler (the Java compiler is provided with SDK 1.3).

   ❖ Define the relative path to the *IdcEjb.jar* file. This JAR file was placed in the *weblogic/classes* directory as part of the Preliminary Setup.

   ❖ Define the relative path to the new *lib* folder and rename the *IdcEjb.jar* file to *wl_IdcEjb.jar* (the *ejbc* utility requires that the output JAR name be different from the input JAR name).

   For example:

   ```
   C:/weblogic/bin/ejbc.exe -compiler C:/jdk1.3/bin/javac
   IdcEjb.jar ./lib/wl_IdcEjb.jar
   ```

   The *weblogic.ejbc* utility performs these actions:

   - Checks all EJB classes and interfaces for compliance with the EJB specification.
   - Generates WebLogic Server container classes for the EJB.

- Runs each EJB container class through the RMI compiler to create a client-side stub and a server-side skeleton.
- Places the XML deployment descriptor and the EJB classes and interfaces in the new JAR file.

Edit the EJB Properties File

Follow these steps to edit the WebLogic EJB Properties file to deploy the re-compiled ContentServerBean:

1. In the *weblogic* directory, open the *weblogic.properties* file in a text-only editor.
2. Locate the WEBLOGIC EJB PROPERTIES section of the *weblogic.properties* file.

Caution: When editing this file, append entries to the existing list. Do not delete or overwrite other EJB entries.

3. Edit the *weblogic.properties* file to deploy the re-compiled ContentServerBean by setting the *weblogic.ejb.deploy* entry to define the complete path to the *wl_IdcEjb.jar* file.

   For example:

   ```
   weblogic.ejb.deploy=\
   C:/weblogic/classes/lib/wl_IdcEjb.jar
   ```

Important: Ensure that the *weblogic.ejb.deploy* entry and the *wl_IdcEjb.jar* directory path entry are not prefixed with a comment tag (#).

4. Save the *weblogic.properties* file.

Edit the Startweblogic File

Follow these steps to edit the *startWebLogic* file to append the re-compiled ContentServerBean JAR file to the end of the WEBLOGIC_CLASSPATH:

1. In the WebLogic Server directory, open the *startWebLogic* file in a text-only editor.
   - For Windows, edit the *startWebLogic.cmd* file.
   - For UNIX, edit the *startWebLogic.sh* file.
2. Edit the POST_CLASSPATH entry to include the complete directory to the *wl_IdcEjb.jar* file.

   For example:

   ```
   set POST_CLASSPATH=C:/weblogic/classes/lib/wl_IdcEjb.jar
   ```
3. Save the *startweblogic.cmd* file.

Start the WebLogic Server

Run the WebLogic Server from the command prompt. The message *WebLogic Server Started* is displayed when the server is started.

For Windows:

```
/Weblogic/startserver.cmd
```

For UNIX:

```
/Weblogic/startServer.sh
```

Tech Tip: On Windows, if the server starts from the command prompt and deploys the ContentServerBean but generates an error when run from the *START* menu, the properties in the *wlconfig* file need to be reconciled to match the properties in the *startWebLogic* file.

The WebLogic Server generates a log file named *weblogic.log* to the *weblogic/myserver* directory. View this log file to verify that the ContentServerBean has successfully deployed.

For example:

```
<EJB> Enterprise JavaBeans initializing

<EJB JAR deployment C:/weblogic/classes/lib/wl_IdcEjb.jar>
EJB home interface: 'idcbean.ContentServerHome' deployed
bound to the JNDI name: 'IdcEjb'

<EJB> 1 EJB jar files loaded, containing 1 EJBs

<EJB> 1 deployed, 0 failed to deploy.
```

The deployment of the ContentServerBean is complete.

Note: For information on testing the EJB, see the appendix *Testing the ContentServerBean EJB*.

ADMINISTRATION NOTES

If you have a standard installation of Stellent, reference the Stellent home page at:

```
http://<stellent_dir>/stellent
```

Note: For additional information, refer to the *Stellent Content Server System Administration Guide*.

Chapter

9

EJB DEPLOYMENT ON IBM WEBSPHERE

INTRODUCTION

This chapter provides information required for the deployment of the ContentServerBean using IBM Websphere® Server as the J2EE compliant Enterprise JavaBean container.

This chapter contains these topics:

❖ Installation Prerequisites

❖ ContentServerBean Deployment

❖ Administration Notes

INSTALLATION PREREQUISITES

These components must be installed prior to the installation of the ContentServerBean:

❖ Stellent Content Server 6.1.
Refer to the *Stellent Content Server Installation Guide*.

❖ Java 2 SDK 1.3.1

❖ Websphere Server 4.0

Note: Refer to *Installing and Configuring Stellent Portlets on WebSphere Portal Server* for information on integrating Stellent Portlets with IBM WebSphere Portal Server to access Stellent Content Server.

CONTENTSERVERBEAN DEPLOYMENT

These are the required installation steps for the ContentServerBean deployment on IBM Websphere Server:

❖ Preliminary Setup

❖ Assemble the ContentServerBean

❖ Install the ContentServerBean

❖ Start the Websphere Application Server

Preliminary Setup

Add the ContentServerBean folders to your system.

Complete the following steps:

1. Insert the Stellent CD-ROM.

2. Copy the Stellent *j2ee* folder to a location on your primary system.

   For example:

   ```
   c:/stellent/j2ee/
   ```

Note: The ContentServerBean JAR file (*IdcEjb.jar*) and the ContentServerBean Test Client are located in the *ejb* sub-folder.

Assemble the ContentServerBean

1. Start the Websphere Application Assembly Tool from a command prompt by running:

   For Windows:

   ```
   <websphere_dir>/bin/assembly.bat
   ```

   For UNIX:

   ```
   <websphere_dir>/bin/assembly.sh
   ```

2. Using the Application Assembly Tool, create an EAR file by selecting **File— Wizards—Create Application Wizard**.
3. On the Specifying Application Properties screen, enter the following:

   Display Name: **ContentServerBean**

   File Name: **ContentServerBean.ear**

4. Click **Next**.

   The Adding Supplementary Files screen is displayed.

5. Click **Next**.

   The Choosing Application Icons screen is displayed.

6. Click **Next**.

   The Adding EJB Module screen is displayed.

7. Click **Add**.
8. Navigate to the *IdcEjb.jar* file, select the file and click **Open**.
9. Click **OK**.
10. Click **Next.**

    The Adding Application Client Modules screen is displayed.

11. Click **Next.**

    The Adding Security Roles screen is displayed.

12. Click **Finish**.
13. Bind the EJB and references to it:
    a. Within the Session Beans folder, select **IdcEjb**.
    b. Right click and select **Properties**.
    c. Select the **Bindings** tab.
    d. Replace the existing text string with `IdcEjb` as the global JNDI name for the session bean.
    e. Click **OK**.
14. Save the EAR file by selecting **File—Save As**.
15. Name the file **ContentServerBean.ear**.
16. Deploy the EAR file.
17. Right click the EAR file by selecting the top entry in the topology tree.
18. Select **Generate Code for Deployment**.
19. Select the appropriate **Database Type.**
20. Click **Generate Now**.
21. When deployment is complete, close this screen.
22. Exit the application assembly tool using the **File**—**Exit** option.

Install the ContentServerBean

The ContentServerBean can be deployed using the Websphere Administrative Console or from the command prompt.

Deploy Using the Websphere Administrative Console

1. Start the Websphere Application Server. From the Windows START menu select *Websphere Application Server* or run the server from a command prompt using the following commands:

   For Windows:

   ```
   <websphere_dir>/bin/startServer.bat
   ```

   For UNIX:

   ```
   <websphere_dir>/bin/startServer.sh
   ```

2. Start the Websphere Administrative Console by browsing to the location of the admin folder. For example:

   ```
   http://localhost:9090/admin
   ```

3. On the navigation pane, expand **Nodes—<localhost>**
4. Click **Enterprise Applications**.

   A list of applications is displayed.
5. Click **Install**.
6. Click **Browse.**
7. Navigate to your deployed EAR file (**Deployed_ContentServerBean.ear**), select the file and click **Open**.
8. Click **Next**.
9. On the Binding Enterprise Beans to JNDI Names screen, verify that the JNDI Name is set to `IdcEjb` and click **Next**.

10. On the *M*apping EJB References to Enterprise Beans screen, verify that the Enterprise Bean is set to `IdcEjb` and click **Next**.
11. Click **Next**.
12. On the EJB Deploy screen, deselect the check box for Re-Deploy option.

Note: Because we deployed the EAR file from the application assembly tool, there is no need to re-deploy the EJB jar at this time.

13. Click **Next**.
14. Click **Finish**.
15. On the navigation pane, expand **Nodes—<localhost>—Application Servers**.
16. Select **Default Server**
17. Change the Module Visibility to Application.
18. Click **OK**.
19. Save the configuration by clicking the **Save** button. Save it under server-cfg.xml and click **OK.**
20. Stop the server.

Deploy Using the Command Prompt

1. Stop the Websphere application server using the Websphere Administrative Console or by using the following commands:

   For Windows:

   ```
   <websphere_dir>/bin/stopServer.bat
   ```

   For UNIX:

   ```
   <websphere_dir>/bin/stopServer.sh
   ```

Note: By default, the script attempts to stop the application server configured in the default configuration file:

`<websphere_dir>/config/server-cfg.xml`
If you are running an application server using a different configuration file, you can specify that configuration file using the `stopServer` command and defining the `-configFile` parameter.

2. Install the ContentServerBean using the application installer tool by using the following commands:

   For Windows:

   ```
   SEAppInstall -install
   <_dir>/Deployed_ContentServerBean.ear -ejbdeploy false
   ```

Important: The `<_dir>` entry must be the fully qualified path to the EAR file.

- ❖ Running `SEAppInstall` with no arguments will return additional information about the arguments.
- ❖ The `-ejbdeploy false` switch disables the running of the *ejbdeploy* tool. The *ejbdeploy* tool was run using the Websphere Application Assembly Tool in an earlier step.

For UNIX:

```
cd <websphere_dir>/bin
```

Invoke `SEAppInstall.sh` using the syntax for your operating system and then use these arguments:

```
-install <_dir>/Deployed_ContentServerBean.ear -
ejbdeploy false
```

Important: The `<_dir>` entry must be the fully qualified path to the EAR file.

- ❖ Running `SEAppInstall` with no arguments will return additional information about the arguments.

- ❖ The `-ejbdeploy false` switch disables the running of the *ejbdeploy* tool. The *ejbdeploy* tool was run using the Websphere Application Assembly Tool in an earlier step.

3. At the prompt to precompile all JSP, respond **No.**
4. At the prompt to precompile individual Web Applications, respond **No.**
5. At the prompt for a default data source for the EJB jar, press **Enter**.
6. At the prompt for the JNDI names, press **Enter**.

   This will default to `IdcEjb.`

Important: If you encounter problems with the Java Naming and Directory Interface (JNDI) names, ensure that you configured the bindings correctly with the Websphere Application Assembly Tool.

7. At the prompt to specify a virtual host, specify **default_host.**

Start the Websphere Server

From the Windows START menu select *Websphere Application Server* or run the server from a command prompt using the following commands:

For Windows:

```
<websphere_dir>/bin/startServer.bat
```

For UNIX:

```
<websphere_dir>/bin/startServer.sh
```

The installation of the ContentServerBean is complete.

Note: For information on testing the EJB, see the appendix *Testing the ContentServerBean EJB.*

Note: For additional information, refer to *Installing and Configuring Stellent Portlets on WebSphere Portal Server.*

Administration Notes

If you have a standard installation of Stellent, reference the Stellent home page at:

```
http://<stellent_dir>/stellent
```

Note: For additional information, refer to the *Stellent Content Server System Administration Guide*.

Note: For information on using a content server instance located remotely at Stellent corporate headquarters see the appendix *Using the Remote Stellent Content Server*.

Chapter

10

EJB Deployment on Sybase EAServer

Introduction

This chapter provides information required for the deployment of the ContentServerBean using Sybase EAServer® Enterprise Application Server as the J2EE compliant Enterprise JavaBean container.

This chapter contains these topics:

❖ Installation Prerequisites

❖ ContentServerBean Deployment

❖ Administration Notes

INSTALLATION PREREQUISITES

These components must be installed prior to the installation of the ContentServerBean:

- ❖ Stellent Content Server 6.1.
  Refer to the *Stellent Content Server Installation Guide*.
- ❖ Java 2 SDK 1.3.1.
- ❖ Sybase EAServer® Enterprise Application Server 3.6.1 .

CONTENTSERVERBEAN DEPLOYMENT

These are the required installation steps for the ContentServerBean deployment on Sybase EAServer Enterprise Application Server

- ❖ Preliminary Setup
- ❖ Deploy the JAR File
- ❖ Install the ContentServerBean on Jaguar Server
- ❖ Start the EAServer Enterprise Application Server

Preliminary Setup

Add the ContentServerBean folders to your system. Complete the following steps:

1. Insert the Stellent CD-ROM.
2. Copy the Stellent *j2ee* folder to a location on your primary system.

   For example:

   ```
   c:/stellent/j2ee/
   ```

Note: The ContentServerBean JAR file (*IdcEjb.jar*) and the ContentServerBean Test Client are located in the *ejb* sub-folder.

Deploy the JAR file

The ContentServerBean can be deployed using the EAServer Deploy Wizard. Follow these steps to deploy the ContentServerBean to your Jaguar repository, then install it into your Jaguar Server.

Note: The Jaguar Component Transaction Server (Jaguar Server), the Jaguar Manager, and the Deploy Wizard are integrated tools provided with EAServer.

1. Start the Jaguar Server using JDK 1.2 from a command prompt by running:

   For Windows:

   ```
   serverstart_jdk12.bat
   ```

   For UNIX:

   ```
   srvstart_jdk12
   ```

2. Run Jaguar Manager and connect to the Jaguar Server
3. Expand the Jaguar Manager, right-click the Packages folder and choose **Deploy—EJB 1.1 Jar**.

   The Deploy Wizard screen is displayed.

4. Using the Deploy Wizard, define the location of the *IdcEjb.jar* file, or click **Browse** and navigate to the file.
5. Click **Next**.

   The Deployment Status screen is displayed.

6. After the *IdcEjb.jar* file is successfully deployed, click **Close**.

   The ContentServerBean is now deployed to the repository, but still needs to be installed to the Jaguar server.

Install the ContentServerBean on Jaguar Server

1. Right-click on the Installed Packages folder of your Jaguar server, and choose Install Package.

   The Package Wizard screen is displayed.

2. Click “Install an existing package” and choose **IdcEjb**.

   The Install Package screen is displayed.

3. Select **IdcEjb**, and click **OK**.

   The EJB is now installed on your Jaguar Server.

Start the EAServer Enterprise Application Server

Stop and start the EAServer Enterprise Application Server. The installation of the ContentServerBean is complete.

Note: For information on testing the EJB, see the appendix *Testing the ContentServerBean EJB*.

ADMINISTRATION NOTES

If you have a standard installation of Stellent, reference the Stellent home page at:

```
http://<stellent_dir>/stellent
```

Note: For additional information, refer to the *Stellent Content Server System Administration Guide*.

Appendix

A

TESTING THE CONTENTSERVERBEAN EJB

INTRODUCTION

This chapter provides information on testing the the ContentServerBean EJB and defining the application server test client variables.

This appendix contains these topics:

❖ Testing the ContentServerBean EJB

❖ Application Server Test Client Settings

TESTING THE CONTENTSERVERBEAN EJB

This section provides the steps for testing the ContentServerBean EJB. It is assumed that the ContentServerBean is deployed in a J2EE compliant Enterprise JavaBean container such as the BEA WebLogic Server.

The ContentServerBean can be tested using the *EjbTestClient.java* file. This is a simple test client for the ContentServerBean EJB. It must be altered to function with your J2EE compliant EJB server, and recompiled.

The default entries provided in the file are currently set up to connect to a BEA WebLogic EJB server and a Stellent Content Server on the same computer as the test client.

Note: See *Application Server Test Client Setting* (page A-3) for information on the application server variables that define the class for talking to the server, the application server URL, and the Java Naming and Directory Interface (JNDI) name.

Follow these steps the customize the entries in the *EjbTestClient.java* file.

1. Open the *EjbTestClient.java* file in a text-only editor.
2. Alter theses variabes for *public class EjbTestClient* to function with your EJB deployment and your content server (see figure A-1):
   - lookup (JNDI lookup name)
   - url (application server URL address)
   - contextFactory
   - appServerUser
   - appServerPassword
   - idcUse
   - idcConnectionString

3. Compile and run the EjbTestClient.java file. For example, run the SDK *javac* compiler from the command prompt (e.g., *C:\jdk1.3\bin\javac*).

The ContentServerBean Test Client runs tests against everything in the EjbServices class

```
protected static String lookup = "IdcEjb";
protected static String url = "t3://localhost:7001";
protected static String contextFactory =
"weblogic.jndi.WLInitialContextFactory";
protected static String appServerUser = null;
protected static String appServerPassword = null;
protected static String idcUser = "sysadmin";
protected static String idcConnectionString =
"socket:localhost:4444";
```

Figure A-1 EjbTestClient.java file

APPLICATION SERVER TEST CLIENT SETTINGS

The application server test client settings define the class for talking to the server, the application server URL, and the Java Naming and Directory Interface (JNDI) name.

BEA WebLogic

```
contextFactory=weblogic.jndi.WLInitialContextFactory
url=t3://ogre:7001
lookup=IdcEjb
```

IBM Websphere

```
contextFactory=
com.ibm.websphere.naming.WsnInitialContextFactory
```

```
url=iiop://localhost:900
```

```
lookup=IdcEjb
```

Sybase EAServer:

```
contextFactory=com.sybase.ejb.InitialContextFactory
```

```
url=iiop://localhost:9000
```

```
lookup=IdcEjb/IdcEjb
```

Appendix

B

USING THE REMOTE STELLENT CONTENT SERVER

INTRODUCTION

For testing and demonstration purposes, Stellent provides a remote Stellent Content Server located at the Stellent headquarters. This content server instance is *testportal.stellent.com* with a port of 4444.

This chapter contains these topics:

❖ Referencing the Remote Content Server

❖ Referencing on WebLogic Personalization Server

❖ Referencing on WebSphere Portal Server

Referencing the Remote Content Server

To use the Stellent Content Server located remotely at Stellent corporate headquarters, edit the deployment descriptor for the custom enterprise application to reference the remote Stellent Content Server instance by defining the remote Stellent content server hostname and port number.

For example:

```
socket:testportal.stellent.com:4444
```

Referencing on WebLogic Personalization Server

To use the Stellent Content Server located remotely at Stellent corporate headquarters with WebLogic Personalization Server, the *idc_beasys.properties* file for the BEA Service Provider Interface (SPI) must be edited.

Follow these steps to edit the *portlet.xml* file:

1. Open the *idc_beasys.properties* file in a text-only editor.
2. Edit the *xcs.url* entry to specify the location of the remote content server. Replace the default entries for the *stellent hostname* and *stellent port number* (see Figure B-1) with this entry:

   ```
   socket:testportal.stellent.com:4444
   ```
3. Edit the *xcs.host* entry to specify the location of the remote content server. Replace the default entry for the *stellent host* (see Figure B-1) with this entry:

   ```
   testportal.stellent.com
   ```

```
xcs.jndiname=IdcEjb
xcs.url=socket:testportal.stellent.com:4444
xcs.user=sysadmin
xcs.dateformat=MM/dd/yy hh:mm a
xcs.host=testportal.stellent.com
```

Figure B-1 idc_beasys.properties:

Note: For additional information, refer to the *Stellent WebLogic Personalization Server Integration Guide*.

REFERENCING ON WEBSPHERE PORTAL SERVER

To use the Stellent Content Server located remotely at Stellent corporate headquarters with WebSphere Portal Server, the *portlet.xml* file for the portlet must be edited. The *portlet.xml* file contains the deployment descriptors for the portlet and is located in the PORTLET-INF sub-folder.

After installing the component for the desired portal, follow these steps to edit the *portlet.xml* file:

1. Open the *portlet.xml* file in a text-only editor.
2. Edit the *idcConnectionString* entries to specify the location of the remote content server. Replace the comment lines for *stellent hostname* and *stellent port number* (see Figure B-2) with these entries:

   ```
   testportal.stellent.com
   4444
   ```

```
<config-param>
<param-name>idcConnectionString</param-name>
<param-value>socket:<!-- Replace this comment with your
stellent hostname -->:<!-- Replace this comment with your
stellent port number --></param-value>
</config-param>
```

Figure B-2 Portlet.xml—idcConnectionString (Stellent Hostname and Port Number):

Important: Do not delete the colon separating the comment lines, as this is a required entry to define the path.

3. Edit the *stellentHost* entry to specify the remote content server. Replace the comment line for *stellent hostname* (see Figure B-3) with this entry:

   `testportal.stellent.com`

```
<config-param>
<param-name>stellentHost</param-name>
<param-value><!-- Replace this comment with your stellent
hostname --></param-value>
</config-param>
```

Figure B-3 Portlet.xml—stellentHost:

4. If more than one portlet is being installed, edit the *portlet.xml* file located in each portlet folder.

Note: For additional information, refer to *Installing and Configuring Stellent Portlets on WebSphere Portal Server.*

Index