Liferay Portal 4.0 - Developers Guide

Joseph Shum

Alexander Chow

Liferay Portal 4.0 - Developers Guide

Joseph Shum
Alexander Chow

Table of Contents

Preface

This document is intended as a reference guide for developers of Liferay Portal 4.0. It is still a work in progress and will be updated frequently with new content. Contributions are welcome. Please email `<support@liferay.com>` to provide documents that you have written and would like to contribute back to the community. Topic suggestions are also welcome.

Need Help? Forums: http://forums.liferay.com

Tracker: http://support.liferay.com

Mailing List: Subscribe [http://lists.sourceforge.net/lists/listinfo/lportal-development] or Browse [http://sourceforge.net/mailarchive/forum.php?forum=lportal-development] Please search through the mailing list first if you have an issue.

The mailing list contains a lot of information because we've been using it for many years. However, its threading is limited and does not allow you to continue a conversation that started a few months ago. We've switched to user forums [http://forums.liferay.com] so the community can have a better place to discuss their experiences with Liferay.

If you find a bug or have a new feature request, please post it on our tracker [http://support.liferay.com].

We also offer professional support services where your company can be assigned a Liferay developer that will ensure that your questions are answered promptly so that your project is never compromised. Purchased support always gets first priority. This business model allows us to build a company that can contribute a great portal to the open source community.

If your company uses Liferay, please consider purchasing support. Liferay has an extremely liberal license model (MIT, very similar to Apache and BSD), which means you can rebundle Liferay, rename it, and sell it under your name. We believe free means you can do whatever you want with it. Our only source of revenue is from professional support and consulting.

Chapter 1. Getting Started

Setting up the Development Environment

The following instructions will help you get your development environment ready for working with the source code. These instructions are specific to setting up for deployment to Orion server and Tomcat 5.5 developing with Java JDK 1.5. Liferay is compatible with Java 1.4 also and a wide array of application servers and containers. You will need to adjust your development environment according to your platform.

Before we can get started, the following conponents must be installed on your machine.

SVN

1. Download and install SmartSVN [http://www.smartsvn.com/].
2. If you wish to browse the source tree, configure SmartSVN to use the **https** protocol to connect to **svn.sourceforge.net**. Authenticate as user **anonymous** with a blank password and specify **/svn-root/lportal/portal** as the UNIX path.

   If you are a developer with privileges to commit to the source tree, configure SmartSVN to use the https protocol to connect to **svn.sourceforge.net.** Authenticate with your private user and password and specify **/svnroot/lportal/portal** as the UNIX path.
3. Check out the portal.

JDK 1.5.0

1. Download and install JDK 1.5.0 [http://java.sun.com/j2se/1.5.0/download.jsp].
2. Set an environment variable called %JAVA_HOME% to point to your JDK directory.

Jikes 1.22

1. Download and unzip Jikes 1.22 [http://www-124.ibm.com/developerworks/oss/jikes].
2. Set an environment variable called %JIKES_HOME% to point to your Jikes directory.
3. Add *`%JIKES_HOME%\bin`* to your %PATH% environment variable.

Ant 1.6.5

1. Download and unzip the latest version of Ant [http://ant.apache.org/].
2. Set an environment variable called %ANT_HOME% to point to your Ant directory.
3. Add %ANT_HOME%\bin to your %PATH% environment variable.

4. Learn how tasks work in Ant. Tasks can be run at the root of the project directory and inside each subproject directory.

5. Run **ant start** at least once to compile the source files and to generate all database scripts and skin images.

Orion 2.0.6

1. Download and unzip Orion 2.0.6 [???].

2. Create a separate properties file named app.server.${user.name}.properties modeled after app.server.properties, which is in the checked out SVN directory. The variable ${user.name} is the user name you used to log into your operating system. Overload the properties with the proper directory for Orion. This properties file is later referenced by Ant so it knows where to deploy the application.

3. Remove the config directory from Orion. Download config.zip [http://content.liferay.com/3.6.1/config.zip] and unzip the new config directory into Orion.

4. Edit /config/datasources.xml to make sure the proper database pool is created.

5. Populate your database with the portal schema and default data.

6. Edit /config/server.xml to make sure the server can find the proper location of Jikes.

7. Create a file named D:\svnroot\liferay\portal\util-java\classes\portal-ext.properties. Add the property portal.ctx with the value /portal.

8. Go to D:\svnroot\liferay\portal. After running **ant start** at least once, run ant **deploy** to copy all the necessary files to Orion.

9. If you are developing in a Windows environment, download run.bat [http://content.liferay.com/3.6.1/run.bat] to the Orion directory and execute the batch file after everything is properly deployed with Ant. If you are developing under a UNIX environment, download run.sh [http://content.liferay.com/3.6.1/run.sh] to the Orion directory and execute the script after everything is properly deployed with Ant.

Chapter 2. Installation and Setup

Application Servers

Liferay Portal supports many different application servers and servlet containers. Liferay Portal Enterprise requires a fully compliant J2EE application server whereas Liferay Portal Professional can run in a simple servlet container.

The source code is identical in both versions. The only difference is that a property setting configures Liferay Portal Enterprise to make all calls through EJBs whereas Liferay Portal Professional makes all calls through POJOs. We offer you choices so that those who need the benefit of EJBs can have it, and those who don't want the overhead of EJBs don't have to pay for it.

We have tried to make the installation process as easy as possible by including a bundled version of Liferay with our supported open source containers. For example, if you want to try out Liferay on Jetty, simply download our bundle with precompiled JSPs, unzip it, and run it. It already comes with a built in Java database so you have to do the least amount of configuration. We also detail the steps taken to configure these containers for use with Liferay. Some instructions are separated as **Easy** vs. **Expert**.

We also plan on adding support for Geronimo, GlassFish, and JFox in as soon as possible. Please contact us if you would like to help us in this effort.

Borland ES 6.5

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download and install Borland ES [http://www.borland.com/].
- Download liferay-portal-ent-4.0.0.ear [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0.ear].
- Populate your database with the portal schema and default data.
- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by Borland ES.
- Create a mail session bound to *mail/MailSession*. You only need to set the locations of the IMAP, POP3, and SMTP servers.
- Follow the instructions [http://info.borland.com/devsupport/bes/faq/6.5/portal/liferay.html] from Borland.
- Deploy `liferay-portal-ent-4.0.0.ear`.
- Open your browser to http://localhost:8080. Click on `Sign In` at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

Geronimo

We plan on adding support for Geronimo as soon as possible. Please contact us if you would like to help us in this effort.

GlassFish

We plan on adding support for GlassFish as soon as possible. Please contact us if you would like to help us in this effort.

JBoss+Jetty 4.0.2

Easy

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download liferay-portal-ent-4.0.0-jboss-jetty.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0-jboss-jetty.zip].
- Unzip liferay-portal-ent-4.0.0-jboss-jetty.zip to C:\LIFERAY (or any another directory). Be sure to unzip with folder names.
- Execute C:\LIFERAY\bin\run.bat to run the database and web server. Make sure there isn't another application already using port 8080.

  When in a Unix environment, the batch file to start the server will end with the extension sh instead of bat. Make sure to **chmod** the batch file so you can execute it. You must start the executable from the directory where it resides.
- Open your browser to http://localhost:8080 [http://localhost:8080/]. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

Expert

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download and install JBoss+Jetty [http://www.jboss.org/].
- Download liferay-portal-ent-4.0.0.ear [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0.ear].
- Populate your database with the portal schema and default data.
- Add to /server/default/conf/jboss-service.xml:

  ```
  <classpath codebase="lib/ext" archives="*"/>
  ```
- Download jetty-5.1.4-jboss-4.0.2.sar [http://prdownloads.sourceforge.net/jetty/jetty-5.1.4-jboss-4.0.2.sar?download] and unjar it to /server/default/deploy/jbossweb-jetty.sar.

  Remove /server/default/deploy/jbossweb-tomcat55.sar.

  Remove /server/default/deploy/jboss-ws4ee.sar.

  Remove /server/default/deploy/management.

- Use the prefix deployment sorter instead of the default deployment sorter.

  Edit /server/default/conf/jboss-service.xml.

  ```
  <attribute
  name="URLComparator">org.jboss.deployment.scanner.PrefixDeploymentSorter</attribute>
  ```

- Find the `org.jboss.web.WebService` MBean that listens on port 8083 and do not allow it send configuration files and other resources. Leaving it at the default value of true allows anyone to download your JBoss configuration files. This is a known vulnerability that will allow hackers to retrieve your database settings, etc.

  Edit /server/default/conf/jboss-service.xml.

  ```
  <attribute name="DownloadServerClasses">false</attribute>
  ```

- Configure data sources for your `database`. Make sure the JDBC driver for your database is accessible by JBoss+Jetty.

- Create a mail session bound to *mail/MailSession*. You only need to set the locations of the IMAP, POP3, and SMTP servers.

  Set the proper mail properties by editing /server/default/deploy/mail-service.xml.

  ```
  <mbean code="org.jboss.mail.MailService" name="jboss:service=MailSession">
      <attribute name="JNDIName">mail/MailSession</attribute>
      <attribute name="User">nobody</attribute>
      <attribute name="Password">password</attribute>
      <attribute name="Configuration">
          <configuration>
              <property name="mail.store.protocol" value="imap" />
              <property name="mail.transport.protocol" value="smtp" />
              <property name="mail.imap.host" value="localhost" />
              <property name="mail.pop3.host" value="localhost" />
              <property name="mail.smtp.host" value="localhost" />
          </configuration>
      </attribute>
  </mbean>
  ```

- Configure JAAS.

  Edit /server/default/conf/login-config.xml and comment out the entire XML for policy *other*.

  ```
  <!--<application-policy name = "other">
      ...
      <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
  flag = "required" />
      ...
  </application-policy>-->
  ```

- Deploy `liferay-portal-ent-4.0.0.ear`.

  Copy liferay-portal-ent-4.0.0.ear to /server/default/deploy.

- Start JBoss+Jetty.

  If you get a `java.lang.OutOfMemoryError` exception while starting up JBoss+Jetty, give your JVM more memory by setting *-Xmx512m*.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

JBoss+Tomcat 4.0.2

Easy

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download liferay-portal-ent-4.0.0-jboss-tomcat.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0-jboss-tomcat.zip].
- Unzip liferay-portal-ent-4.0.0-jboss-tomcat.zip to C:\LIFERAY (or any another directory). Be sure to unzip with folder names.
- Execute C:\LIFERAY\bin\run.bat to run the database and web server. Make sure that there isn't another application using port 8080.

  When in a Unix environment, the batch file to start the server will end with the extension sh instead of bat. Make sure to **chmod** the batch file so you can execute it. You must start the executable from the directory where it resides.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

Expert

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download and install JBoss+Tomcat [http://www.jboss.org/].
- Download liferay-portal-ent-4.0.0.ear [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0.ear].
- Populate your database with the portal schema and default data.
- Add to /server/default/conf/jboss-service.xml:

  ```
  <classpath codebase="lib/ext" archives="*"/>
  ```

- Remove /server/default/deploy/jbossweb-tomcat55.sar/ROOT.war.

  Edit /server/default/deploy/jbossweb-tomcat55.sar/conf/web.xml.

```
<init-param>
    <param-name>listings</param-name>
    <param-value>false</param-value>
</init-param>
<init-param>
    <param-name>input</param-name>
    <param-value>4096</param-value>
</init-param>
<init-param>
    <param-name>output</param-name>
    <param-value>4096</param-value>
</init-param>
```

Edit /server/default/deploy/jbossweb-tomcat55.sar/META-INF/jboss-service.xml.

```
<attribute name="Java2ClassLoadingCompliance">true</attribute>
<attribute name="UseJBossWebLoader">true</attribute>
```

- Use the prefix deployment sorter instead of the default deployment sorter.

  Edit /server/default/conf/jboss-service.xml.

```
<attribute
name="URLComparator">org.jboss.deployment.scanner.PrefixDeploymentSorter</attribute>
```

- Find the org.jboss.web.WebService MBean that listens on port 8083 and do not allow it send configuration files and other resources. Leaving it at the default value of true allows anyone to download your JBoss configuration files. This is a known vulnerability that would allow hackers to retrieve your database settings, etc.

  Edit /server/default/conf/jboss-service.xml.

```
<attribute name="DownloadServerClasses">false</attribute>
```

- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by JBoss+Tomcat.

- Create a mail session bound to mail/MailSession. You only need to set the locations of the IMAP, POP3, and SMTP servers.

  Set the proper mail properties by editing /server/default/deploy/mail-service.xml.

```
<mbean code="org.jboss.mail.MailService" name="jboss:service=MailSession">
    <attribute name="JNDIName">mail/MailSession</attribute>
    <attribute name="User">nobody</attribute>
    <attribute name="Password">password</attribute>
    <attribute name="Configuration">
        <configuration>
            <property name="mail.store.protocol" value="imap" />
            <property name="mail.transport.protocol" value="smtp" />
            <property name="mail.imap.host" value="localhost" />
            <property name="mail.pop3.host" value="localhost" />
            <property name="mail.smtp.host" value="localhost" />
        </configuration>
    </attribute>
```

```
</mbean>
```

- Configure JAAS.

  Edit /server/default/conf/login-config.xml and comment out the entire XML for policy other.

```
<!--<application-policy name = "other">
    ...
    <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
flag = "required" />
    ...
</application-policy>-->
```

- Deploy liferay-portal-ent-4.0.0.ear.

  Copy liferay-portal-ent-4.0.0.ear to /server/default/deploy.

- Start JBoss+Tomcat.

  If you get a `java.lang.OutOfMemoryError` exception while starting up JBoss+Tomcat, give your JVM more memory by setting -Xmx512m.

- Open your browser to http://localhost:8080. Click on `My Liferay` at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

- You can set up JBoss+Tomcat behind Apache using mod_jk2 by following the instructions in this document [http://content.liferay.com/document/Liferay_on_JBoss+Tomcat+Apache.pdf].

Jetty 5.1.4

Easy

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.

- Download liferay-portal-pro-4.0.0-jetty.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-pro-4.0.0-jetty.zip].

- Unzip liferay-portal-pro-4.0.0-jetty.zip to C:\LIFERAY (or any another directory). Be sure to unzip with folder names.

- Execute C:\LIFERAY\bin\run.bat to run the database and web server. Make sure there isn't another application using port 8080.

  When in a Unix environment, the batch file to start the server will end with the extension sh instead of bat. Make sure to **chmod** the batch file so you can execute it. You must start the executable from the directory where it resides.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

Expert

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download and install Jetty [http://www.jettyserver.org/].

  Get jetty-5.1.4 -all.zip because it includes all the required libraries.
- Edit /etc/jetty.xml to configure the JettyPlus server class and add the portal web application.

  ```
  <Configure class="org.mortbay.jetty.plus.Server">
      <Call name="addWebApplication">
          <Arg>/</Arg>
          <Arg>../liferay</Arg>
          <Set name="extractWAR">true</Set>
          <Set
  name="defaultsDescriptor">org/mortbay/jetty/servlet/webdefault.xml</Set>
          <Set name="classLoaderJava2Compliant">true</Set>
      </Call>
  </Configure>
  ```
- Download liferay-portal-pro-4.0.0.war [http://prdownloads.sourceforge.net/lportal/liferay-portal-pro-4.0.0.war].
- Populate your database with the portal schema and default data.
- Edit /extras/etc/start-plus.config.

  ```
  $(jetty.home)/lib/ext/
  $(jetty.home)/lib/ext/*
  ```
- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by Jetty.
- Create a mail session bound to mail/MailSession. You only need to set the locations of the IMAP, POP3, and SMTP servers.

  Edit /etc/jetty.xml and configure a mail session.

  ```
  <Call name="addService">
      <Arg>
          <New class="org.mortbay.jetty.plus.MailService">
              <Set name="Name">MailService</Set>
              <Set name="JNDI">mail/MailSession</Set>
              <Put name="mail.store.protocol">imap</Put>
              <Put name="mail.transport.protocol">smtp</Put>
              <Put name="mail.imap.host">localhost</Put>
              <Put name="mail.pop3.host">localhost</Put>
              <Put name="mail.smtp.host">localhost</Put>
          </New>
      </Arg>
  </Call>
  ```
- Configure JAAS.

  Edit /etc/jetty.xml and configure a security realm.

```
<Call name="addWebApplication">
    <Arg>/</Arg>
    ...
    <Set name="Realm">
        <New class="org.mortbay.jaas.JAASUserRealm">
            <Set name="Name">PortalRealm</Set>
            <Set name="LoginModuleName">PortalRealm</Set>
        </New>
    </Set>
</Call>
```

Create /etc/jaas.config.

```
PortalRealm {
    com.liferay.portal.security.jaas.PortalLoginModule required;
};
```

Make sure the Java command that starts Jetty sets the location of the JAAS config file as a system property.

```
-Djava.security.auth.login.config=../etc/jaas.config
```

- Deploy `liferay-portal-pro-4.0.0.war`.

  Unpack liferay-portal-pro-4.0.0.war to /liferay.

  Move every jar except util-taglib.jar from /liferay/WEB-INF/lib to /lib/ext. This step is only necessary if you plan to hot deploy portlet WARs.

  Edit /etc/jetty.xml to tell Jetty where to find liferay-portal-pro-4.0.0.war.

```
<Call name="addWebApplication">
    <Arg>/</Arg>
    <Arg>../liferay</Arg>
    <Set name="extractWAR">true</Set>
    <Set
name="defaultsDescriptor">org/mortbay/jetty/servlet/webdefault.xml</Set>
    <Set name="classLoaderJava2Compliant">true</Set>
    <Set name="Realm">
        <New class="org.mortbay.jaas.JAASUserRealm">
            <Set name="Name">PortalRealm</Set>
            <Set name="LoginModuleName">PortalRealm</Set>
        </New>
    </Set>
</Call>
```

- Start Jetty.

  If you get a `java.lang.OutOfMemoryError` exception while starting up Jetty, give your JVM more memory by setting -Xmx512m.

  Make sure the Java command that starts Jetty sets the location of the Jetty config file as a system property.

```
-DSTART=../extra/etc/start-plus.config
```

Jetty also needs to know the location of its main configuration file when you start it.

```
java -D... -jar ../start.jar ../etc/jetty.xml
```

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

JFox

We plan on adding support for JFox as soon as possible. Please contact us if you would like to help us in this effort.

JOnAS+Jetty 4.4.3

Easy

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download liferay-portal-ent-4.0.0-jonas-jetty.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0-jonas-jetty.zip].
- Unzip liferay-portal-ent-4.0.0-jonas-jetty.zip to C:\LIFERAY (or any another directory). Be sure to unzip with folder names.
- Set an environment variable called %JONAS_ROOT% to point to where you unzipped liferay-portal-ent-4.0.0-jonas-jetty.zip.
- Execute C:\LIFERAY\bin\nt\jonas.bat to run the database and web server. Make sure there isn't another application using port 8080.

  When in a Unix environment, the batch file to start the server will end with the extension sh instead of bat. Make sure to **chmod** the batch file so you can execute it. You must start the executable from the directory where it resides.
- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

Expert

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download and install JOnAS+Jetty [http://jonas.objectweb.org/].
- Set an environment variable called $JONAS_ROOT to point to where you unzipped JOnAS+Jetty.

- Download liferay-portal-ent-4.0.0.ear [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0.ear].
- Populate your database with the portal schema and default data.
- Open up liferay-portal-ent-4.0.0.ear and repackage cache-ejb.jar, counter-ejb.jar, documentlibrary-ejb.jar, lock-ejb.jar, mail-ejb.jar, and portal-ejb.jar to replace /META-INF/MANIFEST.MF with /META-INF/MANIFEST.MF.JOnAS. JOnAS has a bug where it hangs if different JARs in an EAR reference each other.
- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by JOnAS+Jetty.
- Create a mail session bound to mail/MailSession. You only need to set the locations of the IMAP, POP3, and SMTP servers.

  Set the proper mail properties by editing /conf/mail-session.properties.

  ```
  mail.factory.name=mail/MailSession
  mail.factory.type=javax.mail.Session
  mail.store.protocol=imap
  mail.transport.protocol=smtp
  mail.imap.host=localhost
  mail.pop3.host=localhost
  mail.smtp.host=localhost
  ```

  Edit /conf/jonas.properties so it knows where to look for your mail properties.

  ```
  jonas.service.mail.factories=mail-session
  ```

- Configure JAAS.

  Edit /conf/jaas.config.

  ```
  jetty {
      com.liferay.portal.security.jaas.PortalLoginModule required;
  };
  ```

  Open up liferay-portal-ent-4.0.0.ear and repackage portal-web-complete.war and tunnel-web.war to replace /WEB-INF/web-jetty.xml with /WEB-INF/web-jetty.xml.JOnAS_Jetty.

- Deploy liferay-portal-ent-4.0.0.ear.

  Copy liferay-portal-ent-4.0.0.ear to /apps.

  Edit /conf/jonas.properties and configure the application name.

  ```
  jonas.service.ear.descriptors=liferay-portal-ent-4.0.0.ear
  ```

  Remove /webapps/autoload/ctxroot.war.

- Start JOnAS+Jetty.

  If you get a `java.lang.OutOfMemoryError` exception while starting up JOnAS+Jetty, give your JVM more memory by setting -Xmx512m.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is test@liferay.com and your password is test

JOnAS+Tomcat 4.4.3

Easy

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download liferay-portal-ent-4.0.0-jonas-tomcat.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0-jonas-tomcat.zip].
- Unzip liferay-portal-ent-4.0.0-jonas-tomcat.zip to C:\LIFERAY (or any another directory). Be sure to unzip with folder names.
- Set an environment variable called %JONAS_ROOT% to point to where you unzipped liferay-portal-ent-4.0.0-jonas-tomcat.zip.
- Execute C:\LIFERAY\bin\nt\jonas.bat to run the database and web server. Make sure there isn't another application using port 8080.

  When in a Unix environment, the batch file to start the server will end with the extension sh instead of bat. Make sure to **chmod** the batch file so you can execute it. You must start the executable from the directory where it resides.
- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is `test@liferay.com` and your password is `test`.

Expert

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download and install JOnAS+Tomcat [http://jonas.objectweb.org/].
- Set an environment variable called $JONAS_ROOT to point to where you unzipped JOnAS+Tomcat.
- Download liferay-portal-ent-4.0.0.ear [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0.ear].
- Populate your database with the portal schema and default data.
- Open up liferay-portal-ent-4.0.0.ear and repackage cache-ejb.jar, counter-ejb.jar, documentlibrary-ejb.jar, lock-ejb.jar, mail-ejb.jar, and portal-ejb.jar to replace /META-INF/MANIFEST.MF with /META-INF/MANIFEST.MF.JOnAS. JOnAS has a bug where it hangs if different JARs in an EAR reference each other.
- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by JOnAS+Tomcat.
- Create a mail session bound to mail/MailSession. You only need to set the locations of the IMAP, POP3, and SMTP servers.

Set the proper mail properties by editing /conf/mail-session.properties.

```
mail.factory.name=mail/MailSession
mail.factory.type=javax.mail.Session
mail.store.protocol=imap
mail.transport.protocol=smtp
mail.imap.host=localhost
mail.pop3.host=localhost
mail.smtp.host=localhost
```

Edit /conf/jonas.properties so it knows where to look for your mail properties.

```
jonas.service.mail.factories=mail-session
```

- Configure JAAS.

  Edit /conf/jaas.config.

```
tomcat {
    com.liferay.portal.security.jaas.PortalLoginModule required;
};
```

  Edit /conf/server.xml to use the correct realm.

```
<Realm className="org.objectweb.jonas.security.realm.web.catalina50.JAAS"
/>
```

- Deploy liferay-portal-ent-4.0.0.ear.

  Copy liferay-portal-ent-4.0.0.ear to /apps.

  Edit /conf/jonas.properties and configure the application name.

```
jonas.service.ear.descriptors=liferay-portal-ent-4.0.0.ear
```

  Remove /webapps/autoload/ctxroot.war.

- Start JOnAS+Tomcat.

  If you get a `java.lang.OutOfMemoryError` exception while starting up JOnAS+Tomcat, give your JVM more memory by setting -Xmx512m.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

JRun 4 Updater 3

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.

- Download and install JRun [http://www.macromedia.com/software/jrun/].

- Download liferay-portal-ent-4.0.0.ear [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0.ear].
- Populate your database with the portal schema and default data.
- The dependent libraries are found in /lib inside liferay-portal-ent-4.0.0.ear. Delete the dependent libraries from liferay-portal-ent-4.0.0.ear after you copy them to /servers/lib. JRun has a bug where it cannot properly compile JSPs with Jikes in a Windows environment because the Windows command prompt has a size limitation. JRun also does not give a correct error message but gives a misleading error message stating that JRun cannot find Jikes.
- Remove /servers/default/default-ear.
- Start JRun.

  If you get a `java.lang.OutOfMemoryError` exception while starting up JRun, give your JVM more memory by setting **`-Xmx512m`**.
- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by JRun.
- Create a mail session bound to mail/MailSession. You only need to set the locations of the IMAP, POP3, and SMTP servers.

Note

Configuring mail sessions may not be necessary if you're using JRun Updater 3. The screen for updating mail sessions seems to be missing from JRun Updater 3 and only available in previous versions.

- Configure JAAS.

  Use jrun.security.JDBCLoginModule as your user module class. Point the user module to use jdbc/LiferayPool as the data source. Under user details, set the table name to User_, set the user name to userId, and set the password to password_.

  Use jrun.security.JDBCLoginModule as your role module class. Point the role module to use jdbc/LiferayPool as the data source. Under user query, set the query string to SELECT 'users', 'Roles', COUNT(*) FROM Role_ WHERE roleId NOT LIKE ?.

  The following are links to pages with instructions on how to configure security with JRun.

  http://livedocs.macromedia.com/jrun4docs/JRun_Administrators_Guide/authentic4.jsp

  http://www.macromedia.com/devnet/server_archive/articles/jrun_authentication.html
- Deploy `liferay-portal-ent-4.0.0.ear`.
- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

OracleAS 10.1.2

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download and install OracleAS [http://www.oracle.com/].

- The following are links to pages with instructions on how to configure OracleAS to support JDK 1.4.2 .

  http://otn.oracle.com/tech/java/oc4j/htdocs/OC4J-JSP-FAQ.html

  http://www.rollerweblogger.org/page/roller/20021017

- OracleAS incorrectly detects duplicate page directives. To workaround this bug, set forgive_dup_dir_attr to true.

  http://technet.oracle.com/tech/java/oc4j/htdocs/OC4J-JSP-FAQ.html#Attribute%20defined%20twice [http://technet.oracle.com/tech/java/oc4j/htdocs/OC4J-JSP-FAQ.html#Attribute%20defined%20twice]

- Download liferay-portal-ent-4.0.0.ear [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0.ear].

- Populate your database with the portal schema and default data.

- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by OracleAS.

- Create a mail session bound to mail/MailSession. You only need to set the locations of the IMAP, POP3, and SMTP servers.

  Edit orion-application.xml, found in liferay-portal-ent-4.0.0.ear and configure a mail session.

  ```
  <mail-session location="mail/MailSession" smtp-host="localhost">
      <property name="mail.transport.protocol" value="smtp" />
      <property name="mail.imap.host" value="localhost" />
      <property name="mail.pop3.host" value="localhost" />
  </mail-session>
  ```

- Configure JAAS.

  Edit orion-application.xml, found in liferay-portal-ent-4.0.0.ear, and configure a user manager.

  ```
  <user-manager class="com.evermind.sql.DataSourceUserManager">
      <property name="dataSource" value="jdbc/LiferayPool" />
      <property name="table" value="User_" />
      <property name="usernameField" value="userId" />
      <property name="passwordField" value="password_" />
      <property name="defaultGroups" value="users" />
      <property name="staleness" value="0" />
  </user-manager>
  ```

- Deploy liferay-portal-ent-4.0.0.ear.

  Copy liferay-portal-ent-4.0.0.ear to /j2ee/home/applications.

  Edit /j2ee/home/config/server.xml to tell OracleAS where to find liferay-portal-ent-4.0.0.ear.

  ```
  <application name="liferay"
  path="../applications/liferay-portal-ent-4.0.0.ear" />
  ```

  Edit /j2ee/home/config/server.xml and add a new web site entry.

  ```
  <web-site path="./web-sites/liferay.com-web.xml" />
  ```

  Create /j2ee/home/config/web-sites/liferay.com-web.xml.

```
<web-site port="8080">
    <default-web-app application="liferay" name="portal-web-complete"
load-on-startup="true" />
    <web-app application="liferay" name="tunnel-web" root="/tunnel"
load-on-startup="true" />
    <access-log path="../../log/liferay.com-web-access.log" />
</web-site>
```

Make sure to set load-on-startup to true. OracleAS has a bug that will not allow the servlet init method to call secured EJBs unless that init method is called during startup.

- Start OracleAS.

  If you get a `java.lang.OutOfMemoryError` exception while starting up OracleAS, give your JVM more memory by setting -Xmx512m.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

Orion 2.0.6

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download and install Orion [http://www.orionserver.com/].
- Download liferay-portal-ent-4.0.0.ear [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0.ear].
- Populate your database with the portal schema and default data.
- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by Orion.
- Create a mail session bound to mail/MailSession. You only need to set the locations of the IMAP, POP3, and SMTP servers.

  Edit orion-application.xml, found in liferay-portal-ent-4.0.0.ear and configure a mail session.

```
<mail-session location="mail/MailSession" smtp-host="localhost">
    <property name="mail.transport.protocol" value="smtp" />
    <property name="mail.imap.host" value="localhost" />
    <property name="mail.pop3.host" value="localhost" />
</mail-session>
```

- Configure JAAS.

  Edit orion-application.xml, found in liferay-portal-ent-4.0.0.ear, and configure a user manager.

```
<user-manager class="com.evermind.sql.DataSourceUserManager">
    <property name="dataSource" value="jdbc/LiferayPool" />
    <property name="table" value="User_" />
    <property name="usernameField" value="userId" />
    <property name="passwordField" value="password_" />
    <property name="defaultGroups" value="users" />
    <property name="staleness" value="0" />
```

```
</user-manager>
```

- Deploy liferay-portal-ent-4.0.0.ear.

  Copy liferay-portal-ent-4.0.0.ear to /applications.

  Edit /config/server.xml to tell Orion where to find liferay-portal-ent-4.0.0.ear.

  ```
  <application name="liferay"
  path="../applications/liferay-portal-ent-4.0.0.ear" />
  ```

  Edit /config/server.xml and add a new web site entry.

  ```
  <web-site path="./web-sites/liferay.com-web.xml" />
  ```

  Create /config/web-sites/liferay.com-web.xml.

  ```
  <web-site port="8080">
      <default-web-app application="liferay" name="portal-web-complete"
  load-on-startup="true" />
      <web-app application="liferay" name="tunnel-web" root="/tunnel"
  load-on-startup="true" />
      <access-log path="../../log/liferay.com-web-access.log" />
  </web-site>
  ```

  Make sure to set load-on-startup to true.

- Start Orion.

  If you get a `java.lang.OutOfMemoryError` exception while starting up Orion, give your JVM more memory by setting -Xmx512m.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

Pramati 4.1

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download and install Pramati [http://www.pramati.com/].
- Download liferay-portal-ent-4.0.0.ear [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0.ear].
- Populate your database with the portal schema and default data.
- Edit /server/bin/setup.bat. Add the following snippet right after Pramati declares the CLASSPATH. Download lcp.bat. This script will load everything in /server/lib/ext into the class path.

  ```
  set LOCALCLASSPATH=
  for %%i in ("lib\*.jar") do call "lcp.bat" %%i
  ```

```
set CLASSPATH=%CLASSPATH%;%LOCALCLASSPATH%
```

- Start Pramati.

  If you get a `java.lang.OutOfMemoryError` exception while starting up Pramati, give your JVM more memory by setting -Xmx512m.

- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by Pramati.
- Create a mail session bound to mail/MailSession. You only need to set the locations of the IMAP, POP3, and SMTP servers.
- Deploy liferay-portal-ent-4.0.0.ear.

  We recommend that you use the Deploy Tool instead of the Admin Console because the Deploy Tool is a lot more stable for large EARs. Follow the instructions [http://www.pramati.com/docstore/1290023/liferay.pdf] from Pramati for more information.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

Resin 3.0.14

Easy

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download liferay-portal-pro-4.0.0-resin.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-pro-4.0.0-resin.zip].
- Unzip liferay-portal-pro-4.0.0-resin.zip to C:\LIFERAY (or any another directory). Be sure to unzip with folder names.
- Execute C:\LIFERAY\bin\run.bat to run the database and web server. Make sure that there isn't another application already using port 8080.

  When in a Unix environment, the batch file to start the server will end with the extension sh instead of bat. Make sure to chmod the batch file so you can execute it. You must start the executable from the directory where it resides.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

Expert

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.

- Download and install Resin [http://www.caucho.com/].

- Edit /conf/resin.conf to set up the correct SAX parsers.

  ```
  <system-property
      javax.xml.parsers.DocumentBuilderFactory=
          "org.apache.xerces.jaxp.DocumentBuilderFactoryImpl"
  />
  <system-property
  javax.xml.parsers.SAXParserFactory="org.apache.xerces.jaxp.SAXParserFactoryImpl"
  />
  <system-property
      javax.xml.transform.TransformerFactory=
          "org.apache.xalan.processor.TransformerFactoryImpl"
  />
  <system-property
      org.xml.sax.driver="org.apache.xerces.parsers.SAXParser"
  />
  ```

  Download liferay-portal-ent-4.0.0-src.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0-src.zip].

  Open up liferay-portal-ent-4.0.0-src.zip and copy /portal/lib/saxpath.jar, /portal/lib/xalan.jar/, and /portal/lib/xercesImpl.jar to /lib so that Resin has the proper libraries for SAX parsing.

- Download liferay-portal-pro-4.0.0.war [http://prdownloads.sourceforge.net/lportal/liferay-portal-pro-4.0.0.war].

- Populate your database with the portal schema and default data.

- Edit /conf/resin.conf.

  ```
  <compiling-loader path="${server.rootDir}/common/classes"/>
  <library-loader path="${server.rootDir}/common/lib"/>
  ```

- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by Resin.

- Create a mail session bound to mail/MailSession. You only need to set the locations of the IMAP, POP3, and SMTP servers.

  Edit /conf/resin.conf and configure a mail session.

  ```
  <resource jndi-name="mail/MailSession" type="javax.mail.Session">
      <init>
          <mail.store.protocol>imap</mail.store.protocol>
          <mail.transport.protocol>smtp</mail.transport.protocol>
          <mail.imap.host>localhost</mail.imap.host>
          <mail.pop3.host>localhost</mail.pop3.host>
          <mail.smtp.host>localhost</mail.smtp.host>
      </init>
  </resource>
  ```

- Deploy liferay-portal-pro-4.0.0.war.

  Unpack liferay-portal-pro-4.0.0.war to %CATALINA_HOME%/liferay.

  Move every jar except util-taglib.jar from %CATALINA_HOME%/liferay/WEB-INF/lib to /common/lib/ext. This step is only necessary if you plan to hot deploy portlet WARs.

- Start Resin.

  If you get a `java.lang.OutOfMemoryError` exception while starting up Resin, give your JVM more memory by setting -Xmx512m.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

RexIP 2.5

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.

- Download and install RexIP [http://www.tradecity.com/].

- Download liferay-portal-ent-4.0.0.ear [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0.ear].

- Populate your database with the portal schema and default data.

- Copy the dependent libraries to /lib. The dependent libraries are found in a /lib inside liferay-portal-ent-4.0.0.ear. ReXIP has a bug which prevents it from picking up the dependent libraries declared in MANIFEST.MF.

- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by RexIP.

- Create a mail session bound to mail/MailSession. You only need to set the locations of the IMAP, POP3, and SMTP servers.

  Edit /server.xml and configure a mail session.

```
<mail jndi-name="mail/MailSession" name="mail">
    <properties name="mail.smtp.host" value="localhost"></properties>
</mail>
```

- Configure JAAS.

  Edit server.xml and configure a security realm.

```
<security default-basic-realm="PortalRealm">
    <basic-realm
        check-login-query="SELECT * FROM User_ WHERE userId = ? AND
password_ = ?"
        datasource-name="LiferayPool"
        get-user-query=""
        get-group-query=""
        name="PortalRealm"
        realm-type="jdbc"
        user-role-query="">
    </basic-realm>
</security>
```

- Deploy liferay-portal-ent-4.0.0.ear.

  Edit /server.xml to tell RexIP where to find the unarchived application.

```
<app name="liferayApp" path="liferay/">
    <server name="liferayServer"></server>
</app>
```

- Start RexIP.

  If you get a `java.lang.OutOfMemoryError` exception while starting up RexIP, give your JVM more memory by setting -Xmx512m.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

Sun JSAS 8.01

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.

- Download and install Sun JSAS [http://www.sun.com/software/products/appsrvr/home_appsrvr.xml/].

- Download liferay-portal-ent-4.0.0.ear [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0.ear].

- Populate your database with the portal schema and default data.

- Edit /AppServer/domains/domain1/server1/config/server.policy.

  Change **permission java.io.FilePermission "<>", "read,write";** to **permission java.io.FilePermission "<>", "read,write,execute,delete";**

  Change **permission java.util.PropertyPermission "*", "read";** to **permission java.util.PropertyPermission "*", "read,write";**

  Add **permission java.lang.RuntimePermission "*";**

  Add **permission java.lang.reflect.ReflectPermission "suppressAccessChecks";**

  Add **permission java.security.SecurityPermission "insertProvider.*";**

  Add **permission javax.security.auth.AuthPermission "getLoginConfiguration";**

  Add **permission javax.security.auth.AuthPermission "setLoginConfiguration";**

  Refer to Sun's documentation [http://java.sun.com/j2se/1.4.2/docs/guide/security/permissions.html] for more information.

- Start Sun JSAS.

  If you get a `java.lang.OutOfMemoryError` exception while starting up Sun JSAS, give your JVM more memory by setting -Xmx512m.

  Configure data sources for your database. Make sure the JDBC driver for your database is accessible by Sun JSAS.

  Create a mail session bound to mail/MailSession. You only need to set the locations of the IMAP, POP3, and

SMTP servers.

Configure JAAS.

Create a new security realm called PortalRealm with class name com.liferay.portal.security.jaas.ext.sun8.PortalRealm. Use the class name com.liferay.portal.security.jaas.ext.sun7.PortalRealm if you are using the older Sun ONE 7.

Edit properties for PortalRealm. Add the property jaas-context with the value PortalRealm.

Download liferay-portal-ent-4.0.0-jaas.jar.

Edit the server instance's JVM Settings. Under path settings, add /AppServer/lib/liferay-portal-ent-4.0.0-jaas.jar to the classpath suffix.

Edit /AppServer/domains/domain1/server1/config/login.conf.

```
PortalRealm {
    com.liferay.portal.security.jaas.ext.sun8.PortalLoginModule required;
};
```

Use com.liferay.portal.security.jaas.ext.sun7.PortalLoginModule if you are using the older Sun ONE 7.

Restart Sun JSAS.

- Deploy liferay-portal-ent-4.0.0.ear.

  Restart Sun JSAS because there is a conflict with the default web application that is also listening on /.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

Tomcat 5.0.x/5.5.x

Easy

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.

- Download liferay-portal-pro-4.0.0-tomcat.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-pro-4.0.0-tomcat.zip].

- Unzip liferay-portal-pro-4.0.0-tomcat.zip to C:\LIFERAY (or any another directory). Be sure to unzip with folder names.

- Execute C:\LIFERAY\bin\startup.bat to run the database and web server. Make sure that there isn't another application already using port 8080.

  When in a Unix environment, the batch file to start the server will end with the extension sh instead of bat. Make sure to chmod the batch file so you can execute it. You must start the executable from the directory where it resides.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

Expert

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.

  If you are using Tomcat 5.5.x, you must download and install JDK 5.

- Download and install Tomcat [http://jakarta.apache.org/tomcat/].

  You can download Tomcat 5.0.x or Tomcat 5.5.x. This documentation assumes that you are using Tomcat 5.0.x but will also give special instructions for usage with Tomcat 5.5.x.

- Create /conf/Catalina/localhost/liferay.xml to set up the portal web application.

  ```
  <Context
      path=""
      docBase="../liferay"
      debug="0"
      reloadable="true"
      crossContext="true">
  </Context>
  ```

  For Tomcat 5.5.x, edit /conf/Catalina/localhost/ROOT.xml. You must also remove the reference to path="" in the XML.

- Download liferay-portal-pro-4.0.0.war [http://prdownloads.sourceforge.net/lportal/liferay-portal-pro-4.0.0.war].

- Populate your database with the portal schema and default data.

- Edit /conf/catalina.properties.

  ```
  common.loader=
      ${catalina.home}/common/classes,\
      ...\
      ${catalina.home}/common/lib/ext/*.jar
  ```

- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by Tomcat.

- Create a mail session bound to mail/MailSession. You only need to set the locations of the IMAP, POP3, and SMTP servers.

  Edit /conf/Catalina/localhost/liferay.xml and configure a mail session. For Tomcat 5.5.x, edit / conf/Catalina/localhost/ROOT.xml.

  ```
  <Context...>
      <Resource
          name="mail/MailSession"
          auth="Container"
          type="javax.mail.Session"
      />
      <ResourceParams name="mail/MailSession">
          <parameter>
  ```

```
            <name>mail.store.protocol</name>
            <value>imap</value>
        </parameter>
        <parameter>
            <name>mail.transport.protocol</name>
            <value>smtp</value>
        </parameter>
        <parameter>
            <name>mail.imap.host</name>
            <value>localhost</value>
        </parameter>
        <parameter>
            <name>mail.pop3.host</name>
            <value>localhost</value>
        </parameter>
        <parameter>
            <name>mail.smtp.host</name>
            <value>localhost</value>
        </parameter>
    </ResourceParams>
</Context>
```

- Configure JAAS.

  Edit /conf/Catalina/localhost/liferay.xml and configure a security realm. For Tomcat 5.5.x, edit /conf/Catalina/localhost/ROOT.xml.

```
<Context...>
    <Realm
        className="org.apache.catalina.realm.JAASRealm"
        appName="PortalRealm"
        userClassNames="com.liferay.portal.security.jaas.PortalPrincipal"
        roleClassNames="com.liferay.portal.security.jaas.PortalRole"
        debug="99"
        useContextClassLoader="false"
    />
</Context>
```

  Repeat this step for a file called /conf/Catalina/localhost/tunnel.xml if you want to enable Liferay's HTTP tunneling.

  Create /conf/jaas.config.

```
PortalRealm {
    com.liferay.portal.security.jaas.PortalLoginModule required;};
```

  Edit /bin/catalina.bat so that Tomcat can reference the login module.

```
...

rem ----- Execute...

set JAVA_OPTS=%JAVA_OPTS%
-Djava.security.auth.login.config=%CATALINA_HOME%/conf/jaas.config
```

- Deploy liferay-portal-pro-4.0.0.war.

Unpack liferay-portal-pro-4.0.0.war to %CATALINA_HOME%/liferay.

Move every jar except util-taglib.jar from %CATALINA_HOME%/liferay/WEB-INF/lib to /common/lib/ext. This step is only necessary if you plan to hot deploy portlet WARs.

- Start Tomcat.

  If you get a `java.lang.OutOfMemoryError` exception while starting up Tomcat, give your JVM more memory by setting -Xmx512m.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

WebLogic 8.1 SP4

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download and install WebLogic [http://www.beasys.com/].
- Download liferay-portal-ent-4.0.0.ear [http://prdownloads.sourceforge.net/lportal/liferay-portal-ent-4.0.0.ear].
- Populate your database with the portal schema and default data.
- Edit /user_projects/domains/mydomain/startWebLogic.cmd. Add the following snippet right after WebLogic declares the CLASSPATH. Download lcp.bat [http://content.liferay.com/document/lcp.bat]. This script will load everything in /user_projects/domains/mydomain/lib into the class path.

  ```
  set LOCALCLASSPATH=
  for %%i in ("lib\*.jar") do call "lcp.bat" %%i
  set CLASSPATH=%CLASSPATH%;%LOCALCLASSPATH%

  set MEM_ARGS=-Xmx512m
  ```

  For WebLogic 8.1 SP3, follow the additional instructions from this post [http://forums.liferay.com/index.php?showtopic=763&hl=].

- Start WebLogic.

  If you get a `java.lang.OutOfMemoryError` exception while starting up WebLogic, give your JVM more memory by setting -Xmx512m.

- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by WebLogic.
- Create a mail session bound to mail/MailSession. You only need to set the locations of the IMAP, POP3, and SMTP servers.
- Deploy liferay-portal-ent-4.0.0.ear.
- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

WebSphere 6.0.2.5

Define Environment

- Copy the following under <WAS_HOME>/lib/ext directory.
    - Commons-dbcp.jar
    - Commons-pool.jar
    - System-ext.properties
    - portal-ext.properties
    - Your custom authentication class or the Liferay provided custom authentication class (provided in com.liferay.portal.security.jaas.ext.websphere.PortalUserRegistry.class in liferay-portal-jaas-4.0.0rc2).
    - The JDBC driver used by your custom authentication class, or liferay-jdbc-4.x.jar, if the custom authentication uses the Liferay driver.
- Unzip all the libraries except util-tag.jar from the liferay-portal-4x.war into a new directory (i.e. c:\Liferay\lib).
- Login to the Websphere Admin Console, under Environment, Shared Libraries, ensure “Node” is selected under “Scope”, select Apply.
- Click “New” under “Preferences”.
- Enter a Name (i.e. “LiferayLibs”) and Description. Under “ClassPath”, enter the directory created in Step 2 above (i.e. c:\Liferay\lib”). Click “OK” and “Save” at the top of the page to save your changes.
- Under “Servers”, “Application Servers”, select the application server you’re using (i.e. “server1”).
- Under “Server Infrastructure”, select “Class Loader” under “Java and Process Management”. Click “New”, select “Parent Last” under “Class Loader mode”, click “OK”.
- Under the Class Loader section, you should see the new class loader as “Classloader_12345678”, select this classloader, under “Additional Properties”, select “Add”, and under “Library Name”, you should select the classloader created above (i.e. “LibrayLibs”). Click “OK” and “Save” at the top of the page.
- Restart Websphere.

Define Data Connection

- Login to the Websphere Admin Console
- Under Resources, select JDBC Providers. Be sure “Node” is selected under “Scope”. Click New.
- Under General Properties, Step 1, select “User-Defined” Under General Properties, Step 2, select “User-Defined JDBC Provider” Under General Properties, Step 3, select “User-defined”.
- Click Next.
- Enter in the Name field “Liferay JDBC Provider” (or some other descriptive name)
- Remove any text from the “Class Path” field.
- Click OK, and then “Save” at the top of the page to save your changes before continuing.

- After saving, select the JDBC Provider you just created, click “Data Sources”, and click “New”
- Under General Properties, enter “Liferay” in the “Name” field and “jdbc/LiferayPool” in the “JNDI name” field. Under “Data store helper class name”, leave the default “Generic data store helper” selected. The remaining values (“Authentication Alias”, and “Container-managed Authentication” are not used and should default to “none”)
- Click “OK” and then “Save” at the top of the page to save your changes before continuing.
- Select the “Data Source” you just created (“Liferay”), click “Custom Properties” and add the following, per your database of choice:
  - url
  - password
  - user
  - password
- Save changes.
- Test the connection by selecting under Resources, “JDBC Providers”, “Liferay JDBC Providers” (or whatever you named in step 9 above), and then click “Test Connection”.

Setup Security

- Create your custom login class by creating your own class or using the provided Liferay login class for Websphere (provided in com.liferay.portal.security.jaas.ext.websphere.PortalUserRegistry.class in liferay-portal-jaas-4.0.0rc2)
- In the WebSphere Admin Console, select “Global Security” under the “Security” section. Under “Configuration”, click “Custom” under “User Registries”.
- Enter the custom login class you created or “com.liferay.portal.security.jaas.ext.websphere.PortalUserRegistry” in the “Custom User Registry Class Name” field.
- You may also enter values in the “Server User Id” and “Server User Password” field if you want to use these values when you login to the Admin Console.
- Under “Global Security”, select “Enable global security”, de-select “Enforce Java 2 Security”, and under “Active user registry”, select “Custom user registry”.
- Click OK, and then “Save” at the top of the page to save your changes before continuing.
- If you encounter any errors (class not found, etc), you will see them here when you attempt to save.
- Restart WebSphere and login using the name provided in Step 29 above or using the appropriate database id/password as coded in your custom login class.

Install Liferay

- Login to the Websphere Admin Console
- Ensure the “Default Application” is either removed or not running under “Enterprise Applications”.
- Click “Install New Application”.

- Select the desired Liferay WAR file (i.e. liferay-portal-4x.war) using Local or Remote file system.
- Enter “/” as the Context root, click “Next”.
- Do not change any values on the next page, click “Next”.
- The next page will show “Application Security Warnings”, click “Continue”.
- Enter a valid “Directory to install the application” (i.e. c:\Apps\Liferay”). Under “Application Name”, shorten the name to “Liferay4”, or something similar. Leave all other fields to their default values. Click “Next”.
- Under “Map Modules to Servers”, select the Cells/Webserver targets for your environment, normally you would select all the items in “Clusters and Servers”, click the checkbox under “Select”, then click “Apply”.
- Enter “mail/MailSession” in the “JNDI Name” under “javax.mail.session” section.
- Under the page “Map Resource Reference to Resources”, under “javax.sql.DataSource”, select the checkbox at the bottom of the page under “Select”. Then, under “Specify existing Resource JNDI name”, select “jdbc/LiferayPool”, click the “Apply” button next to this drop-down.
- Select “Next” at the bottom of the page.
- Under “Application Resource Warnings”, click “Continue”.
- Under “Map virtual hosts for Web Modules”, click “Next”.
- Under “Map security roles to users/groups”, across from “users” select the checkbox under “All authenticated”.
- Under the “Summary”, click “Finish”.
- If the application install completed without any errors, click “Save to Master Configuration”.
- Under “Applications”, select the LiferayEAR, under “Class Loading and File Update Detection, change “Class loader mode” to “Parent Last”
- Under “Applications”, select the LiferayEAR application by selecting the check box, then select “Start”.

WebSphere 5.1

- Download and install JDK 1.4.2 [http://java.sun.com/j2se/1.4.2] . Set an environment variable called %JAVA_HOME% to point to your JDK directory.
- Download and install WebSphere [http://www.ibm.com/].
- Download liferay-portal-pro-4.0.0.war [http://prdownloads.sourceforge.net/lportal/liferay-portal-pro-4.0.0.war].
- Populate your database with the portal schema and default data.
- WebSphere does not have access to com.sun.crypto.provider.SunJCE and requires com.ibm.crypto.provider.IBMJCE for the programmatic encryption calls made by the portal.

  Create system-ext.properties in /AppServer/lib/ext and set com.liferay.util.Encryptor.provider.class with the value com.ibm.crypto.provider.IBMJCE.

  This step is not required because Liferay is smart enough to use IBMJCE instead of SunJCE when in Web-Sphere, but will issue a warning in the log files.

- Start WebSphere.

  If you get a `java.lang.OutOfMemoryError` exception while starting up WebSphere, give your JVM more memory by setting -Xmx512m.

- Configure data sources for your database. Make sure the JDBC driver for your database is accessible by WebSphere.

- Configure JAAS.

  Download liferay-portal-ent-4.0.0-jaas.jar into /AppServer/lib/ext.

  Use the Administrative Console to add a custom user registry: Security -> User Registries -> Custom. Set system as the server user id, password as the server user password, and com.liferay.portal.security.jaas.ext.websphere.PortalUserRegistry as the custom registry class name.

  WebSphere is now configured to check security from the jdbc/LiferayPool which points to the Liferay tables. Add a corresponding user to the User_ table.

  ```
  insert into User_ (companyId, userId, password_) values ('system',
  'system', 'password');
  ```

  Edit your Stop the Server shortcut to set the user id and password. If you don't do this, you will not be able to stop the server after your restart WebSphere.

  ```
  "C:\Program Files\WebSphere\AppServer\bin\stopServer.bat" server1 -user
  system -password password
  ```

  Go to Security -> Global Security. Check Enabled. Uncheck Enforce Java 2 Security.

  Set the Active User Registry to Custom.

  Restart WebSphere.

- Deploy liferay-portal-pro-4.0.0.war.

  Open up liferay-portal-pro-4.0.0.war and move every jar except util-taglib.jar from liferay-portal-pro-4.0.0.war to /Anywhere/lib.

  Use the Administrative Console: Applications -> Enterprise Applications -> adminconsole and set the Classloader Mode to PARENT_LAST.

  Go to Environment -> Shared Libraries and create a new library with the name Liferay Libraries and a classpath that points to /Anywhere/lib.

  Go to Servers -> Application Servers -> server1 -> Classloader and create a new classloader with the Libraries set at Liferay Libraries and the Classloader Mode set at PARENT_FIRST.

  Go to Applications -> Enterprise Applications to stop and uninstall the default application because it is also listening on /.

  Go to Applications -> Install New Application. Upload liferay-portal-pro-4.0.0.war and set the context root to /. Click Next.

  Check Generate Default Bindings, then click Next.

  Click Continue.

Click Next for Step 1.

Set mail/MailSession as the mail session and jdbc/LiferayPool as the data source, then click Next for Step 2.

Click Next for Step 3.

Click Next for Step 4.

Check All Authenticated?, then click Next for Step 5.

Click Finish for Step 6.

Restart WebSphere.

- Open your browser to http://localhost:8080. Click on **My Liferay** at the upper right hand corner to enter the login screen. Your login is **`test@liferay.com`** and your password is **`test`**.

Databases

Liferay Portal Enterprise was designed to be database agnostic. To make this possible, all business logic had to be concentrated in the middle tier with the database being as dumb as possible. This means the portal does not rely on any database specific stored procedures or on the database to generate unique keys.

To generate scripts that create and populate the database, go to /portal-ejb and run the ant command: ant build-db. This command will generate the scripts for DB2, Firebird, Hypersonic, Interbase, JDataStore, MySQL, Oracle, PostgreSQL, and SQL Server.

The generated scripts reside in /sql/portal and are named: portal-db2.sql, portal-firebird.sql, portal-hypersonic.sql, portal-interbase.sql, portal-jdatastore.sql, portal-mysql.sql, portal-oracle.sql, portal-postgresql.sql, and portal-sql-server.sql.

DB2

Create a database in DB2 called lportal. Use Command Center to load the script named portal-db2.sql.

BES with DB2

Create a data source bound to jdbc/LiferayPool by editing jndi-definitions and creating liferay.dar. DAR files are custom to BES. You can use the BES Deployment Descriptor Editor to create the DAR file.

```
<jndi-definitions>
    <visitransact-datasource>
        <jndi-name>jdbc/LiferayPool</jndi-name>
<driver-datasource-jndiname>datasources/JdsLiferayDriver</driver-datasource-jndiname>
        <property>
            <prop-name>connectionType</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>Direct</prop-value>
        </property>
        <property>
            <prop-name>dialect</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>db2</prop-value>
        </property>
    </visitransact-datasource>
    <driver-datasource>
        <jndi-name>datasources/JdsLiferayDriver</jndi-name>
<datasource-class-name>org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS</datasource-c
        <log-writer>False</log-writer>
        <property>
            <prop-name>driver</prop-name>
            <prop-type>String</prop-type>
            <prop-value>com.liferay.jdbc.LiferayDriver</prop-value>
        </property>
        <property>
            <prop-name>url</prop-name>
            <prop-type>String</prop-type>
            <prop-value>jdbc:db2://localhost:50000/lportal</prop-value>
        </property>
        <property>
            <prop-name>username</prop-name>
            <prop-type>String</prop-type>
            <prop-value>test</prop-value>
        </property>
        <property>
            <prop-name>password</prop-name>
            <prop-type>String</prop-type>
            <prop-value>test</prop-value>
        </property>
    </driver-datasource>
</jndi-definitions>
```

Copy the JDBC driver to a path that can be picked up by BES. JDBC drivers can be found from the database vendor's web site.

JBoss with DB2

Create a data source bound to jdbc/LiferayPool by editing liferay-ds.xml.

```
<datasources>
   <local-tx-datasource>
     <jndi-name>jdbc/LiferayPool</jndi-name>
     <connection-url>
         jdbc:db2://localhost:50000/lportal
     </connection-url>
     <driver-class>com.liferay.jdbc.LiferayDriver</driver-class>
     <user-name>test</user-name>
     <password>test</password>
     <min-pool-size>0</min-pool-size>
   </local-tx-datasource>
</datasources>
```

Copy the JDBC driver for to /server/default/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Jetty with DB2

Create a data source bound to jdbc/LiferayPool by editing /etc/jetty.xml.

```
<Call name="addService">
    <Arg>
        <New class="org.mortbay.jetty.plus.JotmService">
            <Set name="Name">TransactionMgr</Set>
            <Call name="addDataSource">
```

```
                <Arg>jdbc/LiferayPool</Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.standard.StandardXADataSource">
                        <Set
name="DriverName">com.liferay.jdbc.LiferayDriver</Set>
                        <Set
name="Url">jdbc:db2://localhost:50000/lportal</Set>
                        <Set name="User">test</Set>
                        <Set name="Password">test</Set>
                    </New>
                </Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.pool.StandardXAPoolDataSource">
                        <Arg type="Integer">4</Arg>
                        <Set name="MinSize">4</Set>
                        <Set name="MaxSize">15</Set>
                    </New>
                </Arg>
            </Call>
        </New>
    </Arg>
</Call>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

JOnAS with DB2

Create a data source bound to jdbc/LiferayPool by editing /conf/jonas/liferay-ds.properties.

```
datasource.name=jdbc/LiferayPool
datasource.url=jdbc:db2://localhost:50000/lportal
datasource.classname=com.liferay.jdbc.LiferayDriver
datasource.username=test
datasource.password=test
datasource.mapper=
```

Copy the JDBC driver for to /lib/ext. JDBC drivers can be found from the database vendor's web site.

JRun with DB2

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:db2://localhost:50000/lportal as the URL, com.liferay.jdbc.LiferayDriver as the driver, test as the user name, and test as the password.

OracleAS with DB2

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="com.liferay.jdbc.LiferayDriver"
    url="jdbc:db2://localhost:50000/lportal"
    username="test"
    password="test"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver to /j2ee/home/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Orion with DB2

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="com.liferay.jdbc.LiferayDriver"
    url="jdbc:db2://localhost:50000/lportal"
    username="test"
    password="test"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Pramati with DB2

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:db2://localhost:50000/lportal as the URL, com.liferay.jdbc.LiferayDriver as the driver, test as the user name, and test as the password.

Resin with DB2

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /conf/resin.conf.

```
<database>
    <jndi-name>jdbc/LiferayPool</jndi-name>
    <driver type="com.liferay.jdbc.LiferayDriver">
        <url>jdbc:db2://localhost:50000/lportal</url>
        <user>test</user>
        <password>test</password>
    </driver>
    <prepared-statement-cache-size>8</prepared-statement-cache-size>
    <max-connections>20</max-connections>
    <max-idle-time>30s</max-idle-time>
</database>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

RexIP with DB2

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /server.xml.

```
<datasource
    name="LiferayPool"
    jndi-name="jdbc/LiferayPool"
    driver="com.liferay.jdbc.LiferayDriver"
    url="jdbc:db2://localhost:50000/lportal"
    username="test"
    password="test"
    max-pool="100"
    transactional="false"
/>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Sun JSAS with DB2

Use the Administration Console to do the following:

Create a connection pool org.apache.commons.dbcp.BasicDataSource as the data source class. The pool properties are driverClassName=com.liferay.jdbc.LiferayDriver, url=jdbc:db2://localhost:50000/lportal, username=test, and password=test.

Create a data source bound to jdbc/LiferayPool.

Tomcat with DB2

For Tomcat 5.0.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/liferay.xml.

```
<Context...>
    ...
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
    />
    <ResourceParams name="jdbc/LiferayPool">
        <parameter>
            <name>driverClassName</name>
            <value>com.liferay.jdbc.LiferayDriver</value>
        </parameter>
        <parameter>
            <name>url</name>
            <value>jdbc:db2://localhost:50000/lportal</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>test</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>test</value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>20</value>
        </parameter>
    </ResourceParams>
</Context>
```

For Tomcat 5.5.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/ROOT.xml.

```
<Context...>
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="com.liferay.jdbc.LiferayDriver"
        url="jdbc:db2://localhost:50000/lportal"
        username="test"
        password="test"
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
    />
</Context>
```

For Tomcat 5.0.x and 5.5.x, copy the JDBC driver to /common/lib. JDBC drivers can be found from the database vendor's web site.

WebLogic with DB2

Use the Administration Console to do the following:

Create a connection pool using jdbc:db2://localhost:50000/lportal as the URL, com.liferay.jdbc.LiferayDriver as the driver, test as the user name, and test as the password.

Create a data source bound to jdbc/LiferayPool.

Create a Tx data source bound to jdbc/LiferayEJB.

WebSphere with DB2

Use the Administrative Console: Resources -> JDBC Providers.

Create a new user-defined JDBC provider. Remove any references in the class path. Set org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS as the implementation class name.

Copy commons-dbcp.jar, commons-pool.jar, and your JDBC driver to /AppServer/lib/ext.

Create a data source bound to jdbc/LiferayPool. The custom properties are driver=com.liferay.jdbc.LiferayDriver, url=jdbc:db2://localhost:50000/lportal, user=test, and password=test.

Firebird

Create a database in InterBase located at /opt/data/lportal.gdb. At your UNIX shell or DOS prompt, type isql -U sysdba -P masterkey -d /opt/data/lportal.gdb -i portal-interbase.sql.

BES with Firebird

Create a data source bound to jdbc/LiferayPool by editing jndi-definitions and creating liferay.dar. DAR files are custom to BES. You can use the BES Deployment Descriptor Editor to create the DAR file.

```
<jndi-definitions>
    <visitransact-datasource>
        <jndi-name>jdbc/LiferayPool</jndi-name>
<driver-datasource-jndiname>datasources/JdsLiferayDriver</driver-datasource-jndiname>
        <property>
            <prop-name>connectionType</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>Direct</prop-value>
        </property>
        <property>
            <prop-name>dialect</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>firebird</prop-value>
        </property>
    </visitransact-datasource>
    <driver-datasource>
        <jndi-name>datasources/JdsLiferayDriver</jndi-name>
<datasource-class-name>org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS</datasource-c
        <log-writer>False</log-writer>
        <property>
            <prop-name>driver</prop-name>
            <prop-type>String</prop-type>
            <prop-value>org.firebirdsql.jdbc.FBDriver</prop-value>
        </property>
        <property>
            <prop-name>url</prop-name>
            <prop-type>String</prop-type>
<prop-value>jdbc:firebirdsql://localhost/opt/data/lportal.gdb</prop-value>
        </property>
        <property>
            <prop-name>username</prop-name>
            <prop-type>String</prop-type>
            <prop-value>sysdba</prop-value>
        </property>
        <property>
            <prop-name>password</prop-name>
            <prop-type>String</prop-type>
            <prop-value>masterkey</prop-value>
        </property>
    </driver-datasource>
</jndi-definitions>
```

Copy the JDBC driver to a path that can be picked up by BES. JDBC drivers can be found from the database vendor's web site.

JBoss with Firebird

Create a data source bound to jdbc/LiferayPool by editing liferay-ds.xml.

```
<datasources>
   <local-tx-datasource>
```

```
     <jndi-name>jdbc/LiferayPool</jndi-name>
     <connection-url>
         jdbc:firebirdsql://localhost/opt/data/lportal.gdb
     </connection-url>
     <driver-class>org.firebirdsql.jdbc.FBDriver</driver-class>
     <user-name>sysdba</user-name>
     <password>masterkey</password>
     <min-pool-size>0</min-pool-size>
   </local-tx-datasource>
</datasources>
```

Copy the JDBC driver for to /server/default/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Jetty with Firebird

Create a data source bound to jdbc/LiferayPool by editing /etc/jetty.xml.

```
<Call name="addService">
    <Arg>
        <New class="org.mortbay.jetty.plus.JotmService">
            <Set name="Name">TransactionMgr</Set>
            <Call name="addDataSource">
                <Arg>jdbc/LiferayPool</Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.standard.StandardXADataSource">
                        <Set
name="DriverName">org.firebirdsql.jdbc.FBDriver</Set>
                        <Set
name="Url">jdbc:firebirdsql://localhost/opt/data/lportal.gdb</Set>
                        <Set name="User">sysdba</Set>
                        <Set name="Password">masterkey</Set>
                    </New>
                </Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.pool.StandardXAPoolDataSource">
                        <Arg type="Integer">4</Arg>
                        <Set name="MinSize">4</Set>
                        <Set name="MaxSize">15</Set>
                    </New>
                </Arg>
            </Call>
        </New>
    </Arg>
</Call>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

JOnAS with Firebird

Create a data source bound to jdbc/LiferayPool by editing /conf/jonas/liferay-ds.properties.

```
datasource.name=jdbc/LiferayPool
datasource.url=jdbc:firebirdsql://localhost/opt/data/lportal.gdb
datasource.classname=org.firebirdsql.jdbc.FBDriver
datasource.username=sysdba
datasource.password=masterkey
datasource.mapper=
```

Copy the JDBC driver for to /lib/ext. JDBC drivers can be found from the database vendor's web site.

JRun with Firebird

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:firebirdsql://localhost/opt/data/lportal.gdb as the URL, org.firebirdsql.jdbc.FBDriver as the driver, sysdba as the user name, and masterkey as the password.

OracleAS with Firebird

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="org.firebirdsql.jdbc.FBDriver"
    url="jdbc:firebirdsql://localhost/opt/data/lportal.gdb"
    username="sysdba"
    password="masterkey"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver to /j2ee/home/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Orion with Firebird

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="org.firebirdsql.jdbc.FBDriver"
    url="jdbc:firebirdsql://localhost/opt/data/lportal.gdb"
    username="sysdba"
    password="masterkey"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't already another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Pramati with Firebird

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:firebirdsql://localhost/opt/data/lportal.gdb as the URL, org.firebirdsql.jdbc.FBDriver as the driver, sysdba as the user name, and masterkey as the password.

Resin with Firebird

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /conf/resin.conf.

```
<database>
    <jndi-name>jdbc/LiferayPool</jndi-name>
    <driver type="org.firebirdsql.jdbc.FBDriver">
        <url>jdbc:firebirdsql://localhost/opt/data/lportal.gdb</url>
        <user>sysdba</user>
        <password>masterkey</password>
    </driver>
    <prepared-statement-cache-size>8</prepared-statement-cache-size>
    <max-connections>20</max-connections>
    <max-idle-time>30s</max-idle-time>
</database>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

RexIP with Firebird

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /server.xml.

```
<datasource
    name="LiferayPool"
    jndi-name="jdbc/LiferayPool"
    driver="org.firebirdsql.jdbc.FBDriver"
    url="jdbc:firebirdsql://localhost/opt/data/lportal.gdb"
    username="sysdba"
    password="masterkey"
    max-pool="100"
    transactional="false"
/>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't already another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Sun JSAS with Firebird

Use the Administration Console to do the following:

Create a connection pool org.apache.commons.dbcp.BasicDataSource as the data source class. The pool properties are driverClassName=org.firebirdsql.jdbc.FBDriver, url=jdbc:firebirdsql://localhost/opt/data/lportal.gdb, username=sysdba, and password=masterkey.

Create a data source bound to jdbc/LiferayPool.

Tomcat with Firebird

For Tomcat 5.0.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/liferay.xml.

```
<Context...>
    ...
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
    />
    <ResourceParams name="jdbc/LiferayPool">
        <parameter>
            <name>driverClassName</name>
            <value>org.firebirdsql.jdbc.FBDriver</value>
        </parameter>
        <parameter>
            <name>url</name>
            <value>jdbc:firebirdsql://localhost/opt/data/lportal.gdb</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>sysdba</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>masterkey</value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>20</value>
        </parameter>
    </ResourceParams>
</Context>
```

For Tomcat 5.5.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/ROOT.xml.

```
<Context...>
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="org.firebirdsql.jdbc.FBDriver"
        url="jdbc:firebirdsql://localhost/opt/data/lportal.gdb"
        username="sysdba"
        password="masterkey"
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
    />
</Context>
```

For Tomcat 5.0.x and 5.5.x, copy the JDBC driver to /common/lib. JDBC drivers can be found from the database vendor's web site.

WebLogic with Firebird

Use the Administration Console to do the following:

Create a connection pool using jdbc:firebirdsql://localhost/opt/data/lportal.gdb as the URL, org.firebirdsql.jdbc.FBDriver as the driver, sysdba as the user name, and masterkey as the password.

Create a data source bound to jdbc/LiferayPool.

Create a Tx data source bound to jdbc/LiferayEJB.

WebSphere with Firebird

Use the Administrative Console: Resources -> JDBC Providers.

Create a new user-defined JDBC provider. Remove any references in the class path. Set org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS as the implementation class name.

Copy commons-dbcp.jar, commons-pool.jar, and your JDBC driver to /AppServer/lib/ext.

Create a data source bound to jdbc/LiferayPool. The custom properties are driver=org.firebirdsql.jdbc.FBDriver, url=jdbc:firebirdsql://localhost/opt/data/lportal.gdb, user=sysdba, and password=masterkey.

Hypersonic

Run the Hypersonic Database Manager and log into the database instance. Cut and paste the contents of portal-hypersonic.sql into the manager and excecute the SQL statements.

BES with Hypersonic

Create a data source bound to jdbc/LiferayPool by editing jndi-definitions and creating liferay.dar. DAR files are custom to BES. You can use the BES Deployment Descriptor Editor to create the DAR file.

```
<jndi-definitions>
    <visitransact-datasource>
        <jndi-name>jdbc/LiferayPool</jndi-name>
<driver-datasource-jndiname>datasources/JdsLiferayDriver</driver-datasource-jndiname>
        <property>
            <prop-name>connectionType</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>Direct</prop-value>
        </property>
        <property>
            <prop-name>dialect</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>hypersonic</prop-value>
        </property>
    </visitransact-datasource>
    <driver-datasource>
        <jndi-name>datasources/JdsLiferayDriver</jndi-name>
<datasource-class-name>org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS</datasource-c
        <log-writer>False</log-writer>
        <property>
            <prop-name>driver</prop-name>
            <prop-type>String</prop-type>
            <prop-value>org.hsqldb.jdbcDriver</prop-value>
        </property>
        <property>
            <prop-name>url</prop-name>
            <prop-type>String</prop-type>
            <prop-value>jdbc:hsqldb:hsql://localhost</prop-value>
        </property>
        <property>
            <prop-name>username</prop-name>
            <prop-type>String</prop-type>
            <prop-value></prop-value>
        </property>
        <property>
            <prop-name>password</prop-name>
            <prop-type>String</prop-type>
            <prop-value></prop-value>
        </property>
    </driver-datasource>
</jndi-definitions>
```

Copy the JDBC driver to a path that can be picked up by BES. JDBC drivers can be found from the database vendor's web site.

JBoss with Hypersonic

Create a data source bound to jdbc/LiferayPool by editing liferay-ds.xml.

```
<datasources>
   <local-tx-datasource>
     <jndi-name>jdbc/LiferayPool</jndi-name>
     <connection-url>
         jdbc:hsqldb:hsql://localhost
     </connection-url>
     <driver-class>org.hsqldb.jdbcDriver</driver-class>
     <user-name></user-name>
     <password></password>
     <min-pool-size>0</min-pool-size>
   </local-tx-datasource>
</datasources>
```

Copy the JDBC driver for to /server/default/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Jetty with Hypersonic

Create a data source bound to jdbc/LiferayPool by editing /etc/jetty.xml.

```
<Call name="addService">
    <Arg>
        <New class="org.mortbay.jetty.plus.JotmService">
            <Set name="Name">TransactionMgr</Set>
            <Call name="addDataSource">
                <Arg>jdbc/LiferayPool</Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.standard.StandardXADataSource">
                        <Set name="DriverName">org.hsqldb.jdbcDriver</Set>
                        <Set name="Url">jdbc:hsqldb:hsql://localhost</Set>
                        <Set name="User"></Set>
                        <Set name="Password"></Set>
                    </New>
                </Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.pool.StandardXAPoolDataSource">
```

```
                        <Arg type="Integer">4</Arg>
                        <Set name="MinSize">4</Set>
                        <Set name="MaxSize">15</Set>
                    </New>
                </Arg>
            </Call>
        </New>
    </Arg>
</Call>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

JOnAS with Hypersonic

Create a data source bound to jdbc/LiferayPool by editing /conf/jonas/liferay-ds.properties.

```
datasource.name=jdbc/LiferayPool
datasource.url=jdbc:hsqldb:hsql://localhost
datasource.classname=org.hsqldb.jdbcDriver
datasource.username=
datasource.password=
datasource.mapper=
```

Copy the JDBC driver for to /lib/ext. JDBC drivers can be found from the database vendor's web site.

JRun with Hypersonic

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:hsqldb:hsql://localhost as the URL, org.hsqldb.jdbcDriver as the driver, as the user name, and as the password.

OracleAS with Hypersonic

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="org.hsqldb.jdbcDriver"
    url="jdbc:hsqldb:hsql://localhost"
    username=""
    password=""
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver to /j2ee/home/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Orion with Hypersonic

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="org.hsqldb.jdbcDriver"
    url="jdbc:hsqldb:hsql://localhost"
    username=""
    password=""
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Pramati with Hypersonic

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:hsqldb:hsql://localhost as the URL, org.hsqldb.jdbcDriver as the driver, as the user name, and as the password.

Resin with Hypersonic

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /conf/resin.conf.

```
<database>
    <jndi-name>jdbc/LiferayPool</jndi-name>
    <driver type="org.hsqldb.jdbcDriver">
        <url>jdbc:hsqldb:hsql://localhost</url>
        <user></user>
        <password></password>
    </driver>
    <prepared-statement-cache-size>8</prepared-statement-cache-size>
    <max-connections>20</max-connections>
    <max-idle-time>30s</max-idle-time>
</database>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

RexIP with Hypersonic

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /server.xml.

```
<datasource
```

```
    name="LiferayPool"
    jndi-name="jdbc/LiferayPool"
    driver="org.hsqldb.jdbcDriver"
    url="jdbc:hsqldb:hsql://localhost"
    username=""
    password=""
    max-pool="100"
    transactional="false"
/>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't already another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Sun JSAS with Hypersonic

Use the Administration Console to do the following:

Create a connection pool org.apache.commons.dbcp.BasicDataSource as the data source class. The pool properties are driverClassName=org.hsqldb.jdbcDriver, url=jdbc:hsqldb:hsql://localhost, username=, and password=.

Create a data source bound to jdbc/LiferayPool.

Tomcat with Hypersonic

For Tomcat 5.0.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/liferay.xml.

```
<Context...>
    ...
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
    />
    <ResourceParams name="jdbc/LiferayPool">
        <parameter>
            <name>driverClassName</name>
            <value>org.hsqldb.jdbcDriver</value>
        </parameter>
        <parameter>
            <name>url</name>
            <value>jdbc:hsqldb:hsql://localhost</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value></value>
        </parameter>
        <parameter>
            <name>password</name>
            <value></value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>20</value>
        </parameter>
    </ResourceParams>
</Context>
```

For Tomcat 5.5.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/ROOT.xml.

```
<Context...>
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="org.hsqldb.jdbcDriver"
        url="jdbc:hsqldb:hsql://localhost"
        username=""
        password=""
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
    />
</Context>
```

For Tomcat 5.0.x and 5.5.x, copy the JDBC driver to /common/lib. JDBC drivers can be found from the database vendor's web site.

WebLogic with Hypersonic

Use the Administration Console to do the following:

Create a connection pool using jdbc:hsqldb:hsql://localhost as the URL, org.hsqldb.jdbcDriver as the driver, as the user name, and as the password.

Create a data source bound to jdbc/LiferayPool.

Create a Tx data source bound to jdbc/LiferayEJB.

WebSphere with Hypersonic

Use the Administrative Console: Resources -> JDBC Providers.

Create a new user-defined JDBC provider. Remove any references in the class path. Set org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS as the implementation class name.

Copy commons-dbcp.jar, commons-pool.jar, and your JDBC driver to /AppServer/lib/ext.

Create a data source bound to jdbc/LiferayPool. The custom properties are driver=org.hsqldb.jdbcDriver, url=jdbc:hsqldb:hsql://localhost, user=, and password=.

InterBase

Create a database in InterBase located at /opt/data/lportal.gdb. At your UNIX shell or DOS prompt, type isql -U sysdba -P masterkey -d /opt/data/lportal.gdb -i portal-firebird.sql.

BES with InterBase

Create a data source bound to jdbc/LiferayPool by editing jndi-definitions and creating liferay.dar. DAR files are custom to BES. You can use the BES Deployment Descriptor Editor to create the DAR file.

```
<jndi-definitions>
    <visitransact-datasource>
        <jndi-name>jdbc/LiferayPool</jndi-name>
<driver-datasource-jndiname>datasources/JdsLiferayDriver</driver-datasource-jndiname>
        <property>
            <prop-name>connectionType</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>Direct</prop-value>
        </property>
        <property>
            <prop-name>dialect</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>interbase</prop-value>
        </property>
    </visitransact-datasource>
    <driver-datasource>
        <jndi-name>datasources/JdsLiferayDriver</jndi-name>
<datasource-class-name>org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS</datasource-c
        <log-writer>False</log-writer>
        <property>
            <prop-name>driver</prop-name>
            <prop-type>String</prop-type>
            <prop-value>interbase.interclient.Driver</prop-value>
        </property>
        <property>
            <prop-name>url</prop-name>
            <prop-type>String</prop-type>
<prop-value>jdbc:interbase://localhost/opt/data/lportal.gdb</prop-value>
        </property>
        <property>
            <prop-name>username</prop-name>
            <prop-type>String</prop-type>
            <prop-value>sysdba</prop-value>
        </property>
        <property>
            <prop-name>password</prop-name>
            <prop-type>String</prop-type>
            <prop-value>masterkey</prop-value>
        </property>
    </driver-datasource>
</jndi-definitions>
```

Copy the JDBC driver to a path that can be picked up by BES. JDBC drivers can be found from the database vendor's web site.

JBoss with InterBase

Create a data source bound to jdbc/LiferayPool by editing liferay-ds.xml.

```
<datasources>
   <local-tx-datasource>
     <jndi-name>jdbc/LiferayPool</jndi-name>
     <connection-url>
         jdbc:interbase://localhost/opt/data/lportal.gdb
     </connection-url>
     <driver-class>interbase.interclient.Driver</driver-class>
     <user-name>sysdba</user-name>
     <password>masterkey</password>
     <min-pool-size>0</min-pool-size>
   </local-tx-datasource>
</datasources>
```

Copy the JDBC driver for to /server/default/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Jetty with InterBase

Create a data source bound to jdbc/LiferayPool by editing /etc/jetty.xml.

```
<Call name="addService">
    <Arg>
        <New class="org.mortbay.jetty.plus.JotmService">
            <Set name="Name">TransactionMgr</Set>
            <Call name="addDataSource">
                <Arg>jdbc/LiferayPool</Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.standard.StandardXADataSource">
                        <Set
name="DriverName">interbase.interclient.Driver</Set>
                        <Set
name="Url">jdbc:interbase://localhost/opt/data/lportal.gdb</Set>
                        <Set name="User">sysdba</Set>
                        <Set name="Password">masterkey</Set>
                    </New>
                </Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.pool.StandardXAPoolDataSource">
                        <Arg type="Integer">4</Arg>
                        <Set name="MinSize">4</Set>
                        <Set name="MaxSize">15</Set>
                    </New>
                </Arg>
            </Call>
        </New>
    </Arg>
</Call>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

JOnAS with InterBase

Create a data source bound to jdbc/LiferayPool by editing /conf/jonas/liferay-ds.properties.

```
datasource.name=jdbc/LiferayPool
datasource.url=jdbc:interbase://localhost/opt/data/lportal.gdb
datasource.classname=interbase.interclient.Driver
datasource.username=sysdba
datasource.password=masterkey
datasource.mapper=
```

Copy the JDBC driver for to /lib/ext. JDBC drivers can be found from the database vendor's web site.

JRun with InterBase

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:interbase://localhost/opt/data/lportal.gdb as the URL, interbase.interclient.Driver as the driver, sysdba as the user name, and masterkey as the password.

OracleAS with InterBase

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="interbase.interclient.Driver"
    url="jdbc:interbase://localhost/opt/data/lportal.gdb"
    username="sysdba"
    password="masterkey"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver to /j2ee/home/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Orion with InterBase

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="interbase.interclient.Driver"
    url="jdbc:interbase://localhost/opt/data/lportal.gdb"
    username="sysdba"
    password="masterkey"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Pramati with InterBase

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:interbase://localhost/opt/data/lportal.gdb as the URL, interbase.interclient.Driver as the driver, sysdba as the user name, and masterkey as the password.

Resin with InterBase

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /conf/resin.conf.

```
<database>
    <jndi-name>jdbc/LiferayPool</jndi-name>
    <driver type="interbase.interclient.Driver">
        <url>jdbc:interbase://localhost/opt/data/lportal.gdb</url>
        <user>sysdba</user>
        <password>masterkey</password>
    </driver>
    <prepared-statement-cache-size>8</prepared-statement-cache-size>
    <max-connections>20</max-connections>
    <max-idle-time>30s</max-idle-time>
</database>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

RexIP with InterBase

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /server.xml.

```
<datasource
    name="LiferayPool"
    jndi-name="jdbc/LiferayPool"
    driver="interbase.interclient.Driver"
    url="jdbc:interbase://localhost/opt/data/lportal.gdb"
    username="sysdba"
    password="masterkey"
    max-pool="100"
    transactional="false"
/>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Sun JSAS with InterBase

Use the Administration Console to do the following:

Create a connection pool org.apache.commons.dbcp.BasicDataSource as the data source class. The pool properties are driverClassName=interbase.interclient.Driver, url=jdbc:interbase://localhost/opt/data/lportal.gdb, user-name=sysdba, and password=masterkey.

Create a data source bound to jdbc/LiferayPool.

Tomcat with InterBase

For Tomcat 5.0.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/liferay.xml.

```
<Context...>
    ...
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
    />
    <ResourceParams name="jdbc/LiferayPool">
        <parameter>
            <name>driverClassName</name>
            <value>interbase.interclient.Driver</value>
        </parameter>
        <parameter>
            <name>url</name>
            <value>jdbc:interbase://localhost/opt/data/lportal.gdb</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>sysdba</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>masterkey</value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>20</value>
        </parameter>
    </ResourceParams>
</Context>
```

For Tomcat 5.5.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/ROOT.xml.

```
<Context...>
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="interbase.interclient.Driver"
        url="jdbc:interbase://localhost/opt/data/lportal.gdb"
        username="sysdba"
        password="masterkey"
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
    />
</Context>
```

For Tomcat 5.0.x and 5.5.x, copy the JDBC driver to /common/lib. JDBC drivers can be found from the database vendor's web site.

WebLogic with InterBase

Use the Administration Console to do the following:

Create a connection pool using jdbc:interbase://localhost/opt/data/lportal.gdb as the URL, interbase.interclient.Driver as the driver, sysdba as the user name, and masterkey as the password.

Create a data source bound to jdbc/LiferayPool.

Create a Tx data source bound to jdbc/LiferayEJB.

WebSphere with InterBase

Use the Administrative Console: Resources -> JDBC Providers.

Create a new user-defined JDBC provider. Remove any references in the class path. Set org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS as the implementation class name.

Copy commons-dbcp.jar, commons-pool.jar, and your JDBC driver to /AppServer/lib/ext.

Create a data source bound to jdbc/LiferayPool. The custom properties are driver=interbase.interclient.Driver, url=jdbc:interbase://localhost/opt/data/lportal.gdb, user=sysdba, and password=masterkey.

JDataStore

Create a database in InterBase located at lportal.jds. At your UNIX shell or DOS prompt, type isql -U sysdba -P masterkey -d lportal.jds -i portal-jdatastore.sql.

BES with JDataStore

Create a data source bound to jdbc/LiferayPool by editing jndi-definitions and creating liferay.dar. DAR files are custom to BES. You can use the BES Deployment Descriptor Editor to create the DAR file.

```
<jndi-definitions>
    <visitransact-datasource>
        <jndi-name>jdbc/LiferayPool</jndi-name>
<driver-datasource-jndiname>datasources/JdsLiferayDriver</driver-datasource-jndiname>
        <property>
            <prop-name>connectionType</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>Direct</prop-value>
        </property>
        <property>
            <prop-name>dialect</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>jdatastore</prop-value>
        </property>
    </visitransact-datasource>
    <driver-datasource>
        <jndi-name>datasources/JdsLiferayDriver</jndi-name>
<datasource-class-name>org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS</datasource-c
        <log-writer>False</log-writer>
        <property>
            <prop-name>driver</prop-name>
            <prop-type>String</prop-type>
<prop-value>com.borland.datastore.jdbc.DataStoreDriver</prop-value>
        </property>
        <property>
            <prop-name>url</prop-name>
            <prop-type>String</prop-type>
            <prop-value>jdbc:borland:dslocal:lportal.jds</prop-value>
        </property>
        <property>
            <prop-name>username</prop-name>
            <prop-type>String</prop-type>
            <prop-value>sysdba</prop-value>
        </property>
        <property>
            <prop-name>password</prop-name>
            <prop-type>String</prop-type>
            <prop-value>masterkey</prop-value>
        </property>
    </driver-datasource>
</jndi-definitions>
```

Copy the JDBC driver to a path that can be picked up by BES. JDBC drivers can be found from the database vendor's web site.

JBoss with JDataStore

Create a data source bound to jdbc/LiferayPool by editing liferay-ds.xml.

```
<datasources>
   <local-tx-datasource>
     <jndi-name>jdbc/LiferayPool</jndi-name>
     <connection-url>
         jdbc:borland:dslocal:lportal.jds
     </connection-url>
     <driver-class>com.borland.datastore.jdbc.DataStoreDriver</driver-class>
     <user-name>sysdba</user-name>
     <password>masterkey</password>
     <min-pool-size>0</min-pool-size>
   </local-tx-datasource>
</datasources>
```

Copy the JDBC driver for to /server/default/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Jetty with JDataStore

Create a data source bound to jdbc/LiferayPool by editing /etc/jetty.xml.

```
<Call name="addService">
    <Arg>
        <New class="org.mortbay.jetty.plus.JotmService">
            <Set name="Name">TransactionMgr</Set>
            <Call name="addDataSource">
                <Arg>jdbc/LiferayPool</Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.standard.StandardXADataSource">
                        <Set
name="DriverName">com.borland.datastore.jdbc.DataStoreDriver</Set>
                        <Set name="Url">jdbc:borland:dslocal:lportal.jds</Set>
                        <Set name="User">sysdba</Set>
                        <Set name="Password">masterkey</Set>
                    </New>
                </Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.pool.StandardXAPoolDataSource">
                        <Arg type="Integer">4</Arg>
                        <Set name="MinSize">4</Set>
                        <Set name="MaxSize">15</Set>
                    </New>
                </Arg>
            </Call>
        </New>
    </Arg>
</Call>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

JOnAS with JDataStore

Create a data source bound to jdbc/LiferayPool by editing /conf/jonas/liferay-ds.properties.

```
datasource.name=jdbc/LiferayPool
datasource.url=jdbc:borland:dslocal:lportal.jds
datasource.classname=com.borland.datastore.jdbc.DataStoreDriver
datasource.username=sysdba
datasource.password=masterkey
datasource.mapper=
```

Copy the JDBC driver for to /lib/ext. JDBC drivers can be found from the database vendor's web site.

JRun with JDataStore

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:borland:dslocal:lportal.jds as the URL, com.borland.datastore.jdbc.DataStoreDriver as the driver, sysdba as the user name, and masterkey as the password.

OracleAS with JDataStore

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="com.borland.datastore.jdbc.DataStoreDriver"
    url="jdbc:borland:dslocal:lportal.jds"
    username="sysdba"
    password="masterkey"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver to /j2ee/home/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Orion with JDataStore

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="com.borland.datastore.jdbc.DataStoreDriver"
    url="jdbc:borland:dslocal:lportal.jds"
    username="sysdba"
    password="masterkey"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Pramati with JDataStore

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:borland:dslocal:lportal.jds as the URL, com.borland.datastore.jdbc.DataStoreDriver as the driver, sysdba as the user name, and masterkey as the password.

Resin with JDataStore

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /conf/resin.conf.

```
<database>
    <jndi-name>jdbc/LiferayPool</jndi-name>
    <driver type="com.borland.datastore.jdbc.DataStoreDriver">
        <url>jdbc:borland:dslocal:lportal.jds</url>
        <user>sysdba</user>
        <password>masterkey</password>
    </driver>
    <prepared-statement-cache-size>8</prepared-statement-cache-size>
    <max-connections>20</max-connections>
    <max-idle-time>30s</max-idle-time>
</database>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

RexIP with JDataStore

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /server.xml.

```
<datasource
    name="LiferayPool"
    jndi-name="jdbc/LiferayPool"
    driver="com.borland.datastore.jdbc.DataStoreDriver"
    url="jdbc:borland:dslocal:lportal.jds"
    username="sysdba"
    password="masterkey"
    max-pool="100"
    transactional="false"
/>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Sun JSAS with JDataStore

Use the Administration Console to do the following:

Create a connection pool org.apache.commons.dbcp.BasicDataSource as the data source class. The pool properties are driverClassName=com.borland.datastore.jdbc.DataStoreDriver, url=jdbc:borland:dslocal:lportal.jds, user-name=sysdba, and password=masterkey.

Create a data source bound to jdbc/LiferayPool.

Tomcat with JDataStore

For Tomcat 5.0.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/liferay.xml.

```
<Context...>
    ...
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
    />
    <ResourceParams name="jdbc/LiferayPool">
        <parameter>
            <name>driverClassName</name>
            <value>com.borland.datastore.jdbc.DataStoreDriver</value>
        </parameter>
        <parameter>
            <name>url</name>
            <value>jdbc:borland:dslocal:lportal.jds</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>sysdba</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>masterkey</value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>20</value>
        </parameter>
    </ResourceParams>
</Context>
```

For Tomcat 5.5.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/ROOT.xml.

```
<Context...>
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="com.borland.datastore.jdbc.DataStoreDriver"
        url="jdbc:borland:dslocal:lportal.jds"
        username="sysdba"
        password="masterkey"
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
    />
</Context>
```

For Tomcat 5.0.x and 5.5.x, copy the JDBC driver to /common/lib. JDBC drivers can be found from the database vendor's web site.

WebLogic with JDataStore

Use the Administration Console to do the following:

Create a connection pool using jdbc:borland:dslocal:lportal.jds as the URL, com.borland.datastore.jdbc.DataStoreDriver as the driver, sysdba as the user name, and masterkey as the password.

Create a data source bound to jdbc/LiferayPool.

Create a Tx data source bound to jdbc/LiferayEJB.

WebSphere with JDataStore

Use the Administrative Console: Resources -> JDBC Providers.

Create a new user-defined JDBC provider. Remove any references in the class path. Set org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS as the implementation class name.

Copy commons-dbcp.jar, commons-pool.jar, and your JDBC driver to /AppServer/lib/ext.

Create a data source bound to jdbc/LiferayPool. The custom properties are driver=com.borland.datastore.jdbc.DataStoreDriver, url=jdbc:borland:dslocal:lportal.jds, user=sysdba, and password=masterkey.

MySQL

Create a database in MySQL called lportal. At your UNIX shell or DOS prompt, type mysql lportal < portal-mysql.sql.

Note

As of version 1.8.0, the old JDBC driver org.gjt.mm.mysql.Driver is replaced with the latest JDBC driver com.mysql.jdbc.Driver. The old driver stored booleans in columns of type enum('t', 'f'), the new driver stores booleans in columns of type tinyint.

To manually update your column type, execute the SQL command alter table TableName modify column columnName tinyint;. To automatically update all your tables, execute the SQL script update1.7.5-1.8.0-mysql.sql. Follow the additional instructions from this post if you're having character encoding issues.

BES with MySQL

Create a data source bound to jdbc/LiferayPool by editing jndi-definitions and creating liferay.dar. DAR files are custom to BES. You can use the BES Deployment Descriptor Editor to create the DAR file.

```
<jndi-definitions>
    <visitransact-datasource>
        <jndi-name>jdbc/LiferayPool</jndi-name>
<driver-datasource-jndiname>datasources/JdsLiferayDriver</driver-datasource-jndiname>
        <property>
```

```
            <prop-name>connectionType</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>Direct</prop-value>
        </property>
        <property>
            <prop-name>dialect</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>mysql</prop-value>
        </property>
    </visitransact-datasource>
    <driver-datasource>
        <jndi-name>datasources/JdsLiferayDriver</jndi-name>
<datasource-class-name>org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS</datasource-c
        <log-writer>False</log-writer>
        <property>
            <prop-name>driver</prop-name>
            <prop-type>String</prop-type>
            <prop-value>com.mysql.jdbc.Driver</prop-value>
        </property>
        <property>
            <prop-name>url</prop-name>
            <prop-type>String</prop-type>
<prop-value>jdbc:mysql://localhost/lportal?useUnicode=true&amp;characterEncoding=UTF-8</pr
        </property>
        <property>
            <prop-name>username</prop-name>
            <prop-type>String</prop-type>
            <prop-value>test</prop-value>
        </property>
        <property>
            <prop-name>password</prop-name>
            <prop-type>String</prop-type>
            <prop-value>test</prop-value>
        </property>
    </driver-datasource>
</jndi-definitions>
```

Copy the JDBC driver to a path that can be picked up by BES. JDBC drivers can be found from the database vendor's web site.

JBoss with MySQL

Create a data source bound to jdbc/LiferayPool by editing liferay-ds.xml.

```
<datasources>
   <local-tx-datasource>
     <jndi-name>jdbc/LiferayPool</jndi-name>
     <connection-url>
jdbc:mysql://localhost/lportal?useUnicode=true&amp;characterEncoding=UTF-8
     </connection-url>
     <driver-class>com.mysql.jdbc.Driver</driver-class>
     <user-name>test</user-name>
     <password>test</password>
     <min-pool-size>0</min-pool-size>
   </local-tx-datasource>
</datasources>
```

Copy the JDBC driver for to /server/default/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Jetty with MySQL

Create a data source bound to jdbc/LiferayPool by editing /etc/jetty.xml.

```
<Call name="addService">
    <Arg>
        <New class="org.mortbay.jetty.plus.JotmService">
            <Set name="Name">TransactionMgr</Set>
            <Call name="addDataSource">
                <Arg>jdbc/LiferayPool</Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.standard.StandardXADataSource">
                        <Set name="DriverName">com.mysql.jdbc.Driver</Set>
                        <Set
name="Url">jdbc:mysql://localhost/lportal?useUnicode=true&amp;characterEncoding=UTF-8</Set
                        <Set name="User">test</Set>
                        <Set name="Password">test</Set>
                    </New>
                </Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.pool.StandardXAPoolDataSource">
                        <Arg type="Integer">4</Arg>
                        <Set name="MinSize">4</Set>
                        <Set name="MaxSize">15</Set>
                    </New>
                </Arg>
            </Call>
        </New>
    </Arg>
</Call>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

JOnAS with MySQL

Create a data source bound to jdbc/LiferayPool by editing /conf/jonas/liferay-ds.properties.

```
datasource.name=jdbc/LiferayPool
datasource.url=jdbc:mysql://localhost/lportal?useUnicode=true&characterEncoding=UTF-8
datasource.classname=com.mysql.jdbc.Driver
datasource.username=test
datasource.password=test
datasource.mapper=
```

Copy the JDBC driver for to /lib/ext. JDBC drivers can be found from the database vendor's web site.

JRun with MySQL

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jd-bc:mysql://localhost/lportal?useUnicode=true&characterEncoding=UTF-8 as the URL, com.mysql.jdbc.Driver as the driver, test as the user name, and test as the password.

OracleAS with MySQL

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost/lportal?useUnicode=true&characterEncoding=UTF-8"
    username="test"
    password="test"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver to /j2ee/home/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Orion with MySQL

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost/lportal?useUnicode=true&characterEncoding=UTF-8"
    username="test"
    password="test"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Pramati with MySQL

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:mysql://localhost/lportal?useUnicode=true&characterEncoding=UTF-8 as the URL, com.mysql.jdbc.Driver as the driver, test as the user name, and test as the password.

Resin with MySQL

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /conf/resin.conf.

```
<database>
    <jndi-name>jdbc/LiferayPool</jndi-name>
    <driver type="com.mysql.jdbc.Driver">
<url>jdbc:mysql://localhost/lportal?useUnicode=true&amp;characterEncoding=UTF-8</url>
        <user>test</user>
        <password>test</password>
    </driver>
    <prepared-statement-cache-size>8</prepared-statement-cache-size>
    <max-connections>20</max-connections>
    <max-idle-time>30s</max-idle-time>
</database>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

RexIP with MySQL

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /server.xml.

```
<datasource
    name="LiferayPool"
    jndi-name="jdbc/LiferayPool"
    driver="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost/lportal?useUnicode=true&characterEncoding=UTF-8"
    username="test"
    password="test"
    max-pool="100"
    transactional="false"
/>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Sun JSAS with MySQL

Use the Administration Console to do the following:

Create a connection pool org.apache.commons.dbcp.BasicDataSource as the data source class. The pool properties are driverClassName=com.mysql.jdbc.Driver, url=jdbc:mysql://localhost/lportal?useUnicode=true&characterEncoding=UTF-8, username=test, and password=test.

Create a data source bound to jdbc/LiferayPool.

Tomcat with MySQL

For Tomcat 5.0.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/liferay.xml.

```
<Context...>
    ...
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
    />
    <ResourceParams name="jdbc/LiferayPool">
        <parameter>
            <name>driverClassName</name>
            <value>com.mysql.jdbc.Driver</value>
        </parameter>
        <parameter>
            <name>url</name>
<value>jdbc:mysql://localhost/lportal?useUnicode=true&amp;characterEncoding=UTF-8</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>test</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>test</value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>20</value>
        </parameter>
    </ResourceParams>
</Context>
```

For Tomcat 5.5.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/ROOT.xml.

```
<Context...>
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost/lportal?useUnicode=true&characterEncoding=UTF-8"
        username="test"
        password="test"
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
    />
</Context>
```

For Tomcat 5.0.x and 5.5.x, copy the JDBC driver to /common/lib. JDBC drivers can be found from the database vendor's web site.

WebLogic with MySQL

Use the Administration Console to do the following:

Create a connection pool using jdbc:mysql://localhost/lportal?useUnicode=true&characterEncoding=UTF-8 as the URL, com.mysql.jdbc.Driver as the driver, test as the user name, and test as the password.

Create a data source bound to jdbc/LiferayPool.

Create a Tx data source bound to jdbc/LiferayEJB.

WebSphere with MySQL

Use the Administrative Console: Resources -> JDBC Providers.

Create a new user-defined JDBC provider. Remove any references in the class path. Set org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS as the implementation class name.

Copy commons-dbcp.jar, commons-pool.jar, and your JDBC driver to /AppServer/lib/ext.

Create a data source bound to jdbc/LiferayPool. The custom properties are driver=com.mysql.jdbc.Driver, url=jdbc:mysql://localhost/lportal?useUnicode=true&characterEncoding=UTF-8, user=test, and password=test.

Oracle

Create a database in Oracle called lportal. The Oracle 9 and 10 dump scripts are available here [http://www.liferay.com/web/guest/downloads].

Note: Do not use the default Liferay JDBC driver! The Liferay Driver was required in previous versions to work around a bug in Oracle. This is no longer required.

You must use the Oracle 10g driver located in classes12.jar whether or not you are using Oracle 9 or Oracle 10. The location of classes12.jar is usually at /oracle/product/10.2.0/db_1/jdbc/lib/classes12.jar but may be different depending on the directory where you installed Oracle.

BES with Oracle

Create a data source bound to jdbc/LiferayPool by editing jndi-definitions and creating liferay.dar. DAR files are custom to BES. You can use the BES Deployment Descriptor Editor to create the DAR file.

```
<jndi-definitions>
    <visitransact-datasource>
        <jndi-name>jdbc/LiferayPool</jndi-name>
<driver-datasource-jndiname>datasources/JdsLiferayDriver</driver-datasource-jndiname>
        <property>
            <prop-name>connectionType</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>Direct</prop-value>
        </property>
        <property>
            <prop-name>dialect</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>oracle</prop-value>
        </property>
    </visitransact-datasource>
    <driver-datasource>
        <jndi-name>datasources/JdsLiferayDriver</jndi-name>
<datasource-class-name>org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS</datasource-c
        <log-writer>False</log-writer>
        <property>
            <prop-name>driver</prop-name>
            <prop-type>String</prop-type>
            <prop-value>com.liferay.jdbc.LiferayDriver</prop-value>
        </property>
        <property>
            <prop-name>url</prop-name>
            <prop-type>String</prop-type>
            <prop-value>jdbc:oracle:thin:@localhost:1521:orcl</prop-value>
        </property>
        <property>
            <prop-name>username</prop-name>
            <prop-type>String</prop-type>
            <prop-value>test</prop-value>
        </property>
        <property>
            <prop-name>password</prop-name>
            <prop-type>String</prop-type>
            <prop-value>test</prop-value>
        </property>
    </driver-datasource>
</jndi-definitions>
```

Copy the JDBC driver to a path that can be picked up by BES. JDBC drivers can be found from the database vendor's web site.

JBoss with Oracle

Create a data source bound to jdbc/LiferayPool by editing liferay-ds.xml.

```
<datasources>
   <local-tx-datasource>
     <jndi-name>jdbc/LiferayPool</jndi-name>
     <connection-url>
         jdbc:oracle:thin:@localhost:1521:orcl
     </connection-url>
     <driver-class>com.liferay.jdbc.LiferayDriver</driver-class>
     <user-name>test</user-name>
     <password>test</password>
     <min-pool-size>0</min-pool-size>
   </local-tx-datasource>
</datasources>
```

Copy the JDBC driver for to /server/default/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Jetty with Oracle

Create a data source bound to jdbc/LiferayPool by editing /etc/jetty.xml.

```
<Call name="addService">
    <Arg>
        <New class="org.mortbay.jetty.plus.JotmService">
            <Set name="Name">TransactionMgr</Set>
            <Call name="addDataSource">
                <Arg>jdbc/LiferayPool</Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.standard.StandardXADataSource">
                        <Set
name="DriverName">com.liferay.jdbc.LiferayDriver</Set>
                        <Set
name="Url">jdbc:oracle:thin:@localhost:1521:orcl</Set>
                        <Set name="User">test</Set>
                        <Set name="Password">test</Set>
                    </New>
                </Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.pool.StandardXAPoolDataSource">
                        <Arg type="Integer">4</Arg>
                        <Set name="MinSize">4</Set>
                        <Set name="MaxSize">15</Set>
                    </New>
                </Arg>
            </Call>
        </New>
    </Arg>
</Call>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

JOnAS with Oracle

Create a data source bound to jdbc/LiferayPool by editing /conf/jonas/liferay-ds.properties.

```
datasource.name=jdbc/LiferayPool
datasource.url=jdbc:oracle:thin:@localhost:1521:orcl
datasource.classname=com.liferay.jdbc.LiferayDriver
datasource.username=test
datasource.password=test
datasource.mapper=
```

Copy the JDBC driver for to /lib/ext. JDBC drivers can be found from the database vendor's web site.

JRun with Oracle

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:oracle:thin:@localhost:1521:orcl as the URL, com.liferay.jdbc.LiferayDriver as the driver, test as the user name, and test as the password.

OracleAS with Oracle

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="com.liferay.jdbc.LiferayDriver"
    url="jdbc:oracle:thin:@localhost:1521:orcl"
    username="test"
    password="test"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver to /j2ee/home/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Orion with Oracle

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="com.liferay.jdbc.LiferayDriver"
    url="jdbc:oracle:thin:@localhost:1521:orcl"
    username="test"
    password="test"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Pramati with Oracle

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:oracle:thin:@localhost:1521:orcl as the URL, com.liferay.jdbc.LiferayDriver as the driver, test as the user name, and test as the password.

Resin with Oracle

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /conf/resin.conf.

```
<database>
    <jndi-name>jdbc/LiferayPool</jndi-name>
    <driver type="com.liferay.jdbc.LiferayDriver">
        <url>jdbc:oracle:thin:@localhost:1521:orcl</url>
        <user>test</user>
        <password>test</password>
    </driver>
    <prepared-statement-cache-size>8</prepared-statement-cache-size>
    <max-connections>20</max-connections>
    <max-idle-time>30s</max-idle-time>
</database>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

RexIP with Oracle

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /server.xml.

```
<datasource
    name="LiferayPool"
    jndi-name="jdbc/LiferayPool"
    driver="com.liferay.jdbc.LiferayDriver"
    url="jdbc:oracle:thin:@localhost:1521:orcl"
    username="test"
    password="test"
    max-pool="100"
    transactional="false"
/>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Sun JSAS with Oracle

Use the Administration Console to do the following:

Create a connection pool org.apache.commons.dbcp.BasicDataSource as the data source class. The pool properties are driverClassName=com.liferay.jdbc.LiferayDriver, url=jdbc:oracle:thin:@localhost:1521:orcl, username=test, and password=test.

Create a data source bound to jdbc/LiferayPool.

Tomcat with Oracle

For Tomcat 5.0.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/liferay.xml.

```
<Context...>
    ...
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
    />
    <ResourceParams name="jdbc/LiferayPool">
        <parameter>
            <name>driverClassName</name>
            <value>com.liferay.jdbc.LiferayDriver</value>
        </parameter>
        <parameter>
            <name>url</name>
            <value>jdbc:oracle:thin:@localhost:1521:orcl</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>test</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>test</value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>20</value>
        </parameter>
    </ResourceParams>
</Context>
```

For Tomcat 5.5.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/ROOT.xml.

```
<Context...>
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="com.liferay.jdbc.LiferayDriver"
        url="jdbc:oracle:thin:@localhost:1521:orcl"
        username="test"
        password="test"
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
    />
</Context>
```

For Tomcat 5.0.x and 5.5.x, copy the JDBC driver to /common/lib. JDBC drivers can be found from the database vendor's web site.

WebLogic with Oracle

Use the Administration Console to do the following:

Create a connection pool using jdbc:oracle:thin:@localhost:1521:orcl as the URL, com.liferay.jdbc.LiferayDriver as the driver, test as the user name, and test as the password.

Create a data source bound to jdbc/LiferayPool.

Create a Tx data source bound to jdbc/LiferayEJB.

WebSphere with Oracle

Use the Administrative Console: Resources -> JDBC Providers.

Create a new user-defined JDBC provider. Remove any references in the class path. Set org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS as the implementation class name.

Copy commons-dbcp.jar, commons-pool.jar, and your JDBC driver to /AppServer/lib/ext.

Create a data source bound to jdbc/LiferayPool. The custom properties are driver=com.liferay.jdbc.LiferayDriver, url=jdbc:oracle:thin:@localhost:1521:orcl, user=test, and password=test.

PostgreSQL

Create a database in PostgreSQL called lportal. At your UNIX shell, type psql -d lportal -f portal-postgresql.sql.

BES with PostgreSQL

Create a data source bound to jdbc/LiferayPool by editing jndi-definitions and creating liferay.dar. DAR files are custom to BES. You can use the BES Deployment Descriptor Editor to create the DAR file.

```
<jndi-definitions>
    <visitransact-datasource>
        <jndi-name>jdbc/LiferayPool</jndi-name>
<driver-datasource-jndiname>datasources/JdsLiferayDriver</driver-datasource-jndiname>
        <property>
            <prop-name>connectionType</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>Direct</prop-value>
        </property>
        <property>
            <prop-name>dialect</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>postgresql</prop-value>
        </property>
    </visitransact-datasource>
    <driver-datasource>
        <jndi-name>datasources/JdsLiferayDriver</jndi-name>
<datasource-class-name>org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS</datasource-c
        <log-writer>False</log-writer>
        <property>
            <prop-name>driver</prop-name>
            <prop-type>String</prop-type>
            <prop-value>org.postgresql.Driver</prop-value>
        </property>
        <property>
            <prop-name>url</prop-name>
            <prop-type>String</prop-type>
            <prop-value>jdbc:postgresql://localhost/lportal</prop-value>
        </property>
        <property>
            <prop-name>username</prop-name>
            <prop-type>String</prop-type>
            <prop-value>test</prop-value>
        </property>
        <property>
            <prop-name>password</prop-name>
            <prop-type>String</prop-type>
            <prop-value>test</prop-value>
        </property>
    </driver-datasource>
</jndi-definitions>
```

Copy the JDBC driver to a path that can be picked up by BES. JDBC drivers can be found from the database vendor's web site.

JBoss with PostgreSQL

Create a data source bound to jdbc/LiferayPool by editing liferay-ds.xml.

```
<datasources>
   <local-tx-datasource>
     <jndi-name>jdbc/LiferayPool</jndi-name>
     <connection-url>
         jdbc:postgresql://localhost/lportal
     </connection-url>
     <driver-class>org.postgresql.Driver</driver-class>
     <user-name>test</user-name>
     <password>test</password>
     <min-pool-size>0</min-pool-size>
   </local-tx-datasource>
</datasources>
```

Copy the JDBC driver for to /server/default/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Jetty with PostgreSQL

Create a data source bound to jdbc/LiferayPool by editing /etc/jetty.xml.

```
<Call name="addService">
    <Arg>
        <New class="org.mortbay.jetty.plus.JotmService">
            <Set name="Name">TransactionMgr</Set>
            <Call name="addDataSource">
                <Arg>jdbc/LiferayPool</Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.standard.StandardXADataSource">
                        <Set name="DriverName">org.postgresql.Driver</Set>
                        <Set
name="Url">jdbc:postgresql://localhost/lportal</Set>
                        <Set name="User">test</Set>
                        <Set name="Password">test</Set>
                    </New>
                </Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.pool.StandardXAPoolDataSource">
                        <Arg type="Integer">4</Arg>
                        <Set name="MinSize">4</Set>
                        <Set name="MaxSize">15</Set>
                    </New>
                </Arg>
            </Call>
        </New>
    </Arg>
</Call>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

JOnAS with PostgreSQL

Create a data source bound to jdbc/LiferayPool by editing /conf/jonas/liferay-ds.properties.

```
datasource.name=jdbc/LiferayPool
datasource.url=jdbc:postgresql://localhost/lportal
datasource.classname=org.postgresql.Driver
datasource.username=test
datasource.password=test
datasource.mapper=
```

Copy the JDBC driver for to /lib/ext. JDBC drivers can be found from the database vendor's web site.

JRun with PostgreSQL

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:postgresql://localhost/lportal as the URL, org.postgresql.Driver as the driver, test as the user name, and test as the password.

OracleAS with PostgreSQL

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="org.postgresql.Driver"
    url="jdbc:postgresql://localhost/lportal"
    username="test"
    password="test"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver to /j2ee/home/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Orion with PostgreSQL

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="org.postgresql.Driver"
    url="jdbc:postgresql://localhost/lportal"
    username="test"
    password="test"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Pramati with PostgreSQL

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:postgresql://localhost/lportal as the URL, org.postgresql.Driver as the driver, test as the user name, and test as the password.

Resin with PostgreSQL

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /conf/resin.conf.

```
<database>
    <jndi-name>jdbc/LiferayPool</jndi-name>
    <driver type="org.postgresql.Driver">
        <url>jdbc:postgresql://localhost/lportal</url>
        <user>test</user>
        <password>test</password>
    </driver>
    <prepared-statement-cache-size>8</prepared-statement-cache-size>
    <max-connections>20</max-connections>
    <max-idle-time>30s</max-idle-time>
</database>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

RexIP with PostgreSQL

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /server.xml.

```
<datasource
    name="LiferayPool"
    jndi-name="jdbc/LiferayPool"
    driver="org.postgresql.Driver"
    url="jdbc:postgresql://localhost/lportal"
    username="test"
    password="test"
    max-pool="100"
    transactional="false"
/>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Sun JSAS with PostgreSQL

Use the Administration Console to do the following:

Create a connection pool org.apache.commons.dbcp.BasicDataSource as the data source class. The pool properties are driverClassName=org.postgresql.Driver, url=jdbc:postgresql://localhost/lportal, username=test, and password=test.

Create a data source bound to jdbc/LiferayPool.

Tomcat with PostgreSQL

For Tomcat 5.0.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/liferay.xml.

```
<Context...>
    ...
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
    />
    <ResourceParams name="jdbc/LiferayPool">
        <parameter>
            <name>driverClassName</name>
            <value>org.postgresql.Driver</value>
        </parameter>
        <parameter>
            <name>url</name>
            <value>jdbc:postgresql://localhost/lportal</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>test</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>test</value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>20</value>
        </parameter>
    </ResourceParams>
</Context>
```

For Tomcat 5.5.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/ROOT.xml.

```
<Context...>
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="org.postgresql.Driver"
        url="jdbc:postgresql://localhost/lportal"
        username="test"
        password="test"
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
    />
</Context>
```

For Tomcat 5.0.x and 5.5.x, copy the JDBC driver to /common/lib. JDBC drivers can be found from the database vendor's web site.

WebLogic with PostgreSQL

Use the Administration Console to do the following:

Create a connection pool using jdbc:postgresql://localhost/lportal as the URL, org.postgresql.Driver as the driver, test as the user name, and test as the password.

Create a data source bound to jdbc/LiferayPool.

Create a Tx data source bound to jdbc/LiferayEJB.

WebSphere with PostgreSQL

Use the Administrative Console: Resources -> JDBC Providers.

Create a new user-defined JDBC provider. Remove any references in the class path. Set org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS as the implementation class name.

Copy commons-dbcp.jar, commons-pool.jar, and your JDBC driver to /AppServer/lib/ext.

Create a data source bound to jdbc/LiferayPool. The custom properties are driver=org.postgresql.Driver, url=jdbc:postgresql://localhost/lportal, user=test, and password=test.

SAP

Create a database in SAP called lportal. Load the script named portal-sap.sql.

BES with SAP

Create a data source bound to jdbc/LiferayPool by editing jndi-definitions and creating liferay.dar. DAR files are custom to BES. You can use the BES Deployment Descriptor Editor to create the DAR file.

```
<jndi-definitions>
    <visitransact-datasource>
        <jndi-name>jdbc/LiferayPool</jndi-name>
<driver-datasource-jndiname>datasources/JdsLiferayDriver</driver-datasource-jndiname>
        <property>
            <prop-name>connectionType</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>Direct</prop-value>
        </property>
        <property>
            <prop-name>dialect</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>sapdb</prop-value>
        </property>
    </visitransact-datasource>
    <driver-datasource>
        <jndi-name>datasources/JdsLiferayDriver</jndi-name>
<datasource-class-name>org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS</datasource-c
        <log-writer>False</log-writer>
        <property>
            <prop-name>driver</prop-name>
            <prop-type>String</prop-type>
            <prop-value>com.sap.dbtech.jdbc.DriverSapDB</prop-value>
        </property>
        <property>
            <prop-name>url</prop-name>
            <prop-type>String</prop-type>
            <prop-value>jdbc:sapdb://localhost/lportal</prop-value>
        </property>
        <property>
            <prop-name>username</prop-name>
            <prop-type>String</prop-type>
            <prop-value>test</prop-value>
        </property>
        <property>
            <prop-name>password</prop-name>
            <prop-type>String</prop-type>
            <prop-value>test</prop-value>
        </property>
    </driver-datasource>
</jndi-definitions>
```

Copy the JDBC driver to a path that can be picked up by BES. JDBC drivers can be found from the database vendor's web site.

JBoss with SAP

Create a data source bound to jdbc/LiferayPool by editing liferay-ds.xml.

```
<datasources>
   <local-tx-datasource>
     <jndi-name>jdbc/LiferayPool</jndi-name>
     <connection-url>
         jdbc:sapdb://localhost/lportal
     </connection-url>
     <driver-class>com.sap.dbtech.jdbc.DriverSapDB</driver-class>
     <user-name>test</user-name>
     <password>test</password>
     <min-pool-size>0</min-pool-size>
   </local-tx-datasource>
</datasources>
```

Copy the JDBC driver for to /server/default/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Jetty with SAP

Create a data source bound to jdbc/LiferayPool by editing /etc/jetty.xml.

```
<Call name="addService">
    <Arg>
        <New class="org.mortbay.jetty.plus.JotmService">
            <Set name="Name">TransactionMgr</Set>
            <Call name="addDataSource">
                <Arg>jdbc/LiferayPool</Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.standard.StandardXADataSource">
                        <Set
name="DriverName">com.sap.dbtech.jdbc.DriverSapDB</Set>
                        <Set name="Url">jdbc:sapdb://localhost/lportal</Set>
                        <Set name="User">test</Set>
                        <Set name="Password">test</Set>
                    </New>
                </Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.pool.StandardXAPoolDataSource">
                        <Arg type="Integer">4</Arg>
                        <Set name="MinSize">4</Set>
                        <Set name="MaxSize">15</Set>
                    </New>
                </Arg>
            </Call>
        </New>
    </Arg>
</Call>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

JOnAS with SAP

Create a data source bound to jdbc/LiferayPool by editing /conf/jonas/liferay-ds.properties.

```
datasource.name=jdbc/LiferayPool
datasource.url=jdbc:sapdb://localhost/lportal
datasource.classname=com.sap.dbtech.jdbc.DriverSapDB
datasource.username=test
datasource.password=test
datasource.mapper=
```

Copy the JDBC driver for to /lib/ext. JDBC drivers can be found from the database vendor's web site.

JRun with SAP

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:sapdb://localhost/lportal as the URL, com.sap.dbtech.jdbc.DriverSapDB as the driver, test as the user name, and test as the password.

OracleAS with SAP

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="com.sap.dbtech.jdbc.DriverSapDB"
    url="jdbc:sapdb://localhost/lportal"
    username="test"
    password="test"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver to /j2ee/home/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Orion with SAP

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="com.sap.dbtech.jdbc.DriverSapDB"
    url="jdbc:sapdb://localhost/lportal"
    username="test"
    password="test"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Pramati with SAP

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:sapdb://localhost/lportal as the URL, com.sap.dbtech.jdbc.DriverSapDB as the driver, test as the user name, and test as the password.

Resin with SAP

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /conf/resin.conf.

```
<database>
    <jndi-name>jdbc/LiferayPool</jndi-name>
    <driver type="com.sap.dbtech.jdbc.DriverSapDB">
        <url>jdbc:sapdb://localhost/lportal</url>
        <user>test</user>
        <password>test</password>
    </driver>
    <prepared-statement-cache-size>8</prepared-statement-cache-size>
    <max-connections>20</max-connections>
    <max-idle-time>30s</max-idle-time>
</database>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

RexIP with SAP

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /server.xml.

```
<datasource
    name="LiferayPool"
    jndi-name="jdbc/LiferayPool"
    driver="com.sap.dbtech.jdbc.DriverSapDB"
    url="jdbc:sapdb://localhost/lportal"
    username="test"
    password="test"
    max-pool="100"
    transactional="false"
/>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Sun JSAS with SAP

Use the Administration Console to do the following:

Create a connection pool org.apache.commons.dbcp.BasicDataSource as the data source class. The pool properties are driverClassName=com.sap.dbtech.jdbc.DriverSapDB, url=jdbc:sapdb://localhost/lportal, username=test, and password=test.

Create a data source bound to jdbc/LiferayPool.

Tomcat with SAP

For Tomcat 5.0.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/liferay.xml.

```
<Context...>
    ...
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
    />
    <ResourceParams name="jdbc/LiferayPool">
        <parameter>
            <name>driverClassName</name>
            <value>com.sap.dbtech.jdbc.DriverSapDB</value>
        </parameter>
        <parameter>
            <name>url</name>
            <value>jdbc:sapdb://localhost/lportal</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>test</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>test</value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>20</value>
        </parameter>
    </ResourceParams>
</Context>
```

For Tomcat 5.5.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/ROOT.xml.

```
<Context...>
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="com.sap.dbtech.jdbc.DriverSapDB"
        url="jdbc:sapdb://localhost/lportal"
        username="test"
        password="test"
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
    />
</Context>
```

For Tomcat 5.0.x and 5.5.x, copy the JDBC driver to /common/lib. JDBC drivers can be found from the database vendor's web site.

WebLogic with SAP

Use the Administration Console to do the following:

Create a connection pool using jdbc:sapdb://localhost/lportal as the URL, com.sap.dbtech.jdbc.DriverSapDB as the driver, test as the user name, and test as the password.

Create a data source bound to jdbc/LiferayPool.

Create a Tx data source bound to jdbc/LiferayEJB.

WebSphere with SAP

Use the Administrative Console: Resources -> JDBC Providers.

Create a new user-defined JDBC provider. Remove any references in the class path. Set org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS as the implementation class name.

Copy commons-dbcp.jar, commons-pool.jar, and your JDBC driver to /AppServer/lib/ext.

Create a data source bound to jdbc/LiferayPool. The custom properties are driver=com.sap.dbtech.jdbc.DriverSapDB, url=jdbc:sapdb://localhost/lportal, user=test, and password=test.

SQL Server

Create a database in SQL Server called lportal. Use Enterprise Manager to load the script named portal-sql-server.sql.

BES with SQL Server

Create a data source bound to jdbc/LiferayPool by editing jndi-definitions and creating liferay.dar. DAR files are custom to BES. You can use the BES Deployment Descriptor Editor to create the DAR file.

```
<jndi-definitions>
    <visitransact-datasource>
        <jndi-name>jdbc/LiferayPool</jndi-name>
<driver-datasource-jndiname>datasources/JdsLiferayDriver</driver-datasource-jndiname>
        <property>
            <prop-name>connectionType</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>Direct</prop-value>
        </property>
        <property>
            <prop-name>dialect</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>sqlserver</prop-value>
        </property>
    </visitransact-datasource>
    <driver-datasource>
        <jndi-name>datasources/JdsLiferayDriver</jndi-name>
<datasource-class-name>org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS</datasource-c
        <log-writer>False</log-writer>
        <property>
            <prop-name>driver</prop-name>
            <prop-type>String</prop-type>
            <prop-value>net.sourceforge.jtds.jdbc.Driver</prop-value>
        </property>
        <property>
            <prop-name>url</prop-name>
            <prop-type>String</prop-type>
            <prop-value>jdbc:jtds:sqlserver://localhost/lportal</prop-value>
        </property>
        <property>
            <prop-name>username</prop-name>
            <prop-type>String</prop-type>
            <prop-value>test</prop-value>
        </property>
        <property>
            <prop-name>password</prop-name>
            <prop-type>String</prop-type>
            <prop-value>test</prop-value>
        </property>
    </driver-datasource>
</jndi-definitions>
```

Copy the JDBC driver to a path that can be picked up by BES. JDBC drivers can be found from the database vendor's web site.

JBoss with SQL Server

Create a data source bound to jdbc/LiferayPool by editing liferay-ds.xml.

```
<datasources>
   <local-tx-datasource>
     <jndi-name>jdbc/LiferayPool</jndi-name>
     <connection-url>
         jdbc:jtds:sqlserver://localhost/lportal
     </connection-url>
     <driver-class>net.sourceforge.jtds.jdbc.Driver</driver-class>
     <user-name>test</user-name>
     <password>test</password>
     <min-pool-size>0</min-pool-size>
   </local-tx-datasource>
</datasources>
```

Copy the JDBC driver for to /server/default/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Jetty with SQL Server

Create a data source bound to jdbc/LiferayPool by editing /etc/jetty.xml.

```
<Call name="addService">
    <Arg>
        <New class="org.mortbay.jetty.plus.JotmService">
            <Set name="Name">TransactionMgr</Set>
            <Call name="addDataSource">
                <Arg>jdbc/LiferayPool</Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.standard.StandardXADataSource">
                        <Set
name="DriverName">net.sourceforge.jtds.jdbc.Driver</Set>
                        <Set
name="Url">jdbc:jtds:sqlserver://localhost/lportal</Set>
                        <Set name="User">test</Set>
                        <Set name="Password">test</Set>
                    </New>
                </Arg>
                <Arg>
                    <New
class="org.enhydra.jdbc.pool.StandardXAPoolDataSource">
                        <Arg type="Integer">4</Arg>
                        <Set name="MinSize">4</Set>
                        <Set name="MaxSize">15</Set>
                    </New>
                </Arg>
            </Call>
        </New>
    </Arg>
</Call>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

JOnAS with SQL Server

Create a data source bound to jdbc/LiferayPool by editing /conf/jonas/liferay-ds.properties.

```
datasource.name=jdbc/LiferayPool
datasource.url=jdbc:jtds:sqlserver://localhost/lportal
datasource.classname=net.sourceforge.jtds.jdbc.Driver
datasource.username=test
datasource.password=test
datasource.mapper=
```

Copy the JDBC driver for to /lib/ext. JDBC drivers can be found from the database vendor's web site.

JRun with SQL Server

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:jtds:sqlserver://localhost/lportal as the URL, net.sourceforge.jtds.jdbc.Driver as the driver, test as the user name, and test as the password.

OracleAS with SQL Server

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="net.sourceforge.jtds.jdbc.Driver"
    url="jdbc:jtds:sqlserver://localhost/lportal"
    username="test"
    password="test"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver to /j2ee/home/lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't already another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Orion with SQL Server

Create a data source bound to jdbc/LiferayPool, create a Tx data source bound to jdbc/LiferayEJB, and set the proper JDBC properties by editing /config/data-sources.xml.

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="Liferay"
    location="jdbc/LiferayCore"
    pooled-location="jdbc/LiferayPool"
    xa-location="jdbc/xa/LiferayXA"
    ejb-location="jdbc/LiferayEJB"
    connection-driver="net.sourceforge.jtds.jdbc.Driver"
    url="jdbc:jtds:sqlserver://localhost/lportal"
    username="test"
    password="test"
    inactivity-timeout="30"
    schema="database-schemas/"
/>
```

Copy the JDBC driver for to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Pramati with SQL Server

Use the Management Console to do the following:

Create a data source bound to jdbc/LiferayPool using jdbc:jtds:sqlserver://localhost/lportal as the URL, net.sourceforge.jtds.jdbc.Driver as the driver, test as the user name, and test as the password.

Resin with SQL Server

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /conf/resin.conf.

```
<database>
    <jndi-name>jdbc/LiferayPool</jndi-name>
    <driver type="net.sourceforge.jtds.jdbc.Driver">
        <url>jdbc:jtds:sqlserver://localhost/lportal</url>
        <user>test</user>
        <password>test</password>
    </driver>
    <prepared-statement-cache-size>8</prepared-statement-cache-size>
    <max-connections>20</max-connections>
    <max-idle-time>30s</max-idle-time>
</database>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

RexIP with SQL Server

Create a data source bound to jdbc/LiferayPool and set the proper JDBC properties by editing /server.xml.

```
<datasource
    name="LiferayPool"
    jndi-name="jdbc/LiferayPool"
    driver="net.sourceforge.jtds.jdbc.Driver"
    url="jdbc:jtds:sqlserver://localhost/lportal"
    username="test"
    password="test"
    max-pool="100"
    transactional="false"
/>
```

Copy the JDBC driver to /lib. JDBC drivers can be found from the database vendor's web site.

Make sure there isn't another data source bound to jdbc/LiferayPool. If there is a duplicate service, undeploy it by removing its xml configuration.

Sun JSAS with SQL Server

Use the Administration Console to do the following:

Create a connection pool org.apache.commons.dbcp.BasicDataSource as the data source class. The pool properties are driverClassName=net.sourceforge.jtds.jdbc.Driver, url=jdbc:jtds:sqlserver://localhost/lportal, username=test, and password=test.

Create a data source bound to jdbc/LiferayPool.

Tomcat with SQL Server

For Tomcat 5.0.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/liferay.xml.

```
<Context...>
    ...
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
    />
    <ResourceParams name="jdbc/LiferayPool">
        <parameter>
            <name>driverClassName</name>
            <value>net.sourceforge.jtds.jdbc.Driver</value>
        </parameter>
        <parameter>
            <name>url</name>
            <value>jdbc:jtds:sqlserver://localhost/lportal</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>test</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>test</value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>20</value>
        </parameter>
    </ResourceParams>
</Context>
```

For Tomcat 5.5.x, create a data source bound to jdbc/LiferayPool by editing /conf/Catalina/localhost/ROOT.xml.

```
<Context...>
    <Resource
        name="jdbc/LiferayPool"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="net.sourceforge.jtds.jdbc.Driver"
        url="jdbc:jtds:sqlserver://localhost/lportal"
        username="test"
        password="test"
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
    />
</Context>
```

For Tomcat 5.0.x and 5.5.x, copy the JDBC driver to /common/lib. JDBC drivers can be found from the database vendor's web site.

WebLogic with SQL Server

Use the Administration Console to do the following:

Create a connection pool using jdbc:jtds:sqlserver://localhost/lportal as the URL, net.sourceforge.jtds.jdbc.Driver as the driver, test as the user name, and test as the password.

Create a data source bound to jdbc/LiferayPool.

Create a Tx data source bound to jdbc/LiferayEJB.

WebSphere with SQL Server

Use the Administrative Console: Resources -> JDBC Providers.

Create a new user-defined JDBC provider. Remove any references in the class path. Set org.apache.commons.dbcp.cpdsadapter.DriverAdapterCPDS as the implementation class name.

Copy commons-dbcp.jar, commons-pool.jar, and your JDBC driver to /AppServer/lib/ext.

Create a data source bound to jdbc/LiferayPool. The custom properties are driver=net.sourceforge.jtds.jdbc.Driver, url=jdbc:jtds:sqlserver://localhost/lportal, user=test, and password=test.

Upgrade to new versions of Liferay

Liferay provides default upgrade scripts to make it easier to upgrade to new versions of Liferay.

Run the upgrade scripts

1. Go to the ext/sql directory and run ant build-db. This wil automataically upgrade your database.

Note: If you have modified Liferay's default tables, you may need to customize the upgrade script according to your implementation. Upgrade scripts are located in the ext/sql/update-* folders.

Regenerate classes

If you auto-generated classes using service.xml, you will need to paste in the new version of the DTD and generate your classes again.

Upgrading Themes from 3.6.1 to 4.0.0

See Chapter 4: Theme Building for instructions on upgrading your theme from 3.6.1 to 4.0.0.

Mail Servers

Liferay Portal Enterprise can integrate with Washington IMAP+Sendmail, Cyrus IMAP+Postfix, and Dovecot+Postfix. Support for integration with Microsoft Exchange and other IMAP servers are planned and will be implemented in the near future.

The portal synchronizes with the mail server's user authentication by adding a mail server account when a portal account is added, deleting a mail server account when a portal account is deleted, and updating a mail server account when a portal account is updated. To do this, the portal must have privileges to modify and to update the mail server's user database.

The portal must also keep track of how email addresses map to certain accounts. For example, in the default installation, the portal maps the user id `liferay.com.1` to the email address `test@liferay.com`.

One possible deployment scenario is to run the mail EJBs on the mail server and run the portal EJBs on the web server. In this case, the mail server and web server are two different machines. The portal EJBs will give abstract commands (add/delete/update user) to the remote mail EJBs to carry out. The mail EJBs then run the appropriate system commands for the specific mail server and operating system.

Another possible deployment scenario is to have the mail EJBs and portal EJBs run on the same machine. This can all be configured by editing `portal.properties`.

Users access their email through an IMAP server. Access is limited to IMAP so that the portal does not have to be programmed to know where to persist the mail.

Washington IMAP+Sendmail

1. Install Sendmail and Expect on your mail server. Expect allows you to add, delete, or update users in one command. An example script for Red Hat is included in /mail-ejb/scripts/redhat.

2. Configure /portal-ejb/classes/portal.properties for your mail server.

3. The following instructions assume:

- The server envronment is linux
- The server name is called PORTAL_HOST
- You are logged in as root
- The distribution is Liferay Portal Professional 3.2.0 (Bundled with Tomcat)
- Tomcat is installed at /usr/local/tomcat
- Tomcat is running under the user named tomcat, group name tomcat
- You are using sendmail for email
- Portal sendmail users are created under the path /home/liferay/users
- sendmail is running on PORTAL_HOST

```
# Install expect command
apt-get install expect


# Give tomcat user a password
passwd tomcat




#give tomcat user a login shell
vi /etc/passwd
tomcat:x:500:500::/usr/local/tomcat:/bin/bash




# Use sudo to allow tomcat to add users
visudo

Defaults logfile=/var/log/sudolog
Defaults:tomcat    timestamp_timeout=-1, passwd_tries=1
tomcat  ALL=/usr/sbin/adduser, /usr/sbin/userdel, /usr/bin/passwd




# Enable UW-imap
vi /etc/xinet.d/imap

# default: off
# description: The IMAP service allows remote users to access their mail using
\
#              an IMAP client such as Mutt, Pine, fetchmail, or Netscape \
#              Communicator.
service imap
{
       socket_type             = stream
       wait                    = no
       user                    = root
```

```
        server                  = /usr/sbin/imapd
        log_on_success  += HOST DURATION
        log_on_failure  += HOST
        disable                 = no
}


# Restart the xinetd deamon
/etc/rc.d/init.d/xinetd restart




# Add Tomcat mail/MailSession settings
vi /usr/local/tomcat/conf/Catalina/localhost/liferay.xml
                <parameter>
                        <name>mail.smtp.host</name>
                        <value>localhost</value>
                </parameter>
                <parameter>
                        <name>mail.imap.host</name>
                        <value>localhost</value>
                </parameter>
                <parameter>
                        <name>mail.store.protocol</name>
                        <value>imap</value>
                </parameter>
                <parameter>
                        <name>mail.transport.protocol</name>
                        <value>smtp</value>
                </parameter>
                <parameter>
                        <name>mail.pop3.host</name>
                        <value>localhost</value>
                </parameter>



# Make the email mapping table writable by tomcat
chmod 664    /etc/mail/virtusertable
chmod 664    /etc/mail/virtusertable.db
chgrp tomcat /etc/mail/virtusertable



# Create lucent paths
mkdir /usr/local/tomcat/liferay/lucene


# Create sendmail users path
mkdir /home/liferay
mkdir /home/liferay/users

chown -R tomcat /home/liferay
chgrp -R tomcat /home/liferay
chmod -R 660    /home/liferay





# Create custom portal properties
# see http://www.liferay.com/static/content/portal.properties.html
```

```
vi /usr/local/tomcat/common/classes/portal-ext.properties

mail.hook.impl=com.liferay.mail.util.SendmailHook
mail.mx.update=true
mail.hook.sendmail.add.user=/usr/local/tomcat/bin/autouseradd %1%
mail.hook.sendmail.change.password=/usr/local/tomcat/bin/autopasswd %1% %2%
mail.hook.sendmail.delete.user=/usr/local/tomcat/bin/autouserdel %1%
mail.hook.sendmail.home=/home/liferay/users
mail.hook.sendmail.virtusertable=/etc/mail/virtusertable
mail.box.style=mail/
mail.username.replace=true
passwords.allow.dictionary.word=false
mail.junk-mail.warning.size=512000
mail.trash.warning.size=512000
mail.attachments.max.size=3072000
mail.audit.trail=root@PORTAL_HOST
lucene.dir /usr/local/tomcat/liferay/lucene/
```

```
# Create change password command
vi /usr/local/tomcat/bin/autopasswd

#!/usr/bin/expect -f
set password [lindex $argv 1]

spawn sudo /usr/bin/passwd [lindex $argv 0]
expect -i  $spawn_id "password:"
sleep .5
send "$password\r"
expect "password:"W
sleep .5
send "$password\r"
expect eof
```

```
# Create user add command
vi /usr/local/tomcat/bin/autouseradd

#!/usr/bin/expect -f
# 1st argument is the user id to add.
# Note: setting mail.username.replace=true in
/common/classes/portal-ext.properties
#   will replace the .'s with _'s in userid, which is required for linux

set userid [lindex $argv 0]
spawn sudo /usr/sbin/adduser $userid -s /bin/false
expect eof
```

```
# Create user remove command
vi /usr/local/tomcat/bin/autouserdel

#!/usr/bin/expect -f
# 1st argument is the user id to remove
# Note: setting mail.username.replace=true in
/common/classes/portal-ext.properties
#   will replace the .'s with _'s in userid, which is required for linux

set userid [lindex $argv 0]
spawn sudo /usr/sbin/userdel -r $userid
expect eof
```

```
# Set command file permissions
chmod 700    /usr/local/tomcat/bin/autopasswd
chown tomcat /usr/local/tomcat/bin/autopasswd
chgrp tomcat /usr/local/tomcat/bin/autopasswd
chmod 700    /usr/local/tomcat/bin/autouseradd
chown tomcat /usr/local/tomcat/bin/autouseradd
chgrp tomcat /usr/local/tomcat/bin/autouseradd
chmod 700    /usr/local/tomcat/bin/autouserdel
chown tomcat /usr/local/tomcat/bin/autouserdel
chgrp tomcat /usr/local/tomcat/bin/autouserdel
```

```
# Activate tomcat sudo, so it never prompts again
su tomcat
/usr/local/tomcat/bin/autouseradd badusername
/usr/local/tomcat/bin/autopasswd  badusername asst1453
/usr/local/tomcat/bin/autouserdel badusername
exit
```

Cyrus IMAP+Postfix

1. Install Fedora Core 4 [http://fedora.redhat.com/].

   For a minimal installation, choose to install a custom server. Deselect all packages groups. Select the package groups: Text-based Internet, Mail Server, DNS Name Server, FTP Server, MySQL Database, Network Servers, Development Tools, Legacy Software Development, Administration Tools, and System Tools.

   Make sure the following RPMs are also selected. The packages cyrus-imapd and cyrus-imapd-utils are only available in Fedora Core 2 and Fedora Core 4. They were not part of Fedora Core 1 and needed to be compiled manually. In Fedora Core 4, they were moved to Extras and you will need to use yum to install these packages.

   Mail Server: +cyrus-imapd, +cyrus-imapd-utils

   MySQL Database: +mysql-server

   Development Tools: +expect

2. Update Fedora. This may take a while even if you have a fast connection.

rpm --import /usr/share/rhn/RPM-GPG-KEY-fedora

yum list

yum upgrade

3. Turn off Sendmail.

   chkconfig --level 3 sendmail off

   /etc/rc.d/init.d/sendmail stop

4. Edit /etc/sysconfig/saslauthd.

   Replace `MECH=shadow` with `MECH=pam`.

   Turn on Cyrus SASL.

   chkconfig --level 3 saslauthd on

   /etc/rc.d/init.d/saslauthd start

5. Download Cyrus IMAP. If you are using Fedora Core 2 or later, you can use the RPMs from Fedora: cyrus-imapd and cyrus-imapd-utils. If you are using Fedora Core 1 or an earlier version of Red Hat, download cyrus-imapd-2.1.16-6.src.rpm [http://www.invoca.ch/pub/packages/cyrus-imapd/2.1/cyrus-imapd-2.1.16-6.src.rpm] and build the RPM for your environment from the source distribution.

   Build Cyrus IMAP.

   rpmbuild --rebuild cyrus-imapd-2.1.16-6.src.rpm

   Install Cyrus IMAP.

   rpm -i cyrus-imapd-2.1.16-6.i386.rpm

   rpm -i cyrus-imapd-utils-2.1.16-6.i386.rpm

   Turn on Cyrus IMAP.

   chkconfig --level 3 cyrus-imapd on

   /etc/rc.d/init.d/cyrus-imapd start

6. Download the source distribution of Postfix [http://ftp.wl0.org/official/2.1/SRPMS/postfix-2.1.6-1.src.rpm].

   Install Postfix with support for MySQL and Cyrus SASL.

   rpm -ivh postfix-2.1.6-1.src.rpm

   cd /usr/src/redhat/SOURCES

   bash

   export POSTFIX_MYSQL_REDHAT=1

   export POSTFIX_SASL=2

export POSTFIX_TLS=1

sh make-postfix.spec

exit

cd /usr/src/redhat/SPECS

rpmbuild -ba postfix.spec

cd /usr/src/redhat/RPMS/i386

rpm -i --force postfix-2.1.6-1.mysql.sasl2.tls.fc4.i386.rpm

7. Download the source distribution of PAM MySQL [http://www.invoca.ch/pub/packages/cyrus-imapd/contrib/pam_mysql-0.5-0.src.rpm].

   Install PAM MySQL.

   rpm -ivh pam_mysql-0.5-0.src.rpm

   cd /usr/src/redhat/SPECS

   rpmbuild -ba pam_mysql.spec

   cd /usr/src/redhat/RPMS/i386

   rpm -i pam_mysql-0.5-0.i386.rpm

8. Copy /mail-ejb/scripts/fedora/cyrus/mysql_virtual.cf to /etc/postfix/mysql_virtual.cf. Modify mysql_virtual.cf to point to your MySQL database.

   Edit /etc/postfix/virtual. Add the line **`yourdomain.com anything`** for each virtual domain that Postfix will manage. A correspending entry is needed in the MySQL database so that email to postmaster@yourdomain.com can be delivered to a Cyrus IMAP account.

   Transform /etc/postfix/virtual to a format Postfix can read.

   postmap /etc/postfix/virtual

   Edit /etc/postfix/master.cf. Replace the two instances of `/cyrus/bin/deliver` with **`/usr/lib/cyrus-imapd/deliver`**. Add these two lines:

```
procmail  unix  -       n       n       -       -       pipe
  flags=R user=cyrus argv=/usr/bin/procmail -t -m USER=${user}
EXTENSION=${extension} /home/cyrus/procmailrc
```

   Edit /etc/postfix/main.cf. Add these lines:

```
#
# Custom Settings
#

mynetworks = 127.0.0.0/8, 192.168.0.0/16, 128.135.12.7/32

mailbox_command = /usr/bin/procmail -t -a "$EXTENSION"
mailbox_transport = procmail
```

```
virtual_maps = hash:/etc/postfix/virtual,
mysql:/etc/postfix/mysql_virtual.cf

smtpd_recipient_restrictions = permit_mynetworks, check_client_access
hash:/etc/postfix/pop-before-smtp, check_relay_domains
```

Set `mynetworks` to the IPs that are allowed to connect to Postfix. Turn on Postfix.

chkconfig --level 3 postfix on

/etc/rc.d/init.d/postfix start

9. Copy /mail-ejb/scripts/fedora/cyrus/procmailrc to /home/cyrus/procmailrc. Make sure the cyrus user can access the script.

   chown cyrus:mail /home/cyrus

   chown cyrus:mail /home/cyrus/procmailrc.

   Copy /mail-ejb/scripts/fedora/cyrus/cyrus_adduser to /usr/bin/cyrus_adduser. Edit cyrus_adduser and replace localhost with the mail server's host name. Make sure the script can be executed.

   chmod u+x /usr/bin/cyrus_adduser.

   Copy /mail-ejb/scripts/fedora/cyrus/cyrus_userdel to /usr/bin/cyrus_userdel. Edit cyrus_userdel and replace localhost with the mail server's host name. Make sure the script can be executed.

   chmod u+x /usr/bin/cyrus_userdel.

Note

If you copy cyrus_adduser and cyrus_userdel from a Windows environment to a Linux environment, you need to run dos2unix cyrus_adduser to convert the file so that Linux can read the file correctly.

10. Edit /etc/pam.d/pop so that POP authentication is checked via MySQL. Remove the current lines and add these lines:

```
#%PAM-1.0
auth sufficient pam_mysql.so user=dbuser passwd=dbpassword host=127.0.0.1
db=cyrus
table=CyrusUser usercolumn=userId passwdcolumn=password_ crypt=0

account required pam_mysql.so user=dbuser passwd=dbpassword host=127.0.0.1
db=cyrus
table=CyrusUser usercolumn=userId passwdcolumn=password_ crypt=0
```

Edit /etc/pam.d/imap so that IMAP authentication is checked via MySQL. Remove the current lines and add these lines:

```
#%PAM-1.0
auth sufficient pam_mysql.so user=dbuser passwd=dbpassword host=127.0.0.1
db=cyrus
table=CyrusUser usercolumn=userId passwdcolumn=password_ crypt=0

account required pam_mysql.so user=dbuser passwd=dbpassword host=127.0.0.1
db=cyrus
```

```
table=CyrusUser usercolumn=userId passwdcolumn=password_ crypt=0
```

11. Turn on MySQL.

    chkconfig --level 3 mysqld on

    /etc/rc.d/init.d/mysqld start

    Configure MySQL so that it can be accessed by the username dbuser and password dbpassword.

    use mysql;

    insert into user values ('127.0.0.1', "dbuser", password("dbpassword"), "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y");

    Create the database and tables that will be used to authenticate IMAP users.

    create database cyrus;

    use cyrus;

    create table CyrusUser (userId varchar(75) not null primary key, password_ varchar(75) not null);

    create table CyrusVirtual (emailAddress varchar(75) not null primary key, userId varchar(75) not null);

    The Expect scripts `cyrus_adduser` and `cyrus_userdel` that are used to add and delete Cyrus IMAP users require a default cyrus user to authenticate with.

    insert into CyrusUser (userId, password_) values ('cyrus', 'cyrus_password');

    Every virtual domain requires a postmaster@yourdomain.com entry so that email to postmaster@yourdomain.com can be delivered to a Cyrus IMAP account.

    insert into CyrusVirtual (emailAddress, userId) values ('postmaster@yourdomain.com', 'your_domain_1');

    Create a default account for your_domain_1.

    insert into CyrusUser (userId, password_) values ('your_domain_1', 'your_password');

    insert into CyrusVirtual (emailAddress, userId) values ('joe.blogs@yourdomain.com', 'your_domain_1');

    quit;

    cyrus_adduser cyrus_password your_domain_1

12. Turn on SpamAssassin.

    chkconfig --level 3 spamassassin on

    /etc/rc.d/init.d/spamassassin start

13. Download ClamAV [http://crash.fce.vutbr.cz/crash-hat/3/clamav/clamav-0.86.1-1.i386.rpm].

    Install ClamAV.

rpm -i clamav-0.86.1-1.i386.rpm

Turn on ClamAV.

chkconfig --level 3 clamd on

/etc/rc.d/init.d/clamd start

Download ClamAssassin [http://drivel.com/clamassassin/clamassassin-1.2.2.tar.gz].

Install ClamAssassin.

gunzip clamassassin-1.2.2.tar.gz

tar xvf clamassassin-1.2.2.tar

cd clamassassin-1.2.2

./configure

cp clamassassin /usr/local/bin

Edit /usr/local/bin/clamassassin.

Set `SUBJECTHEAD` to "**`[VIRUS]`** ".

14. Copy /mail-ejb/scripts/fedora/cyrus/procmail_vacation to /usr/local/bin/procmail_vacation. Make sure the script can be executed.

    chmod u+x /usr/local/bin/procmail_vacation.

    Download SendEmail [http://caspian.dotconf.net/menu/Software/SendEmail/sendEmail-v1.52.tar.gz].

    Install SendEmail.

    gunzip sendEmail-v1.52.tar.gz

    tar xvf sendEmail-v1.52.tar

    cd sendEmail-v1.52

    chmod u+x sendEmail

    chown cyrus:mail sendEmail

    cp sendEmail /usr/local/bin

15. Download the source distribution of Pop-before-smtp [http://sourceforge.net/projects/popbsmtp].

    Pop-before-smtp requires perl-TimeDate and perl-Net-Netmask.

    Install perl-TimeDate from the distributed RPM.

    Install perl-Net-Netmask.

    perl -MCPAN -e 'install Net::Netmask'

    Install Pop-before-smtp.

gunzip pop-before-smtp-1.38.tar.gz

tar xvf pop-before-smtp-1.38.tar

cd pop-before-smtp-1.38

chown root:root *

cp pop-before-smtp.init /etc/rc.d/init.d/pop-before-smtp

cp pop-before-smtp /usr/sbin/

cp pop-before-smtp-conf.pl /etc

Edit /etc/pop-before-smtp-conf.pl by uncommenting and modifying certain sections so it matches the following information.

```
$dbfile = '/etc/postfix/pop-before-smtp';

$grace = 120*60;

# Set the log file we will watch for pop3d/imapd records.
$file_tail{'name'} = '/var/log/maillog';

# For Cyrus (including a tweak for IP addrs that don't resolve):
$pat = '^(... .. ..:..:..) \S+ (?:pop3d|imapd)\[\d+\]: ' .
```

Turn on Pop-before-smtp.

chkconfig --level 3 pop-before-smtp on

/etc/rc.d/init.d/pop-before-smtp start

16. Restart your mail server.

    shutdown -r now

Dovecot+Postfix

First build a generic Liferay email hook, ShellHook.java, that shells out all of the email methods. You install it by adding these lines to portal-ext.properties: **`mail.hook.impl=com.liferay.mail.util.ShellHook`**

`mail.hook.shell.script=/usr/sbin/mailadmin.ksh`

`mail.box.style=INBOX`

We next built a generic Korn Shell Script, mailadmin.ksh, that implements each method for Dovecot, or any other email system you want. It supports an interactive command line interface for testing:

`mailadmin.ksh --help`

`mailadmin.ksh`

`mailadmin.ksh addForward [userId] [emailAddresses]`

`mailadmin.ksh addUser [userId] [password] [firstName] [middleName] [lastName]`

[emailAddress]

mailadmin.ksh addVacationMessage [userId] [emailAddress] [vacationMessage]

mailadmin.ksh deleteEmailAddress [userId]

mailadmin.ksh deleteUser [userId]

mailadmin.ksh updateBlocked [userId] [blockedEmailAddress]

mailadmin.ksh updateEmailAddress [userId] [emailAddress]

mailadmin.ksh updatePassword [userId] [password]

All of the code is in SVN. mailadmin is at: mail-ejb/scripts/fedora/ksh/mailadmin.ksh Here are the step-by-step installation instructions:

```
# Edit SASL-auth authentication to use MySQL with the Postfix setup

vi /etc/pam.d/smtp
#%PAM-1.0
auth sufficient pam_mysql.so user=DBUSR passwd=DBPASSWD host=127.0.0.1 db=mail
table=postfix_users usercolumn=email passwdcolumn=clear crypt=0
account required pam_mysql.so user=DBUSR passwd=DBPASSWD host=127.0.0.1
db=mail table=postfix_users usercolumn=email passwdcolumn=clear crypt=0




# CONFIGURE VMAIL USER AND EMAIL PATHS

groupadd -g 510 vmail
useradd  -u 510 -g vmail vmail
mkdir -p /var/vmail/EMAILDOMAIN
chown -R vmail:vmail /var/vmail
chmod -R 770         /var/vmail

# Add vmail user to tomcat group and tomcat user to vmail group
# Note the vmail uid, 510, is inserted into the postfix_users table below
vi /etc/group
tomcat:x:500:vmail
vmail:x:510:tomcat




# CONFIGURE MYSQL

#  Add DBUSR to MySql database for managing email tables

mysql -u root -p
use mysql;
insert into user values ('127.0.0.1', "DBUSR", old_password("DBPASSWD"), "Y",
"Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y");
commit;
quit


# Login as email user and build email database, "mail", and postfix tables
```

```
mysql -u DBUSR -p

create database mail;
use mail;

CREATE TABLE postfix_alias (
 id int(11) unsigned NOT NULL auto_increment,
 alias varchar(128) NOT NULL default '',
 destination varchar(128) NOT NULL default '',
 PRIMARY KEY (id)
) TYPE=MyISAM;
CREATE TABLE postfix_relocated (
 id int(11) unsigned NOT NULL auto_increment,
 email varchar(128) NOT NULL default '',
 destination varchar(128) NOT NULL default '',
 PRIMARY KEY (id)
) TYPE=MyISAM;

CREATE TABLE postfix_transport (
 id int(11) unsigned NOT NULL auto_increment,
 domain varchar(128) NOT NULL default '',
 destination varchar(128) NOT NULL default '',
 PRIMARY KEY (id),
 UNIQUE KEY domain (domain)
) TYPE=MyISAM;

CREATE TABLE postfix_users (
 id int(11) unsigned NOT NULL auto_increment,
 email varchar(128) NOT NULL default '',
 clear varchar(128) NOT NULL default '',
 crypt varchar(128) NOT NULL default '',
 name tinytext NOT NULL,
 uid int(11) unsigned NOT NULL default '1004',
 gid int(11) unsigned NOT NULL default '1004',
 homedir tinytext NOT NULL,
 maildir tinytext NOT NULL,
 quota tinytext NOT NULL,
 access enum('Y','N') NOT NULL default 'Y',
 postfix enum('Y','N') NOT NULL default 'Y',
 PRIMARY KEY (id),
 UNIQUE KEY email (email)
) TYPE=MyISAM;

CREATE TABLE postfix_virtual (
 id int(11) unsigned NOT NULL auto_increment,
 email varchar(128) NOT NULL default '',
 destination varchar(128) NOT NULL default '',
 PRIMARY KEY (id)
) TYPE=MyISAM;

CREATE TABLE postfix_access (
 id int(10) unsigned NOT NULL auto_increment,
 source varchar(128) NOT NULL default '',
 access varchar(128) NOT NULL default '',
 type enum('recipient','sender','client') NOT NULL default 'recipient',
 PRIMARY KEY (id)
) TYPE=MyISAM

commit;

#  Add an email domain
INSERT INTO `postfix_transport` VALUES
(3,'EMAILDOMAIN','virtual:');
```

```
#  Add an email user (automated by Liferay using ShellHook, and mailadmin.ksh)
#  vmail uid is 510
INSERT INTO `postfix_users` VALUES
(17,'LIFERAYUSR@EMAILDOMAIN','LIFERAYPWD','','',510,510,'/var/vmail','EMAILDOMAIN/LIFERAYU
#  Add an email forward
INSERT INTO `postfix_virtual` VALUES
(27,'LIFERAYLOGIN','LIFERAYUSR@EMAILDOMAIN');

commit;
quit
```

```
# CONFIGURE POSTFIX

cd /etc/postfix/
rm -rf ssl/
rm -rf sasl/


vi /etc/postfix/mysql-aliases.cf
user = DBUSR
password = DBPASSWD
dbname = mail
table = postfix_alias
select_field = destination
where_field = alias
hosts = 127.0.0.1

vi /etc/postfix/mysql-client.cf
user = DBUSR
password = DBPASSWD
dbname = mail
table = postfix_access
select_field = access
where_field = source
additional_conditions = and type = 'client'
hosts = 127.0.0.1

vi /etc/postfix/mysql-recipient.cf
user = DBUSR
password = DBPASSWD
dbname = mail
table = postfix_access
select_field = access
where_field = source
additional_conditions = and type = 'recipient'
hosts = 127.0.0.1

vi /etc/postfix/mysql-relocated.cf
user = DBUSR
password = DBPASSWD
dbname = mail
table = postfix_relocated
select_field = destination
where_field = email
hosts = 127.0.0.1
```

```
vi /etc/postfix/mysql-sender.cf
user = DBUSR
password = DBPASSWD
dbname = mail
table = postfix_access
select_field = access
where_field = source
additional_conditions = and type = 'sender'
hosts = 127.0.0.1

vi /etc/postfix/mysql-transport.cf
user = DBUSR
password = DBPASSWD
dbname = mail
table = postfix_transport
select_field = destination
where_field = domain
hosts = 127.0.0.1

vi /etc/postfix/mysql-virtual-gid.cf
user = DBUSR
password = DBPASSWD
dbname = mail
table = postfix_users
select_field = gid
where_field = email
additional_conditions = and postfix = 'y'
hosts = 127.0.0.1

vi /etc/postfix/mysql-virtual-maps.cf
user = DBUSR
password = DBPASSWD
dbname = mail
table = postfix_users
select_field = maildir
where_field = email
additional_conditions = and postfix = 'y'
hosts = 127.0.0.1

vi /etc/postfix/mysql-virtual-uid.cf
user = DBUSR
password = DBPASSWD
dbname = mail
table = postfix_users
select_field = uid
where_field = email
additional_conditions = and postfix = 'y'
hosts = 127.0.0.1

vi /etc/postfix/mysql-virtual.cf
user = DBUSR
password = DBPASSWD
dbname = mail
table = postfix_virtual
select_field = destination
where_field = email
hosts = 127.0.0.1

chmod 640          /etc/postfix/mysql-*
chown root:postfix /etc/postfix/mysql-*


vi /etc/postfix/main.cf
```

```
# see /usr/share/postfix/main.cf.dist for a commented, fuller version of this
file.
# Do not change these directory settings - they are critical to Postfix
operation.
command_directory = /usr/sbin
daemon_directory = /usr/libexec/postfix
program_directory = /usr/libexec/postfix
smtpd_banner = $myhostname ESMTP $mail_name
setgid_group = postdrop
biff = no
append_dot_mydomain = no
myhostname = EMAILDOMAIN
myorigin = $myhostname
mydestination = EMAILDOMAIN, $transport_maps
relayhost =
mynetworks = 127.0.0.0/8
mailbox_command =
mailbox_size_limit = 0
recipient_delimiter = +
smtpd_sasl_auth_enable = yes
smtpd_recipient_restrictions = permit_sasl_authenticated, permit_mynetworks,
reject_unauth_destination
smtpd_sasl_security_options = noanonymous
smtpd_sasl_local_domain = $myhostname
broken_sasl_auth_clients = yes
smtpd_recipient_restrictions = permit_mynetworks, permit_sasl_authenticated,
check_recipient_access mysql:/etc/postfix/mysql-recipient.cf,
reject_unauth_destination, permit
smtpd_sender_restrictions = check_sender_access
mysql:/etc/postfix/mysql-sender.cf
smtpd_client_restrictions = check_client_access
mysql:/etc/postfix/mysql-client.cf
alias_maps = mysql:/etc/postfix/mysql-aliases.cf
relocated_maps = mysql:/etc/postfix/mysql-relocated.cf
transport_maps = mysql:/etc/postfix/mysql-transport.cf
virtual_maps = mysql:/etc/postfix/mysql-virtual.cf
virtual_mailbox_base = /var/vmail
virtual_mailbox_maps = mysql:/etc/postfix/mysql-virtual-maps.cf
virtual_uid_maps = mysql:/etc/postfix/mysql-virtual-uid.cf
virtual_gid_maps = mysql:/etc/postfix/mysql-virtual-gid.cf
local_recipient_maps = $alias_maps $virtual_mailbox_maps
```

```
chmod 644        /etc/postfix/main.cf
chown root:root  /etc/postfix/main.cf
```

```
vi /etc/postfix/master.cf
smtp     inet  n       -       n       -       -       smtpd
pickup    fifo  n       -       n       60      1       pickup
cleanup   unix  n       -       n       -       0       cleanup
qmgr      fifo  n       -       n       300     1       qmgr
rewrite   unix  -       -       n       -       -       trivial-rewrite
bounce    unix  -       -       n       -       0       bounce
defer     unix  -       -       n       -       0       bounce
trace     unix  -       -       n       -       0       bounce
verify    unix  -       -       n       -       1       verify
flush     unix  n       -       n       1000?   0       flush
proxymap  unix  -       -       n       -       -       proxymap
smtp      unix  -       -       n       -       -       smtp
relay     unix  -       -       n       -       -       smtp
showq     unix  n       -       n       -       -       showq
```

```
error     unix  -       -       n       -       -       error
local     unix  -       n       n       -       -       local
virtual   unix  -       n       n       -       -       virtual
lmtp      unix  -       -       n       -       -       lmtp
anvil     unix  -       -       n       -       1       anvil
maildrop  unix  -       n       n       -       -       pipe
 flags=DRhu user=vmail argv=/usr/local/bin/maildrop -d ${recipient}
old-cyrus unix  -       n       n       -       -       pipe
 flags=R user=cyrus argv=/usr/lib/cyrus-imapd/deliver -e -m ${extension}
${user}
cyrus     unix  -       n       n       -       -       pipe
 user=cyrus argv=/usr/lib/cyrus-imapd/deliver -e -r ${sender} -m ${extension}
${user}
uucp      unix  -       n       n       -       -       pipe
 flags=Fqhu user=uucp argv=uux -r -n -z -a$sender - $nexthop!rmail.postfix
($recipient)
ifmail    unix  -       n       n       -       -       pipe
 flags=F user=ftn argv=/usr/lib/ifmail/ifmail -r $nexthop ($recipient)
bsmtp     unix  -       n       n       -       -       pipe
 flags=Fq. user=foo argv=/usr/local/sbin/bsmtp -f $sender $nexthop $recipient
procmail  unix  -       n       n       -       -       pipe
 flags=R user=cyrus argv=/usr/bin/procmail -t -m USER=${user}
EXTENSION=${extension} /home/cyrus/procmailrc




chmod 644       /etc/postfix/master.cf
chown root:root /etc/postfix/master.cf




# CONFIGURE DOVECOT

cd
wget
http://dag.wieers.com/packages/dovecot/dovecot-0.99.13-1.1.el3.rf.i386.rpm
apt-get install rh-postgresql-libs
rpm -Uvh dovecot-0.99.13-1.1.el3.rf.i386.rpm


vi /etc/dovecot.conf
protocols =  imaps pop3s imap pop3
ssl_disable = yes
ssl_cert_file = /etc/ssl/certs/dovecot.pem
ssl_key_file = /etc/ssl/private/dovecot.pem
login = imap
login_executable = /usr/libexec/dovecot/imap-login
login = pop3
login_executable = /usr/libexec/dovecot/pop3-login
mail_extra_groups = mail
default_mail_env = maildir:/var/vmail/%d/%n/Maildir
imap_executable = /usr/libexec/dovecot/imap
pop3_executable = /usr/libexec/dovecot/pop3
auth = default
auth_mechanisms = plain
auth_default_realm = EMAILDOMAIN
auth_userdb = mysql /etc/dovecot-mysql.conf
auth_passdb = mysql /etc/dovecot-mysql.conf
auth_user = root
auth_verbose = yes
```

```
vi /etc/dovecot-mysql.conf
db_host = 127.0.0.1
db_port = 3306
db = mail
db_user = DBUSR
db_passwd = DBPASSWD
db_client_flags = 0
default_pass_scheme = PLAIN
password_query = SELECT clear FROM postfix_users WHERE email = '%n@%d' or
email = '%n@EMAILDOMAIN'
user_query = SELECT maildir, uid, gid FROM postfix_users WHERE email = '%n@%d'
or email = '%n@EMAILDOMAIN'



# CONFIGURE LIFERAY

# configure mailadmin.ksh
cp mailadmin.ksh /usr/sbin
vi  /usr/sbin/mailadmin.ksh
DOMAIN=EMAILDOMAIN              # Domain being managed
MYSQL_USERNAME=DBUSR              # MySQL user
MYSQL_PASSWORD=DBPASSWD             # MySQL password
TOMCAT_UID=500                       # Mail File Creation user id - tomcat
VMAIL_GID=510                       # Mail File Creation group id - vmail


chmod 750           /usr/sbin/mailadmin.ksh
chown tomcat:tomcat /usr/sbin/mailadmin.ksh

# create mailadmin log file
touch               /var/log/mailadmin.log
chmod 660           /var/log/mailadmin.log
chown tomcat:tomcat /var/log/mailadmin.log

# configure liferay to use mailadmin.ksh
vi /usr/local/tomcat/common/classes/portal-ext.properties
  mail.hook.impl=com.liferay.mail.util.ShellHook
  mail.hook.shell.script=/usr/sbin/mailadmin.ksh
  mail.box.style=INBOX

# update these JARs with latest from SVN HEAD
/usr/local/tomcat/common/lib/ext/mail-ejb.jar   -> add
com.liferay.mail.util.ShellHook.class
/usr/local/tomcat/common/lib/ext/portal-ejb.jar -> update
com.liferay.portal.util.PropsUtil.class
/usr/local/tomcat/common/lib/ext/portal-ejb.jar -> update
com.liferay.util.StringUtil.class


# Configure Tomcat

#add mail/MailSession settings
vi /usr/local/tomcat/conf/Catalina/localhost/liferay.xml
<parameter>
       <name>mail.smtp.host</name>
       <value>localhost</value>
</parameter>
<parameter>
       <name>mail.imap.host</name>
       <value>localhost</value>
```

```
</parameter>
<parameter>
       <name>mail.store.protocol</name>
       <value>imap</value>
</parameter>
<parameter>
       <name>mail.transport.protocol</name>
       <value>smtp</value>
</parameter>
<parameter>
       <name>mail.pop3.host</name>
       <value>localhost</value>
</parameter>
```

```
# Enable autostart on reboots

chkconfig postfix on
chkconfig dovecot on


# Verify Install Commands

tail -f 50 /var/log/maillog
tail -f 50 /var/log/messages



# Restart saslauthd before Postfix, so that Postfix doesn't start with
# a bad SASL setup, otherwise it doesn't answer smtp requests
/etc/init.d/saslauthd restart

# make sure saslauthd restarts
ps -ef | grep saslauthd | grep -v grep



# make sure postfix restarts
/etc/init.d/postfix restart
ps -ef | grep postfix | grep -v grep


# make sure dovecot restarts
/etc/init.d/dovecot restart
ps -ef | grep dovecot | grep -v grep


reboot


# make sure everything starts
ps -ef | grep postfix   | grep -v grep
ps -ef | grep dovecot   | grep -v grep
ps -ef | grep saslauthd | grep -v grep
```

```
# Test SMTP by sending an email to LIFERAYUSR@EMAILDOMAIN

telnet localhost 25
EHLO EMAILDOMAIN
MAIL FROM:test@test.com
RCPT TO:LIFERAYUSR@EMAILDOMAIN
DATA
Test msg
.

quit



# Test SMTP by sending an email to alias LIFERAYLOGIN

telnet localhost 25
EHLO EMAILDOMAIN
MAIL FROM:test@test.com
RCPT TO:LIFERAYLOGIN
DATA
Test msg
.

quit




#TEST IMAP by logging in as LIFERAYUSR@EMAILDOMAIN

telnet localhost imap
x LOGIN LIFERAYUSR@EMAILDOMAIN LIFERAYPWD
x STATUS "INBOX" (MESSAGES)
x SELECT "INBOX"
x FETCH 1 BODY[HEADER]
x LOGOUT

# Test using usedId without a Domain name
telnet localhost imap
x LOGIN LIFERAYUSR LIFERAYPWD
x STATUS "INBOX" (MESSAGES)
x SELECT "INBOX"
x FETCH 1 BODY[HEADER]
x LOGOUT
```

Microsoft Exchange

Coming soon...

Multiple Portal Instances

Liferay was built from the ground up to be used by application service providers. The following is a sample list of portals running off of one portal instance hitting one database and shows the capabilities of Liferay: ht-tp://demo.liferay.net, http://my.ccuc.net, http://my.3sixteen.com, http://www.gatewayfriends.org, ht-tp://www.jasonandiris.com. Users in each of these portals have no information about the other portals. They are separated by domain and each portal exists in its own space based on the company's id.

JBoss+Jetty

Coming soon...

JBoss+Tomcat

1. Edit C:\Windows\system32\drivers\etc\hosts so that *www.alpha.com* and *www.beta.com* resolve to your local machine. This will help in testing the setup.

   ```
   127.0.0.1 www.alpha.com
   127.0.0.1 www.beta.com
   ```

2. Install JBoss+Tomcat.

3. Rename /server/default/deploy/ext.ear/portal-web-complete.war to portal-web-alpha.war.

   Copy /server/default/deploy/ext.ear/portal-web-alpha.war to portal-web-beta.war.

4. Edit /server/default/deploy/ext.ear/portal-web-alpha.war/WEB-INF/web.xml and change the value of the company id to *alpha.com*.

   ```
   <context-param>
       <param-name>company_id</param-name>
       <param-value>alpha.com</param-value>
   </context-param>
   ```

   On startup, the portal will create the necessary database entries and create a default user account with **`test@alpha.com`** as the login and **`test`** as the password.

   Repeat this step for /server/default/deploy/ext.ear/portal-web-beta.war/WEB-INF/web.xml but replace *alpha.com* with *beta.com*.

5. Edit /server/default/deploy/ext.ear/portal-web-alpha.war/WEB-INF/jboss-web.xml to add the context root and virtual host.

   ```
   <jboss-web>
       <security-domain>java:/jaas/PortalRealm</security-domain>
       <context-root>/</context-root>
       <virtual-host>www.alpha.com</virtual-host>
       ...
   </jboss-web>

   Repeat this step for
   /server/default/deploy/ext.ear/portal-web-beta.war/WEB-INF/jboss-web.xml
   but replace alpha.com with beta.com.
   ```

6. Edit /server/default/deploy/ext.ear/META-INF/application.xml.

   Remove the old reference to portal-web-complete.war and add the references to portal-web-alpha.war and portal-web-beta.war.

   ```
   <module>
       <web>
           <web-uri>portal-web-alpha.war</web-uri>
   ```

```
        <context-root>/</context-root>
    </web>
</module>
<module>
    <web>
        <web-uri>portal-web-beta.war</web-uri>
        <context-root>/</context-root>
    </web>
</module>
```

7. Edit /server/default/deploy/jbossweb-tomcat50.sar/server.xml to add the alpha and beta hosts.

```
<Engine name="jboss.web" defaultHost="www.alpha.com">
    ...
    <Host name="www.alpha.com" ...
        ...
        <DefaultContext cookies="true" crossContext="true" override="true"
/>
    </Host>
    <Host name="www.beta.com" ...
        ...
        <DefaultContext cookies="true" crossContext="true" override="true"
/>
    </Host>
</Engine>
```

8. Start JBoss+Tomcat and access the portals with your browser at *http://www.alpha.com* and **`http://www.beta.com`**. Your user login and password combinations are **`test@alpha.com/test`** and **`test@beta.com/test`**.

Jetty

Coming soon...

Orion

1. The steps for Orion are more complicated because Orion allows you to share two servlet WARs. Instead of copying the entire portal-web-complete.war (several megs) for each portal instance, you can have a mini WAR that contains the basic XML information and have it reference the larger portal-web.war.

2. Edit C:\Windows\system32\drivers\etc\hosts so that *www.alpha.com* and *www.beta.com* resolve to your local machine. This will help in testing the setup.

```
127.0.0.1 www.alpha.com
127.0.0.1 www.beta.com
```

3. Install Orion with the instructions for your Development Environment [???]. The instructions for Orion will not make sense unless you are developing from the source.

4. Edit /portal/web-sites/web-sites.xml and add entries for *alpha.com* and *beta.com*.

```
<web-site id="alpha.com">
```

```
        <forward-url>/c</forward-url>
    </web-site>
    <web-site id="beta.com">
        <forward-url>/c</forward-url>
    </web-site>
```

5. Run **ant clean start deploy** from /portal.

6. Edit /orion/config/application.xml and add entries for *alpha.com* and *beta.com*.

   ```
   <web-module id="alpha.com-web" path="../applications/alpha.com-web.war" />
   <web-module id="beta.com-web" path="../applications/beta.com-web.war" />
   ```

7. Edit /orion/config/server.xml and add entries for *alpha.com* and *beta.com*.

   ```
   <web-site path="./web-sites/alpha.com-web.xml" />
   <web-site path="./web-sites/beta.com-web.xml" />
   ```

8. Create /orion/config/web-sites/alpha.com-web.xml.

   ```
   <web-site virtual-hosts="www.alpha.com">
       <default-web-app application="default" name="alpha.com-web"
   load-on-startup="true" />
       <web-app application="default" name="portal-web" root="/portal"
   load-on-startup="true" />
       <access-log path="../../log/alpha.com-web-access.log" />
   </web-site>
   ```

   On startup, the portal will create the necessary database entries and create a default user account with **`test@alpha.com`** as the login and **`test`** as the password.

   Repeat this step for /orion/config/web-sites/beta.com-web.xml but replace *alpha.com* with *beta.com*.

9. Start Orion and access the portals with your browser at *http://www.alpha.com* and *http://www.beta.com*. Your user login and password combinations are **`test@alpha.com/test`** and **`test@beta.com/test`**.

Tomcat

Coming soon...

Chapter 3. Customizing the Portal

The following instructions are for those who wish to customize the portal with their own portlets that will not be given back to the general public. These instructions are meant to keep custom code separated from the Liferay code so that upgrades can be made easily. The Ant scripts referred to by these instructions are only available for JBoss+Jetty, JBoss+Tomcat, OracleAS, Orion, RexIP, and WebLogic and have not yet been made available for the other supported application servers.

Follow the instructions found in Development Environment if you want to modify the actual source and contribute your changes back to the project.

Download and unzip the source to /portal. From there, type **ant start**. Then type ant **build-ext** to create /ext which is your extension environment. The location of /ext is configured in /portal/release.properties.

You should see the following directories: /ext/downloads, /ext/ext-ear, /ext/ext-ejb, /ext/ext-lib, /ext/ext-web, /ext/lib, /ext/portlets, /ext/servers, /ext/sql, and /ext/web-sites. Read /ext/readme.txt for basic instructions on how to deploy the extension EAR on the various application servers.

/ext

1. Copy */ext/app.server.properties* to */ext/app.server.${user.name}.properties* where ${user.name} is the user name you used to log into your operating system.

   Edit */ext/app.server.${user.name}.properties* and set *app.server.type*=jboss-jetty to use JBoss/Jetty as your application server. You can modify the value of app.server.type to use other application servers.

2. Copy */ext/build.properties* to */ext/build.${user.name}.properties* where ${user.name} is the user name you used to log into your operating system.

   Edit */ext/build.${user.name}.properties* and set *jsp.precompile*=on to precopmile JSPs for JBoss+Jetty or JBoss+Tomcat. This takes a few minutes and should only be used when deploying to a production server.

/ext/downloads

1. Download liferay-portal-ent-4.0.0-jboss-jetty.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-jboss-jetty-4.0.0.zip?download] to /ext/downloads.

   Run **ant install-jboss-jetty** under /ext/servers to install JBoss+Jetty to */ext/servers/*jboss-jetty.

2. Download liferay-portal-ent-4.0.0-jboss-tomcat.zip to /ext/downloads.

   Run **ant install-jboss-tomcat** under /ext/servers to install JBoss+Tomcat to /ext/servers/jboss-tomcat.

3. Download liferay-portal-pro-4.0.0-jetty.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-jetty-4.0.0.zip?download] to /ext/downloads.

   Run **ant install-jetty** under /ext/servers to install Jetty to /ext/servers/jetty.

4. Download liferay-portal-ent-4.0.0-jonas-jetty.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-jonas-jetty-4.0.0.zip?download] to /ext/downloads.

   Run **ant install-jonas-jetty** under /ext/servers to install JOnAS+Jetty to /ext/servers/jonas-jetty.

5. Download liferay-portal-ent-4.0.0-jonas-tomcat.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-jonas-tomcat-4.0.0.zip?download] to /ext/downloads.

   Run **ant install-jonas-tomcat** under /ext/servers to install JOnAS+Tomcat to /ext/servers/jonas-tomcat.

6. Download oc4j_extended.zip [http://www.oracle.com/] to /ext/downloads.

   Run **ant install-oc4j** under /ext/servers to install OracleAS to /ext/servers/oc4j.

7. Download orion2.0.7 .zip [http://www.orionserver.com/mirrordownload.jsp?file=orion2.0.6.zip] to /ext/downloads.

   Run **ant install-orion** under /ext/servers to install Orion to /ext/servers/orion.

8. Download liferay-portal-tomcat-jdk5-4.0.0.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-tomcat-jdk5-4.0.0.zip?download] to /ext/downloads if you are using Java 1.5.

   If you are using Java 1.4 Download liferay-portal-tomcat-4.0.0.zip [http://prdownloads.sourceforge.net/lportal/liferay-portal-tomcat-4.0.0.zip?download]

   Run **ant install-tomcat** under /ext/servers to install Tomcat to /ext/servers/tomcat.

/ext/ext-ear

1. Run **ant deploy** from /ext/ext-ear to deploy your custom EAR to your specified application server (see /ext/app.server.properties).
2. Your custom EAR utilizes the Liferay JARs and WARs contained in /ext/ext-ear/modules. This allows you to easily upgrade Liferay because all the functionality is packed into archives.

/ext/ext-ejb

1. Important files in this directory are /ext/ext-ejb/classes/portal-ext.properties, /ext/ext-ejb/classes/system-ext.properties, and /ext/ext-ejb/classes/content/Language-ext.properties. See Properties and Languages.
2. Run **ant deploy** from /ext/ext-ejb to compile your source and to deploy your classes to the expanded EAR in your specified application server's deployment directory.

/ext/ext-lib

1. Place any extra dependent libraries in here. They will be copied over to the deployment directory when you run **ant deploy** from /ext.

/ext/ext-web

1. Run ant deploy from /ext/ext-web.

   If /ext/ext-web/tmp does not exist, the script will unjar /ext/ext-ear/modules/portal-web.war into /ext/ext-web/tmp. The script will then copy everything from /ext/ext-web/docroot over to /ext/ext-web/tmp and then copy everything from /ext/ext-web/tmp to the deployment directory of your specified application server.

   This provides an easy way to extend the portal without changing the source and makes upgrading very easy.

2. To add a portlet, edit /ext/ext-web/docroot/WEB-INF/portlet-ext.xml, /ext/ext-web/docroot/WEB-INF/liferay-portlet-ext.xml, /ext/ext-web/docroot/WEB-INF/liferay-display.xml, and /ext/ext-ejb/classes/content/Language-ext.properties. These *-ext.xml files are read after their parent files are read and override their parent values.

   See Portlet Examples for information on general portlet design.

/ext/lib

1. This directory contains all of the portal's deployment time dependent libraries and compile time dependent libraries. They will be copied over to the deployment directory when you run **ant deploy** from /ext.

/ext/portlets

1. You can hot deploy portlets by putting WAR files in this directory. See Hot Deploy for more information.

/ext/servers

1. See the section on /ext/downloads and read /ext/readme.txt.

/ext/sql

1. This directory contains all of the portal's database scripts. See the section called Databases.

   There are also two files named test.properties and test.script that are copied over to your specified application server's bin directory when you run **ant deploy** from /ext. The two files are used by Hypersonic and contain a snapshot of the portal's default data.

/ext/web-sites

1. All the files in /ext/web-sites/liferay.com-web/docroot are copied to the deployment directory when you run **ant deploy** from /ext or /ext/ext-web.

   By default, /ext/web-sites/liferay.com-web/docroot/index.html forwards the user to http://localhost/c/extranet/home. You can replace this file to pretty up the default page or change the default behavior.

You can add more instances of the portal by adding different versions of liferay.com-web in /ext/web-sites. Be sure to edit each instance's web.xml and change the *company_id* parameter. The directory structure is setup in this manner so that you can easily add different instances of the portal based on the *company_id* and virtual hostname. We are working to upgrade the script to automate this process. For now, read the section called Application Service Provider for more information.

Eclipse

1. You can download a .pdf document outlining instructions on how to customize the portal using Eclipse.

Chapter 4. Building Themes

Themes - Liferay Portal 4.0.0

One of the biggest changes for themes in Liferay Portal 4.0.0 is that hot-deployable themes must now use Velocity templates. The following outlines why this change was necessary:

Why Hot-Deployable Themes must use Velocity Templates:

Up until Liferay Portal 4.0.0RC2, Liferay used to place its JARs in the shared class loader. The benefit of this was that all web applications could access Liferay's classes since they were in the shared class loader. The drawback was that JARs in the shared class loader would automatically override JARs in other web applications. So if a web application required Hibernate 2.0 and Liferay required Hibernate 3.0, this would cause conflicts because the JARs in the shared class loader would override the JARs in the web application.

Starting from Liferay Portal 4.0.0 final, Liferay no longer places its JARs in the shared class loader. This allows Liferay to work with web applications that require different versions of Struts, Hibernate, etc. without any changes to the code. This also means that hot-deployable themes cannot access Liferay's classes because they are not in the shared class loader.

Here's a more concrete example with Tomcat as the servlet container:	Tomcat's shared class loader is here: tomcat/common/lib/ext. Instead of placing its JARs in the shared class loader, Liferay now puts them here: tomcat/liferay/WEB-INF/lib. Web applications in tomcat/webapps cannot access Liferay's classes because they are no longer in tomcat/common/lib/ext. If your theme is named *.war, it is a web application. For example, ant-themes-4.0.0.war [???] is a web application. These themes are also called hot-deployable themes since you can deploy them at runtime. Themes that have a .war extension must use Velocity templates because they cannot access Liferay's classes.

Should I use Velocity or JSP?

Use Velocity for cleaner code and better maintainability. We converted all of our community themes [???] to use Velocity. These themes are a great way to learn how to build your own themes using Velocity. Also see VelocityTaglib.java for a list of tags that are available to you.

Use JSP for flexibility or if you do not have time to learn Velocity. If you want to create themes using JSP, you will need to create them in ext/ext-web/docroot/html/themes. See portal/portal-web/docroot/html/themes for examples on how to create JSP themes.

Note: All themes developed in ext/themes or themes that have a .war extension cannot access Liferay's classes because of the new class loader. These themes must be developed using Velocty templates. On the other hand, JSP themes must be developed in ext/ext-web/docroot/html/themes. These themes must be developed here because they need to access Liferay's classes.

Upgrading Themes from 3.6.1 to 4.0.0 - Velocity

Convert your theme to Velocity. See our community themes [???] for examples on how to create themes using Velocity templates.

Upgrading Themes from 3.6.1 to 4.0.0 - JSP

Port your theme to the ext/ext-web/docroot/html/themes directory. If you do not know how to create an extension environment, read this section first: http://content.liferay.com/4.0.0/docs/quickstart/ch04s01.html [???]

The following guidelines will help you to port your theme to Liferay Portal 4.0.0.

1. If you developed your theme as a *.war file or if you developed your theme in ext/themes, deploy your theme to your server. Then copy the resulting theme directory to ext/ext-web/docroot/html/themes. Skip this step if your theme is already located in ext/ext-web/docroot/html/themes.

   Example: If your theme is ant-themes-4.0.0.war, deploy this theme to your server. If you are using Tomcat, the theme will be deployed to tomcat/webapps. Copy the resulting "ant-themes-4.0.0" directory and all of its contents to ext/ext-web/docroot/html/themes

   When you are finished the directory structure should look like this: ext/ext-web/docroot/html/themes/ant-themes-4.0.0. If your "ant-themes-4.0.0" directory is empty you have not deployed the theme correctly.

2. Convert the JSPs:

Before	After
<liferay:runtime-portlet	<liferay-portlet:runtime
<liferay:include flush=”true”	<liferay-util:include Remove "flush" attribute
layoutId	plid
getLayoutId	getPlid
getUserId	getOwnerId
getResTotal	getResolution Only use this method in css.jsp
LayoutServiceUtil.getLayout	LayoutLocalServiceUtil.getLayout
tilesSubNav	This variable was removed

3. Convert icons in portlet_top.jsp to <liferay-portlet> taglib icons:

   <liferay-portlet:icon-configuration /> <liferay-portlet:icon-edit /> <liferay-portlet:icon-edit-guest /> <liferay-portlet:icon-help /> <liferay-portlet:icon-print /> <liferay-portlet:icon-minimize /> <liferay-portlet:icon-maximize /> <liferay-portlet:icon-close />

4. Convert hrefs in top.jsp to use themeDisplay getter methods

   getURLMyAccount getURLSignOut getURLAddContent getURLPageSettings getURLSignIn

5. Convert “Communities” to “My Places”

   <div id="layout-my-places"> <liferay-portlet:runtime portletName="<%= PortletKeys.MY_PLACES %>" /> </div>

6. Images

   - Liferay convention is to put all custom images into images/custom
   - Copy default images from Classic theme.

7. CSS Styles

- Remove any styles with “n1” or “w1” in the name
- Add the following styles:

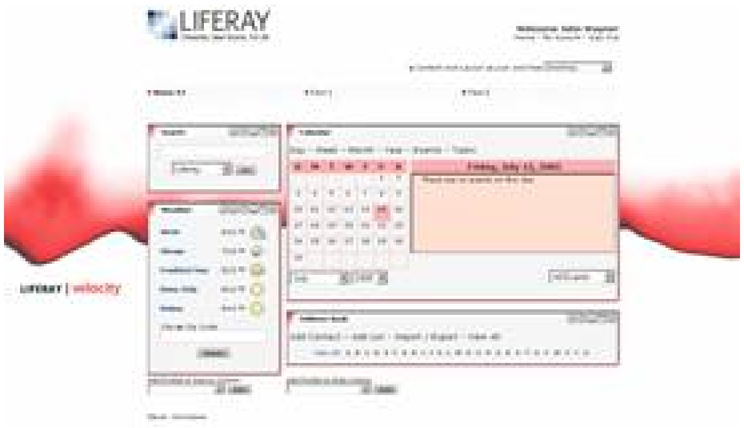
  portal-add-content layout-column_column-? layout-column-highlight portlet-dragging-placeholder layout-my-places
- Change the following styles:

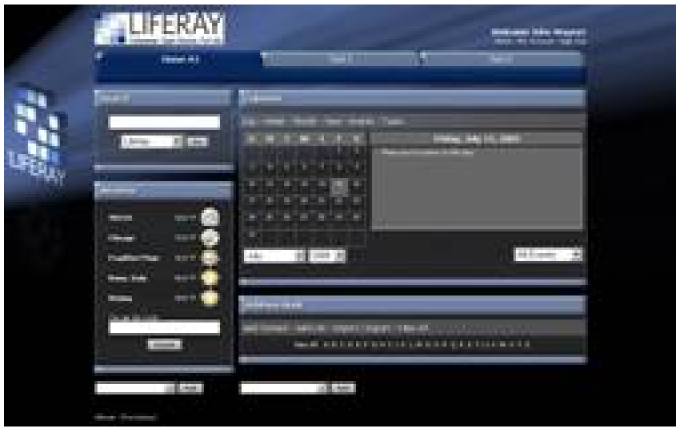
Before	After
#gamma-tab	.gamma-tab
#current	.current

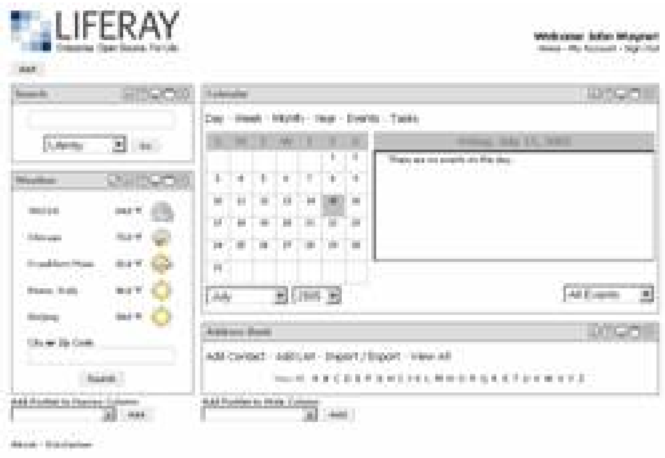
Themes - Liferay Portal 3.6.1

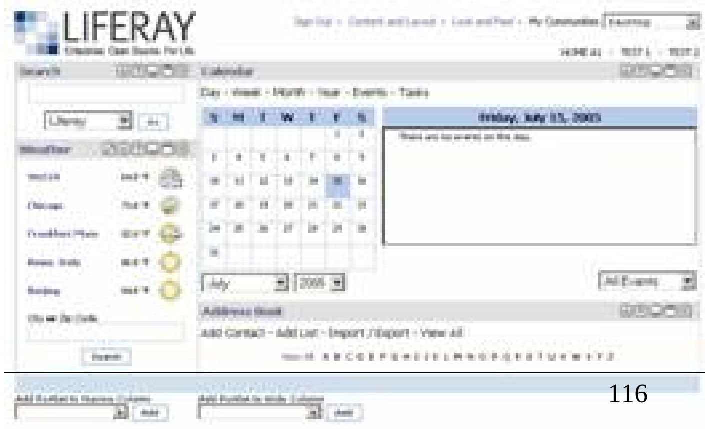
This tutorial will explain Liferay Portal’s themes feature in detail. It will also discuss how to use Themes in conjunction with other features to easily build your website within Liferay Portal.

Themes make it possible to easily switch to different presentations or "look and feel" layers. Within a single .war file, a designer/developer can deliver an integrated package of JSP (or Velocity), Javascript, image, and configuration files that will control all presentation logic and design attributes for a portal community. Liferay Portal comes with a handful of pre-made themes that showcase its versatility:


LIFERAY
LIFERAY | velocity


LIFERAY


LIFERAY
Enterprise Open Source. For Life.
Welcome John Wayne!
Search
Calendar
Day - Week - Month - Year - Events - Tasks
Weather
Address Book
Add Contact - Add List - Import / Export - View All


LIFERAY
Enterprise Open Source. For Life.
Search
Calendar
Day - Week - Month - Year - Events - Tasks
Friday, July 15, 2005
Weather
Address Book
Add Contact - Add List - Import / Export - View All

- Different themes can be assigned to specific user groups (a.k.a. communities)
- Users can choose a unique theme for their own personal portal page
- Both Java Server Page (JSP) and Velocity (VM) languages are supported
- Themes are hot-deployable as .wars (when supported by the app server)

To select a different theme, go to the **Look and Feel** section in the header bar and choose one of the available themes and a corresponding color scheme. The process is as straightforward as setting a new desktop background in Windows.

Making Your Own Theme

Making a new theme requires only four main steps:

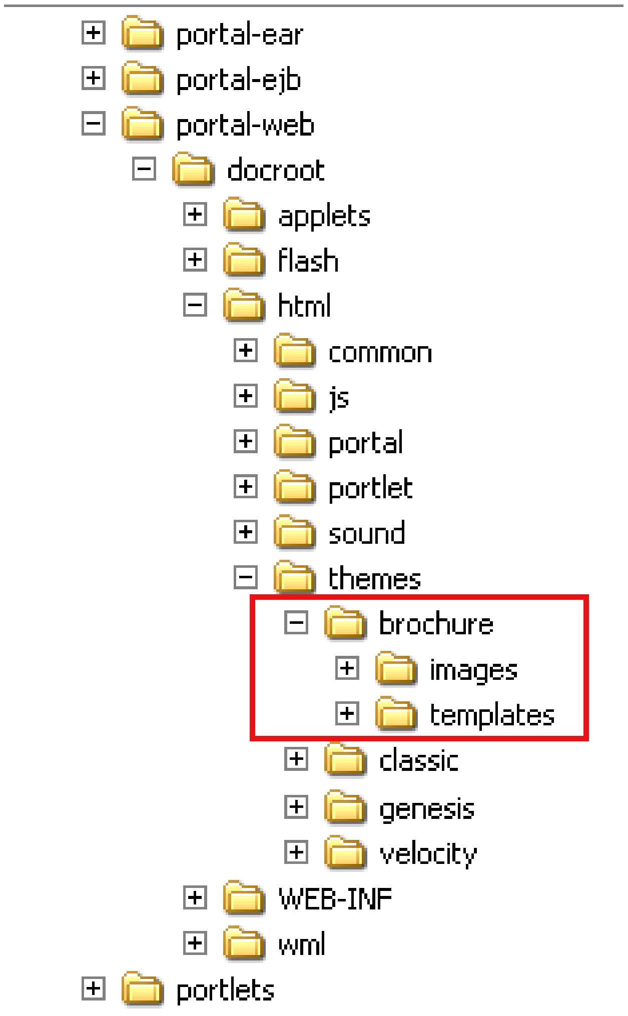
1. Configure properties
2. Edit templates (JSP or VM)
3. Define CSS styles
4. Code JavaScript (optional)

Let’s take a high-level look at how Liferay created the JSP-based "Brochure" theme (Figure 2.6) by modifying a few things in a copy of the "Classic" theme (Figure 2.1).

Note: If you want to see an example using VM, take a look at the bundled "Velocity" theme (Figure 2.3).

Configuration

First, we created a copy of the "Classic" theme’s directory and renamed it "brochure." (Figure 2.1.1.1). We then deleted the color-schemes directory because we decided we would only have one color scheme for the "Brochure" theme. Although you do not have to follow the layout shown here, this convention will make your themes easier to organize and keep them standard.



```
⊞ portal-ear
⊞ portal-ejb
⊟ portal-web
    ⊟ docroot
        ⊞ applets
        ⊞ flash
        ⊟ html
            ⊞ common
            ⊞ js
            ⊞ portal
            ⊞ portlet
            ⊞ sound
            ⊟ themes
                ⊟ brochure
                    ⊞ images
                    ⊞ templates
                ⊞ classic
                ⊞ genesis
                ⊞ velocity
        ⊞ WEB-INF
        ⊞ wml
⊞ portlets
```

Figure 2.1.1.1 Directory structure

Once a directory is created to house your new theme, you need to let Liferay know the new theme's directory location. This is done in liferay-look-and-feel.xml (Figure 2.1.1.2). Notice that the theme configuration shows theme id="brochure" and identifies the compatible version. The root-path, templates-path and images-path are also set here. Make sure that these values match your directory structure.

```
<look-and-feel>
<compatibility>
<version>3.6.1</version>
</compatibility>
<theme id="brochure" name="Brochure">
<root-path>/html/themes/brochure</root-path>
<templates-path>/html/themes/brochure/templates</templates-path>
<images-path>/html/themes/brochure/images</images-path>
<template-extension>jsp</template-extension>
<color-scheme id="01" name="Default">
<!-- color-scheme content omitted to simplify example -->
</color-scheme>
</theme>
</look-and-feel>
```

Figure 2.1.1.2 Liferay-look-and-feel.xml

Templates

Now that Liferay Portal is configured to load your theme, the next step is to edit the template files that give the theme its uniqueness. Here is a brief description of some of the templates that are used in the default theme.

init.jsp	Initializes variables and properties needed for the theme.
css.jsp	Contains CSS style definitions for your entire theme.
portal_normal.jsp	Controls the layout of portal templates for normal pages.
portal_pop_up.jsp	Controls the layout of portal templates for pop-ups.
top.jsp	Draws the top of the portal.
bottom.jsp	Draws the bottom of the portal.
navigation.jsp	Draws the navigation bar of the portal.
portlet_top.jsp	Draws the top of each portlet.
Draws the top of each portlet.	Draws the bottom of each portlet.
javascript.jsp	Contains JavaScript declarations.

Each template controls the layout of a particular area within the portal. For instance, portlet_top.jsp controls the top portion of each portlet; navigation.jsp determines how the navigation bar will look.

Keep in mind that not all of these templates require modification nor are you limited to the ones mentioned here. Feel free to add another template to control a different area of the portal. Just make sure to include the template in `portal_normal.jsp and/or portal_pop_up.jsp`.

A number of changes were made to these templates for the brochure theme. In the figures below, you can see one such major change made to the placement and appearance of the portal configuration menu (Figures 2.1.2.1 and 2.1.2.2):

Figure 2.1.2.1 The "Classic" theme's portal config menu is integrated with the tabs.

Figure 2.1.2.2 The "Brochure" theme's portal config menu is relocated to the top of the portal.

In the "Classic" theme, the portal configuration menu is part of the navigation bar. It consists of buttons and a drop-down menu. In contrast, the "Brochure" theme has the portal configuration menu relocated to the top of the portal and separated from the navigation bar. Also notice that the buttons are now simple text links. Some of the required code changes are displayed below in Figure 2.1.2.3:

```
<div id="layout-outer-side-decoration">
<div id="layout-inner-side-decoration">
<div id="layout-top-decoration">
<div id="layout-corner-ul"></div>
<div id="layout-corner-ur"></div>
</div>
<div id="layout-box">
<div id="layout-top">
<div id="layout-logo">
<a class="bg" href="<%= themeDisplay.getPathFriendlyURL() %>/guest/home"><img
border="0" hspace="0" src="<%= themeDisplay.getCompanyLogo() %>"
vspace="0"></a>
</div>

<c:if test="<%= themeDisplay.isSignedIn() %>">
<div id="layout-user-menu">
```

```
<a style="font-size: 8pt;" href="<%= themeDisplay.getURLSignOut()
%>"><bean:message key="sign-out" /></a> -

<c:if test="<%=
GetterUtil.getBoolean(PropsUtil.get(PropsUtil.UNIVERSAL_PERSONALIZATION)) ||
RoleLocalServiceUtil.isPowerUser(user.getUserId()) %>">
<a style="font-size: 8pt" href="<%= themeDisplay.getPathMain()
%>/portal/personalize_forward?group_id=<%= portletGroupId %>"><%=
LanguageUtil.get(pageContext, "content-and-layout") %></a> -
</c:if>

<c:if test="<%=
GetterUtil.getBoolean(PropsUtil.get(PropsUtil.LOOK_AND_FEEL_MODIFIABLE)) %>">
<a style="font-size: 8pt" href="<%= themeDisplay.getPathMain()
%>/portal/look_and_feel_forward?group_id=<%= portletGroupId %>"><%=
LanguageUtil.get(pageContext, "look-and-feel") %></a> -
</c:if>

<font class="portlet-font" style="font-size: 8pt;">
<%= LanguageUtil.get(pageContext, "my-communities") %>
</font>

<% String selectedStyle = "style=\"background: " +
colorScheme.getPortletMenuBg() + "; color: " +
colorScheme.getPortletMenuText() + ";\" "; %>

<font size="2">
<select name="my_communities_sel" style="font-family: Verdana, Arial;
font-size: smaller; font-weight: normal;" onChange="self.location = '<%=
themeDisplay.getPathMain() %>/portal/group_forward?group_id=' + this.value;">
<option <%= !layout.isGroup() ? selectedStyle : "" %> value="<%=
Group.DEFAULT_PARENT_GROUP_ID %>"><%= LanguageUtil.get(pageContext, "desktop")
%></option>

<% List myCommunities = UserLocalServiceUtil.getGroups(user.getUserId());
for (int i = 0; i < myCommunities.size(); i++) {
Group myCommunity = (Group)myCommunities.get(i);
%>
<option <%= layout.isGroup() &&
layout.getGroupId().equals(myCommunity.getGroupId()) ? selectedStyle +
"selected" : "" %> value="<%= myCommunity.getGroupId() %>"><%=
myCommunity.getName() %></option>

<% } %>

</select>
</font>
</div>
</c:if>
```

Figure 2.1.2.3 top.jsp for the "Brochure" theme

CSS Styles

Another major part of every theme is the collection of presentation attributes such as font size, background color, padding, and margin. The best way to control these attributes in your theme is to use cascading style sheets (CSS). As mentioned briefly above, css.jsp contains all the CSS style definitions for your theme. Liferay Portal’s default themes include all JSR-168 standard style definitions. You may also add theme-specific style definitions to css.jsp. Using the templates and adding custom images as necessary, you can use standard and custom style definitions to give your theme a distinctive look and feel. Let’s look at some of the style changes made in the "Brochure" theme.

Header Bar

```
.portlet-header-bar {
    background-color: #e0e0e0;
    margin: 0 1px 0 1px;
    padding: 3px 0px 3px 0px;
    position: relative;
    z-index: 2;
    }
```

```
    <%@ include file="init.jsp" %>

    <div class="portlet-container" style="width: <%= portletDisplay.getWidth()
%>">
    <div class="portlet-header-bar">
    <c:if test="<%= Validator.isNotNull(portletDisplay.getTitle()) %>">
    <div class="portlet-title">
    <b> <%= portletDisplay.getTitle() %> </b>
    </div>
    </c:if>
```

The style definition portlet-header-bar is used in the portlet_top template. The portlet_top template is used to draw the top of each portlet within the portal. As you can see from the portlet-header-bar definition, the header bar for each portlet should have a gray background color, with a small margin on its sides, padding on top and bottom, positioned relatively and with a z-index of 2.

Dynamic Style Values

Style values can be retrieved dynamically from color scheme variables defined in liferay-look-and-feel.xml. Because the style sheet is a JSP file, you can use JSP tags to dynamically assign some of your style definition values. For example:

```
.portlet-title {
    color: <%= colorScheme.getPortletTitleText() %>
    font-family: Tahoma, Arial;
    font-size: smaller;
    vertical-align: middle;
    }
```

In the above code, portlet-title's color attribute has its value retrieved dynamically. This way, depending on the color scheme selected on the "Look and Feel" page in the portal, the title bar will have a color appropriate to the currently selected color scheme.

Please refer to the theme user guide for more information.

JavaScript

You can use JavaScript to create a more dynamic interface for your theme. To use a script, put the script file inside your theme directory, then provide the location of this file in the script declaration in `javascript.jsp`. For example, if your JavaScript file is called `my_script.js` and it is located in `html/themes/my_themes/js`, make the following declaration in `javascript.jsp`:

```
 <script language="JavaScript" src="<%= themeDisplay.getPathThemeRoot()
%>/js/my_script.js"></script>
```

The JSP expression `<%= themeDisplay.getPathThemeRoot() %>` helps keep you from hard-coding information in your templates. Refer to the API to see other variables and methods available to you.

Both the "Clean" theme (Figure 2.5) and the "Genesis" theme (Figure 2.4) use JavaScript to create a dynamic interface.

CMS + Portal = Website

In addition to themes, Liferay Portal has several enhancements that make it very straightforward to combine static CMS articles and functionality-rich portlets on the same portal page. One direct application of this capability is to create your public website using Liferay, and thereby enable the inclusion of portlets on your website. Not only would you be able to reuse functional components and find innovative ways to use them, but you could also quickly build new pages out of common components within a tightly-managed CMS framework.

As an example, suppose an insurance company wants to show their prospective customers how much they can save by switching to their insurance plan. Online visitors can go to their website, fill out a short form, and receive pricing information instantaneously. If this insurance company uses Liferay to build their website, they can easily develop a pricing calculator portlet, grant guest access, and display it on the front page of their website. The CMS-managed article describing the plan can be placed right beside the calculator portlet.

Also, incorporating portlets into your website helps to encourage reuse. If two different pages on your website need to offer the same functionality, one portlet can simply be placed in both pages.

Liferay Portal also comes with portlets specifically designed to help you build a public website easily. The Breadcrumb Portlet keeps track of how deep a user has gone into the navigation of a site. The Navigation Portlet can be configured to act as a standard nested navigation menu. These and other portlets can be re-used and easily included on every content page where they are needed. Once the main Theme layout has been created, pages can be put together using these portlets without touching the presentation code. Layout is handled strictly through portal page configuration.

Besides providing functionality, a website also needs to provide information. Liferay Portal allows CMS articles to be inserted into your portal just like any other portlet. You can thus easily bring various articles together to assemble a website.

Using the above example, the insurance company may want one portlet containing a CMS article describing their pricing. Right next to it is another portlet with CMS articles on customer testimonials. The layout of the page the articles live in is managed within the portal’s admin section, and the content and layout of the individual articles is easily maintained through the journal portlet.

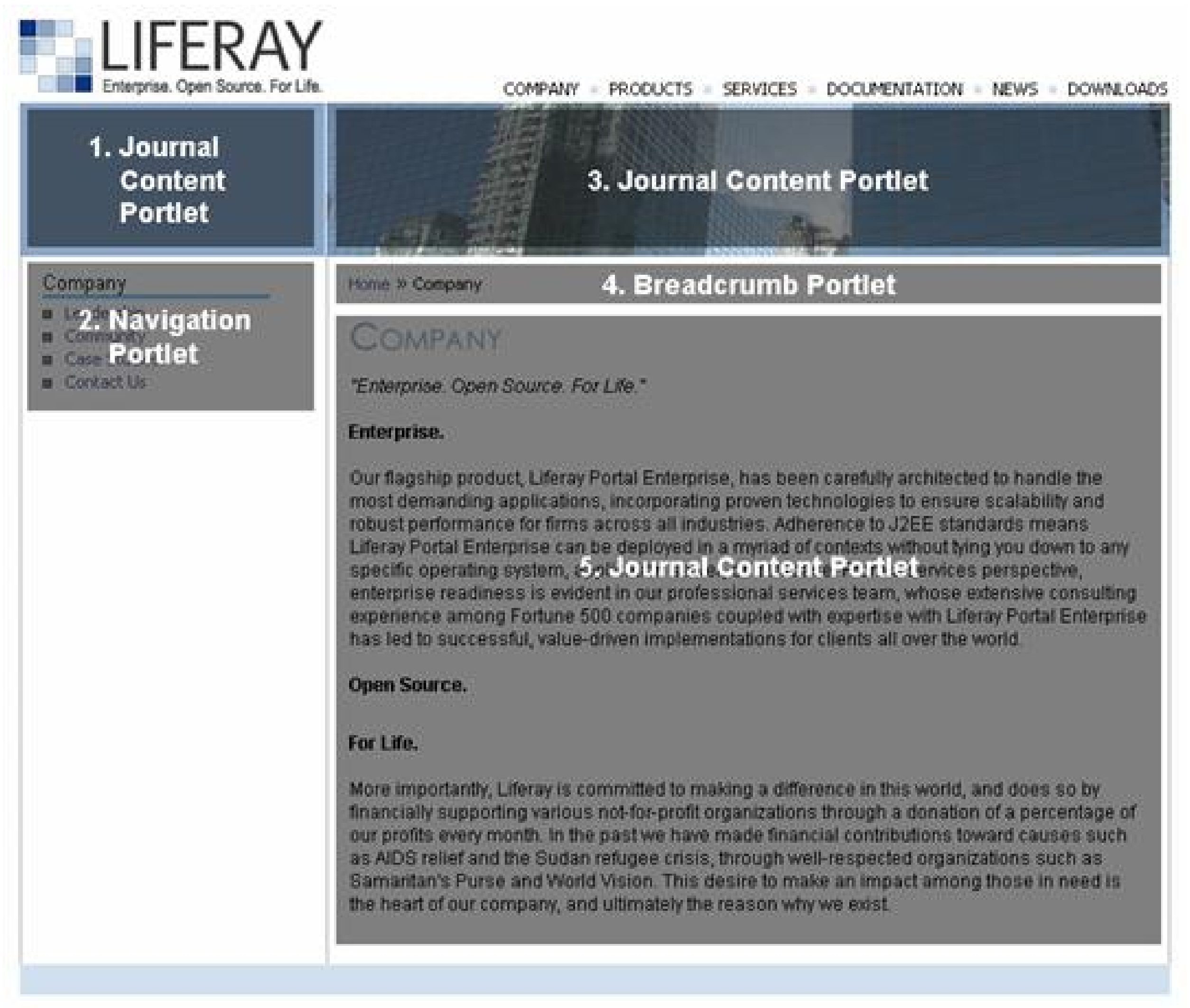
Liferay Portal also ships with the Runtime Portlet and the Portlet Aggregator. The Runtime Portlet is used to embed portlets into CMS articles. This portlet is used at Liferay.com to embed an instance of the Login Portlet within the article that is displayed on the main web page.

The Portlet Aggregator provides a flexible means to lay out several portlets. You are no longer confined to the strict rigidity of a column layout, but you can set the layout of several portlets dynamically using HTML and CSS.

Note: For documentation on Liferay’s CMS, please see the **Liferay Portal User Guide**.

Example of CMS and Web Portal Integration

Liferay’s main website was recently rebuilt to utilize the portal instead of using linked CMS pages. In this new version, each page on the website is a layout (page) comprised of several different portlets. To illustrate, let’s take Liferay’s "Company Overview" page as an example. Figure 2.1.4.1 shows the webpage with each of the five portlets marked. **(You can view the actual page here)**



COMPANY

"Enterprise. Open Source. For Life."

Enterprise.

Our flagship product, Liferay Portal Enterprise, has been carefully architected to handle the most demanding applications, incorporating proven technologies to ensure scalability and robust performance for firms across all industries. Adherence to J2EE standards means Liferay Portal Enterprise can be deployed in a myriad of contexts without tying you down to any specific operating system, [illegible] services perspective, enterprise readiness is evident in our professional services team, whose extensive consulting experience among Fortune 500 companies coupled with expertise with Liferay Portal Enterprise has led to successful, value-driven implementations for clients all over the world.

Open Source.

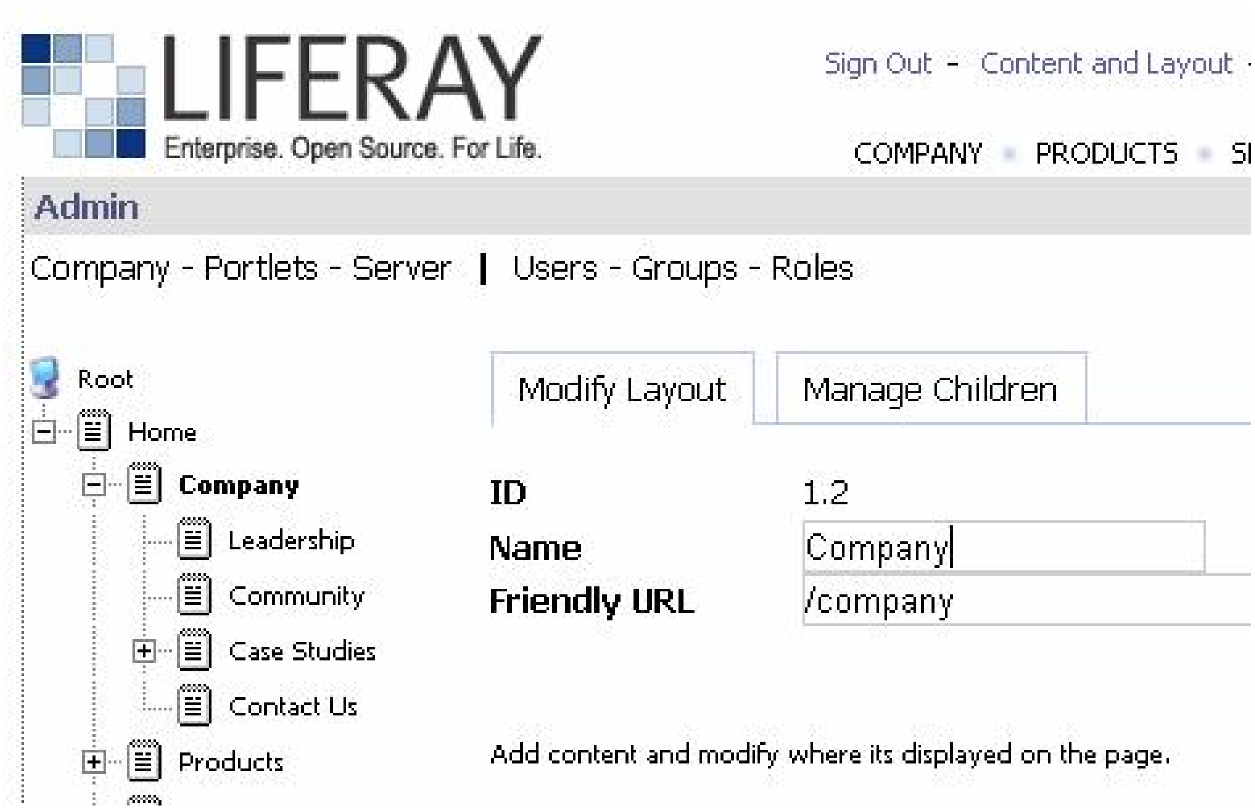
For Life.

More importantly, Liferay is committed to making a difference in this world, and does so by financially supporting various not-for-profit organizations through a donation of a percentage of our profits every month. In the past we have made financial contributions toward causes such as AIDS relief and the Sudan refugee crisis, through well-respected organizations such as Samaritan's Purse and World Vision. This desire to make an impact among those in need is the heart of our company, and ultimately the reason why we exist.

Figure 2.1.4.1 Company Overview

"Company Overview," like most other pages on Liferay.com, is a portal layout (page) with two columns, one narrow and one wide, containing various portlets that supply its content. Portlets two and four, denoted in Figure 2.1.4.1, are instances of the Navigation and Breadcrumb Portlets, respectively. The remaining three portlets are instances of the Journal Content portlet; each contains its own CMS article. Portlet one and three contain a single picture each, and portlet five contains a detailed overview of Liferay.

You can add, delete, and organize page layouts by using the Admin section (Figure 2.1.4.2), accessed by the "Content and Layout" button.



ID 1.2
Name Company
Friendly URL /company

Add content and modify where its displayed on the page.

Figure 2.1.4.2 Admin Portlet

For the Company page, we added the "Company" layout right under the "Home" layout. Also notice the Friendly URL field. This field allows you to assign a simple URL to a layout. So instead of linking to this page with the URL www.liferay.com/c/portal/layout?p_p_id=34... [about:blank], you can use a simplified URL like www.liferay.com/web/guest/company [about:blank].

Once the "Company" page layout has been added, you can add different portlets to quickly assemble your webpage.

For example, to add the Breadcrumb Portlet to the right column:

1. Go to the bottom of the right column
2. Choose **Breadcrumb** in the dropdown menu
3. Click the **Add** button

To add a CMS article in the left column:

1. Go to the bottom of the left column
2. Choose **Journal Content** in the dropdown menu
3. Click the **Add** button
4. Click the **Edit** button to specify the ID of the article you want
5. Specify whether you would like to see a border

6. Add an optional title and click the **Save Settings** button

Note: When assembling your webpage, remember to extract all the presentation attributes to your theme. This way, all the attributes are centralized. For example, to remove the gap between the wide and narrow columns, we set the "margin" attribute in the "Brochure" theme to zero.

Conclusion

Liferay provides a lot of flexibility and control to customize the overall user experience of your portal or website. The availability of a number of different ways to manage, use, and display CMS content and portlets provides limitless possibilities for innovation using Liferay Portal.

To see Liferay Portal in action, visit demo.liferay.net and create a user account for yourself. You can also explore Liferay.com to get a sense of how the portal was used to build that site and begin to imagine the possibilities for your own website or portal.

Chapter 5. Code Generation

service.xml

Liferay Portal features a convenient code generation module that allows the developer to auto generate all of the "Model" components for a portlet. The business and persistance tier classes are auto generated based on specifications outlined in a file called service.xml. Each portlet can have its own service.xml where the portlet and each entity in the portlet are defined. A portlet entity is defined as a table specifying primary keys, audit fields, and other fields used in the table as well as any finder methods that will be required. Custom portlet exceptions can also be auto generated.

Below is an example service.xml file for the Bookmarks Portlet.

```
<?xml version="1.0"?>
<!DOCTYPE service-builder PUBLIC "-//Liferay//DTD Service Builder 4.0.0//EN"
"http://www.liferay.com/dtd/liferay-service-builder_4_0_0.dtd">

<service-builder root-dir=".." package-path="com.liferay.portlet">
        <portlet name="Bookmarks" short-name="Bookmarks" />
        <entity name="BookmarksEntry" local-service="true"
remote-service="true">

                <!-- PK fields -->

                <column name="entryId" type="String" primary="true" />

                <!-- Audit fields -->

                <column name="companyId" type="String" />
                <column name="userId" type="String" />
                <column name="createDate" type="Date" />
                <column name="modifiedDate" type="Date" />

                <!-- Other fields -->

                <column name="folderId" type="String" />
                <column name="name" type="String" />
                <column name="url" type="String" />
                <column name="comments" type="String" />
                <column name="visits" type="int" />

                <!-- Order -->

                <order by="asc">
                        <order-column name="folderId" />
                        <order-column name="name" case-sensitive="false" />
                </order>

                <!-- Finder methods -->

                <finder name="FolderId" return-type="Collection">
                        <finder-column name="folderId" />
                </finder>

                <!-- References -->

                <reference package-path="com.liferay.portal" entity="Resource"
/>
        </entity>
        <entity name="BookmarksFolder" local-service="true"
remote-service="true">

                <!-- PK fields -->

                <column name="folderId" type="String" primary="true" />

                <!-- Group instance -->

                <column name="groupId" type="String" />

                <!-- Audit fields -->

                <column name="companyId" type="String" />
                <column name="userId" type="String" />
                <column name="createDate" type="Date" />
                <column name="modifiedDate" type="Date" />

                <!-- Other fields -->

                <column name="parentFolderId" type="String" />
                <column name="name" type="String" />
                <column name="description" type="String" />

                <!-- Order -->

                <order by="asc">
                        <order-column name="parentFolderId" />
                        <order-column name="name" case-sensitive="false" />
                </order>

                <!-- Finder methods -->

                <finder name="GroupId" return-type="Collection">
                        <finder-column name="groupId" />
                </finder>
                <finder name="G_P" return-type="Collection">
                        <finder-column name="groupId" />
                        <finder-column name="parentFolderId" />
                </finder>

                <!-- References -->

                <reference package-path="com.liferay.portal" entity="Resource"
/>
        </entity>
        <exceptions>
                <exception>EntryURL</exception>
                <exception>FolderName</exception>
        </exceptions>
</service-builder>
```

Upon reading service.xml found in the Bookmarks portlet, the following model classes are generated. Each model class reflects a table in the database. Never edit BookmarksEntryModel. Do edit BookmarksEntry to add hand massaged code. BookmarksEntry is generated once and extends BookmarksEntryModel. This allows the ease of generated code and flexibility of hand massaged code.

Most of the EJBs, HBMs, and Models are generated through the ant task build-service, which reads the file service.xml in /portal-ejb. Each portlet that persist data has its own service.xml (do a search in /portal-ejb and you will get a list back). Copy this file to /portal-ejb when you want to generate the persistence classes for that portlet. This is an internal tool that is built on top of the XDoclet engine.

For example, upon reading service.xml found in the Bookmarks portlet, the following model classes are generated. Each model class reflects a table in the database. Never edit BookmarksEntryModel. Do edit BookmarksEntry to add hand massaged code. BookmarksEntry is generated once and extends BookmarksEntryModel. This allows you the ease of generated code and flexibility of hand massaged code.

com.liferay.portlet.bookmarks.model.BookmarksEntry

com.liferay.portlet.bookmarks.model.BookmarksEntryModel

com.liferay.portlet.bookmarks.model.BookmarksFolder

com.liferay.portlet.bookmarks.model.BookmarksFolderModel

Hibernate classes are generated that map to the model classes. This allows for an n-tier architecture for cases where your model classes are marshalled across the wire and your Hibernate classes are not.

com.liferay.portlet.bookmarks.service.persistence.BookmarksEntryHBM

com.liferay.portlet.bookmarks.service.persistence.BookmarksFolderHBM

Persistence methods to add, update, delete, find, remove, and count the Hibernate entries are generated as the default persistence mechanism.

com.liferay.portlet.bookmarks.service.persistence.BookmarksEntryPersistence

com.liferay.portlet.bookmarks.service.persistence.BookmarksFolderPersistence

Helper classes are generated that call the persistence methods. By default, the helper classes call the Hibernate persistence methods to update the database. You can override this in portal.properties and set your own persistence class as long as it extends the default persistence class. This means you can customize where you store your data. It can be a traditional database, a LDAP server, or even something else.

com.liferay.portlet.bookmarks.service.persistence.BookmarksEntryUtil

com.liferay.portlet.bookmarks.service.persistence.BookmarksFolderUtil

Pooling classes are also created to minimize object creation. Behavior can be modified in **portal.properties**.

com.liferay.portlet.bookmarks.service.persistence.BookmarksEntryPool

com.liferay.portlet.bookmarks.service.persistence.BookmarksFolderPool

POJO implementations that extend PrincipalBean are generated to hold business logic that check the caller principal and can be called **remotely**. Calling getUserId() returns the user id of the current user. Calling getUser() returns the User model that represents the current user. The Session EJB that extends the POJO implementation implements PrincipalSessionBean.

For example, these classes allow you to delete a bookmark entry or folder only if you are the creator of that entry or folder.

These classes are only generated once if they do not already exist.

com.liferay.portlet.bookmarks.service.impl.BookmarksEntryServiceImpl

com.liferay.portlet.bookmarks.service.impl.BookmarksFolderServiceImpl

Helper classes are generated based on the POJO implementations. They help save developer time and prevent polluted code. Instead of writing many lines of code just to look up the appropriate Session EJB wrapper or POJO implementation, you simply call BookmarksEntryServiceUtil.addEntry to call the equivalent method in BookmarksEntryServiceImpl.addEntry.

BookmarksEntryServiceUtil calls BookmarksFolderServiceFactory to look up the class that implements BookmarksEntryService. BookmarksFolderServiceFactory defers to Spring and settings in portal.properties on whether to load the Session EJB wrapper or the plain POJO implementation. The Session EJB extends the POJO implementation.

com.liferay.portlet.bookmarks.service.ejb.BookmarksEntryServiceEJB

com.liferay.portlet.bookmarks.service.ejb.BookmarksEntryServiceEJBImpl

com.liferay.portlet.bookmarks.service.ejb.BookmarksEntryServiceHome

com.liferay.portlet.bookmarks.service.spring.BookmarksEntryService

com.liferay.portlet.bookmarks.service.spring.BookmarksEntryServiceFactory

com.liferay.portlet.bookmarks.service.spring.BookmarksEntryServiceUtil

com.liferay.portlet.bookmarks.service.ejb.BookmarksFolderServiceEJB

com.liferay.portlet.bookmarks.service.ejb.BookmarksFolderServiceEJBImpl

com.liferay.portlet.bookmarks.service.ejb.BookmarksFolderServiceHome

com.liferay.portlet.bookmarks.seervice.spring.BookmarksFolderService

com.liferay.portlet.bookmarks.service.spring.BookmarksFolderServiceFactory

com.liferay.portlet.bookmarks.service.spring.BookmarksFolderServiceUtil

Tunneling classes are generated so that developers can call the POJO implementations over port 80. An example of this is given in section V of this document.

com.liferay.portlet.bookmarks.service.http.BookmarksEntryServiceHttp

com.liferay.portlet.bookmarks.service.http.BookmarksFolderServiceHttp

Soap classes are generated so that developers can call the POJO implementations over port 80. Soap is slower than tunneling because tunneling streams request in binary format. Soap is more flexible than tunneling because the client classes are not limited to Java.

com.liferay.portlet.bookmarks.service.http.BookmarksEntryServiceSoap

com.liferay.portlet.bookmarks.service.http.BookmarksFolderServiceSoap

POJO implementations classes that do **not** extend PrincipalBean are generated to hold business logic that do **not** check the caller principal and can be called **locally**. These classes exist so that business logic can be easily integrated with other projects.

These classes are only generated once if they do not already exist.

com.liferay.portlet.bookmarks.service.impl.BookmarksEntryLocalServiceImpl

com.liferay.portlet.bookmarks.service.impl.BookmarksFolderLocalServiceImpl

Helper classes are also generated.

com.liferay.portlet.bookmarks.service.ejb.BookmarksEntryLocalServiceEJB

com.liferay.portlet.bookmarks.service.ejb.BookmarksEntryLocalServiceEJBImpl

com.liferay.portlet.bookmarks.service.ejb.BookmarksEntryLocalServiceHome

com.liferay.portlet.bookmarks.service.spring.BookmarksEntryLocalService

com.liferay.portlet.bookmarks.service.spring.BookmarksEntryLocalServiceFactory

com.liferay.portlet.bookmarks.service.spring.BookmarksEntryLocalServiceUtil

com.liferay.portlet.bookmarks.service.ejb.BookmarksFolderLocalServiceEJB

com.liferay.portlet.bookmarks.service.ejb.BookmarksFolderLocalServiceEJBImpl

com.liferay.portlet.bookmarks.service.ejb.BookmarksFolderLocalServiceHome

com.liferay.portlet.bookmarks.service.spring.BookmarksFolderLocalService

com.liferay.portlet.bookmarks.service.spring.BookmarksFolderLocalServiceFactory

com.liferay.portlet.bookmarks.service.spring.BookmarksFolderLocalServiceUtil

Some of our users needed to call the Local Service classes remotely, so Remote Service classes that parallel their Local counterparts are also generated.

com.liferay.portlet.bookmarks.service.ejb.BookmarksEntryRemoteServiceEJB

com.liferay.portlet.bookmarks.service.ejb.BookmarksEntryRemoteServiceEJBImpl

com.liferay.portlet.bookmarks.service.ejb.BookmarksEntryRemoteServiceHome

com.liferay.portlet.bookmarks.service.spring.BookmarksEntryRemoteService

com.liferay.portlet.bookmarks.service.spring.BookmarksEntryRemoteServiceFactory

com.liferay.portlet.bookmarks.service.spring.BookmarksEntryRemoteServiceUtil

com.liferay.portlet.bookmarks.service.ejb.BookmarksFolderRemoteServiceEJB

com.liferay.portlet.bookmarks.service.ejb.BookmarksFolderRemoteServiceEJBImpl

com.liferay.portlet.bookmarks.service.ejb.BookmarksFolderRemoteServiceHome

com.liferay.portlet.bookmarks.service.spring.BookmarksFolderRemoteService

com.liferay.portlet.bookmarks.service.spring.BookmarksFolderRemoteServiceFactory

com.liferay.portlet.bookmarks.service.spring.BookmarksFolderRemoteServiceUtil

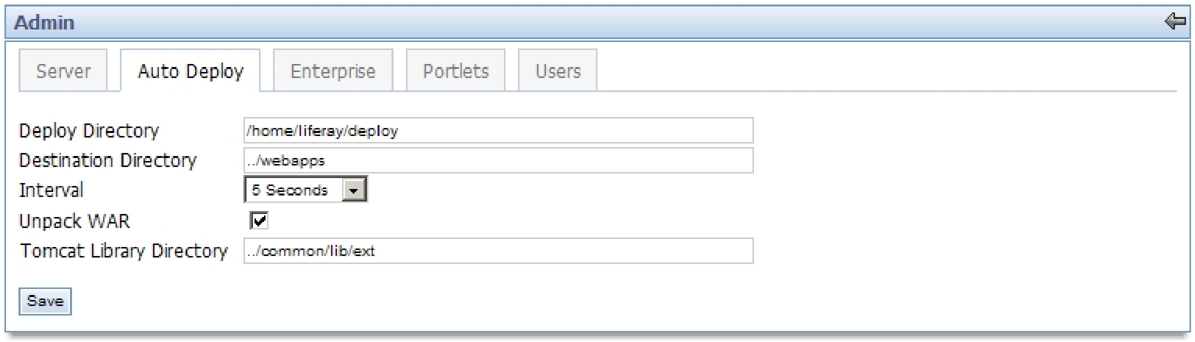
Chapter 6. Deployment

Hot Deploy

Liferay allows you to easily hot deploy layout templates, JSR 168 portlets, and themes. Layout templates allow portlets to be arranged inside the constraints of custom layouts. JSR 168 portlets add functional abilities to the portal. Themes modify the look and feel of the portal. Layout templates, portlets and themes can be deployed at runtime by utilizing the hot deploy features of Liferay.

Layout Templates, Portlets and Themes

1. Go to the drive where you installed your server. Check to see that /home/liferay/deploy exists. If it does not, create it. You can also customize the location of this directory from the Admin portlet:



Deploy Directory	/home/liferay/deploy
Destination Directory	../webapps
Interval	5 Seconds
Unpack WAR	✓
Tomcat Library Directory	../common/lib/ext

2. Start the server. The server will now automatically scan for *.war files.

3. Download one of the layout templates, sample portlets or themes to /home/liferay/deploy. Click here [http://www.liferay.com/web/guest/downloads/samples] for a list of sample portlets, themes and layouts. (Any JSR 168 compliant portlet WAR will work as well.)

4. Install either JBoss+Jetty, JBoss+Tomcat, Jetty, Resin, Tomcat or WebSphere.

5. If you have already set up the extension environment, you can hot deploy portlets, layouts or themes by dropping them into ext/portlets, ext/layouttpl or ext/themes respectively. Then run **ant deploy** from that directory.

   Notes: If the hot deploy feature does not work, make sure that your *.war file is a JSR 168 compliant portlet WAR, theme or layout. (You cannot use themes and layouts designed for Liferay Portal v3.6.1 or layouts and themes that were designed for other portals).

Chapter 7. Security and Permissions

Introduction

Fine grain permissioning is one of the main new features of Liferay Portal. Developers can now implement access security into their custom portlets, giving administrators and users a lot more control over their portlets and contents. This document will provide a reference for implementing this new security feature into their custom portlets. Developers should first read the Security and Permissions section of the Liferay User Guide before continuing with this document.

Overview

Adding fine grain permissioning to custom portlets consists of four main steps (also known as DRAC):

1. Define all resources and their permissions.
2. For all the resources defined in step 1, register them into the permission system. This is also known simply as “adding resources.”
3. Associate the necessary permissions to these resources.
4. Check permission before returning resources.

Implementing Permissions

In this section, each of the four main steps in adding Liferay’s security feature into custom portlets (built on top of the Liferay portal) will be explained. The following are two definitions that are important to remember.

> **Resource**: a generic term for any object represented in the portal. Example of resources includes portlets (e.g., Message Boards, Calendar, etc.), Java classes (e.g., Message Board Topics, Calendar Events, etc.), and files (e.g., documents, images, etc.). **Permission**: an action acting on a resource. For example, the view in “viewing the calendar portlet” is defined as a permission in Liferay.

Keep in mind that the permission for a portlet resource is implemented a little differently from the other resources such as Java classes and files. In each of the subsections below, the permission implementation for the portlet resource is explained first, then the model (and file) resource.

Defining Resources and Actions

For your custom portlet, Liferay portal needs to know whether there are resources that require permission and whether there are custom permissions. This information is encapsulated in an XML file found in the `portal/portal-ejb/classes/resource-actions` directory. If your portlet only needs the view and the configuration permission, and that the portlet doesn’t use any models with permission, then you do not need to create this XML file. The reason is that all portlets in Liferay automatically inherit these two permissions. However, if your portlet does have custom permission and/or uses models that have custom permissions, then you will need to create an XML file defining the resources and actions. Let’s take a look at `blogs.xml` in `portal/portal-ejb/classes/resource-actions` and see how the blogs portlet defined these resources and actions:

```
<?xml version="1.0"?>

<resource-action-mapping>
        <portlet-resource>
                <portlet-name>33</portlet-name>
                <supports>
                        <action-key>ADD_ENTRY</action-key>
                        <action-key>CONFIGURATION</action-key>
                        <action-key>VIEW</action-key>
                </supports>
                <community-defaults>
                        <action-key>VIEW</action-key>
                </community-defaults>
                <guest-defaults>
                        <action-key>VIEW</action-key>
                </guest-defaults>
                <guest-unsupported>
                        <action-key>ADD_ENTRY</action-key>
                </guest-unsupported>
        </portlet-resource>
        <model-resource>
<model-name>com.liferay.portlet.blogs.model.BlogsCategory</model-name>
                <portlet-ref>
                        <portlet-name>33</portlet-name>
                </portlet-ref>
                <supports>
                        <action-key>DELETE</action-key>
                        <action-key>PERMISSIONS</action-key>
                        <action-key>UPDATE</action-key>
                        <action-key>VIEW</action-key>
                </supports>
                <community-defaults>
                        <action-key>VIEW</action-key>
                </community-defaults>
                <guest-defaults>
                        <action-key>VIEW</action-key>
                </guest-defaults>
                <guest-unsupported>
                        <action-key>UPDATE</action-key>
                </guest-unsupported>
        </model-resource>
        <model-resource>
<model-name>com.liferay.portlet.blogs.model.BlogsEntry</model-name>
                <portlet-ref>
                        <portlet-name>33</portlet-name>
                </portlet-ref>
                <supports>
                        <action-key>ADD_COMMENT</action-key>
                        <action-key>DELETE</action-key>
                        <action-key>PERMISSIONS</action-key>
                        <action-key>UPDATE</action-key>
                        <action-key>VIEW</action-key>
                </supports>
                <community-defaults>
                        <action-key>VIEW</action-key>
                </community-defaults>
                <guest-defaults>
                        <action-key>VIEW</action-key>
                </guest-defaults>
                <guest-unsupported>
                        <action-key>UPDATE</action-key>
                </guest-unsupported>
        </model-resource>
</resource-action-mapping>
```

Portlet Resource

In the XML, the first thing defined is the portlet itself. Right under the root element `<resource-action-mapping>`, we have a child element called `<portlet-resource>`. In this element, we define the portlet name, which is 33 in our case. Next, we list all the actions this portlet supports under the `<supports>` tag. Keep in mind that this is at the portlet level. To understand what should be listed here, developers should ask themselves what actions belong to the portlet itself or what actions are performed on the portlet that may require a security check. In our case, users need permission to add an entry (`ADD_ENTRY`), configure blogs portlet settings (`CONFIGURATION`), and view the blogs itself (`VIEW`). Each of these supported permissions is within its own `<action-key>` tag. After we've defined all the actions that require a check, we move on to define some of the default permission settings. The `community-defaults` tag defines what actions are permitted by default for this portlet on the community (group) page the portlet resides. Put it another way, what should a user that has access to the community this portlet resides be able to do minimally? For the blogs portlet, a user with access to the community containing the blogs portlet should be able to view it. Likewise, the `guest-defaults` tag defines what actions are permitted by default to guests visiting a layout containing this portlet. So if a guest has access to the community page that contains a blogs portlet, the guest should, at the very least, be able to view the portlet according to `blogs.xml` (not necessarily the content of the portlet). Otherwise, the guest will see an error message within the portlet. Depending on your custom portlet, you may add more actions here that make sense. The `guest-unsupported` tag contains actions that a visiting guest should never be able to do. For example, the guest visiting the blogs portlet should never be able to add a blog entry since the blog belongs to either a user or a group of users. So even if a user wants to grant guests the ability to add a blog entry to her blog, there is no way for her to grant that permission because the `blogs.xml` doesn't permit such an action for guests.

Model Resource

After defining the portlet as a resource, we move on to define models within the portlet that also require access check. The model resource is surrounded by the `<model-resource>` tag. Within this tag, we first define the model name. This must be the fully qualified Java class name of the model. Next we define the portlet name that this model belongs to under the `portlet-ref` tag. Though unlikely, a model can belong to multiple portlets, which you may use multiple `<portlet-name>` tags to define. Similar to the portlet resource element, the model resource element also allows you to define a supported list of actions that require permission to perform. You must list out all the performable actions that require a permission check. As you can see for a blog entry, a user must have permission in order to add comments to an entry, delete an entry, change the permission setting of an entry, update an entry, or simply to view an entry. The `<community-defaults>` tag, the `<guest-defaults>` tag, and the `<guest-unsupported>` tag are all similar in meaning to what's explained for portlet resource in section 3.1.1.

Default.xml

After defining your permission scheme for your custom portlet, you then need to tell Liferay the location of this file. For Liferay core, the XML file would normally reside in `portal/portal-ejb/classes/resource-actions` and a reference to the file would appear in the `default.xml` file. For the EXT environment, the recommended setup is to put your XML file in `ext/ext-ejb/classes/resource-actions`. Create a file called `default-ext.xml` and model it after the `default.xml` file. Add all your custom resource-action XML files in the `default-ext.xml` file. Then copy the property resource.actions.configs found in `portal.properties` and paste it into `portal-ext.properties`. Lastly, add a comma to the end of the property value and then add the path to your `default-ext.xml` file. (i.e. `resource.actions.configs=resource-actions/default.xml,resource-actions/default-ext.xml`) Below is an example of the `default.xml` file.

```
<?xml version="1.0"?>

<resource-action-mapping>
        <resource file="resource-actions/portal.xml" />
        <resource file="resource-actions/blogs.xml" />
        <resource file="resource-actions/bookmarks.xml" />
        <resource file="resource-actions/calendar.xml" />
        <resource file="resource-actions/communities.xml" />
        <resource file="resource-actions/documentlibrary.xml" />
        <resource file="resource-actions/imagegallery.xml" />
        <resource file="resource-actions/journal.xml" />
        <resource file="resource-actions/messageboards.xml" />
        <resource file="resource-actions/polls.xml" />
        <resource file="resource-actions/shopping.xml" />
        <resource file="resource-actions/wiki.xml" />
</resource-action-mapping>
```

Adding Resources

After defining resources and actions, the next task is to write code that adds resources into the permissioning system. A lot of the logic to add resources is encapsulated in the `ResourceLocalServiceImpl` class. So adding resources is as easy as calling the add resource method in `ResourceLocalServiceUtil` class.

```
public void addResources(
        String companyId, String groupId, String userId, String name,
        String primKey, boolean portletActions,
        boolean addCommunityPermissions, boolean addGuestPermissions);
```

For all the Java objects that require access permission, you need to make sure that they are added as resources every time a new one is created. For example, every time a user adds a new entry to her blog, the `addResources`(…) method is called to add the new entry to the resource system. Here's an example of the call from the `BlogsEntryLocalServiceImpl` class.

```
ResourceLocalServiceUtil.addResources(
        entry.getCompanyId(), entry.getGroupId(), entry.getUserId(),
        BlogsEntry.class.getName(), entry.getPrimaryKey().toString(),
        false, addCommunityPermissions, addGuestPermissions);
```

The parameters `companyId`, `groupId`, and `userId` should be self explanatory. The `name` parameter is the fully qualified Java class name for the resource object being added. The `primKey` parameter is the primary key of the resource object. As for the `portletActions` parameter, set this to true if you're adding portlet action permissions. In our example, we set it to `false` because we're adding a model resource, which should be associated with permissions related to the model action defined in `blogs.xml`. The `addCommunityPermissions` and the `addGuestPermissions` parameters are inputs from the user. If set to true, `ResourceLocalService` will then add the default permissions to the current community group and the guest group for this resource respectively.

UI Interface

If you would like to provide your user the ability to choose whether to add the default community permission and the guest permission for the resources within your custom portlet, Liferay has a custom JSP tag you may use to quickly add that functionality. Simply insert the `<liferay-ui:input-permissions />` tag into the appropriate JSP and the checkboxes will show up on your JSP. Of course, make sure the tag is within the appropriate `<form>` tags.

Deleting Resources

To prevent having a lot of dead resources taking up space in the `Resource_` database table, you must remember to remove them from the `Resource_` table when the resource is no longer applicable. Simply call the `deleteResource(…)` method in `ResourceLocalServiceUtil`. Here's an example of a blogs entry being removed:

```
ResourceLocalServiceUtil.deleteResource(
        entry.getCompanyId(), BlogsEntry.class.getName(),
        Resource.TYPE_CLASS, Resource.SCOPE_INDIVIDUAL,
        entry.getPrimaryKey().toString());
```

Adding Permission

Portlet Permission

On the portlet level, no code needs to be written in order to have the permission system work for your custom portlet. Your custom portlet will automatically have all the permission features. If you've defined any custom permissions (supported actions) in your portlet-resource tag in section 3.1, those are automatically added to a list of permissions and users can readily choose them. Of course, for your custom permissions to have any value, you'll need to show or hide certain functionality in your portlet. You can do that by checking the permission first before performing the intended functionality. This will be covered in section 3.4.

Model Permission

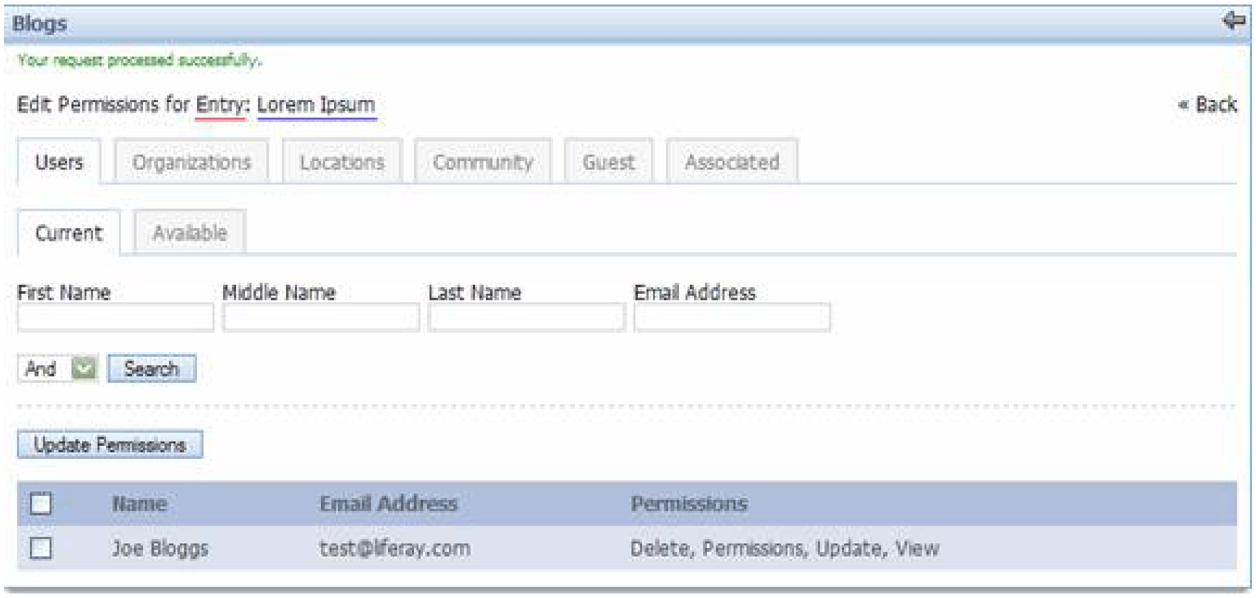
In order to allow a user to set permissions on the model resources, you will need to expose the permission interface to the user. This can be done by adding two Liferay UI tag to your JSP. The first one is the `<liferay-security:permissionsURL>` tag which returns a URL that takes the user to the page to configure the permission settings. The second tag is the `<liferay-ui:icon>` tag that shows a permission icon to the user. Below is an example found in the file `view_entry_content.jsp.`

```
<liferay-security:permissionsURL
        modelResource="<%= BlogsEntry.class.getName() %>"
        modelResourceDescription="<%= entry.getTitle() %>"
        resourcePrimKey="<%= entry.getPrimaryKey().toString() %>"
        var="entryURL"
/>

<liferay-ui:icon image="permissions" url="<%= entryURL %>" />
```

The attributes you need to provide to the first tag are `modelResource`, `modelResourceDescription`, `resourcePrimKey`, and var. The `modelResource` attribute is the fully qualified Java object class name. It then gets translated in `Language.properties` to a more readable name (underlined in red in figure 3.3.2.1).

```
model.resource.com.liferay.portlet.blogs.model.BlogsEntry=Entry
```


Blogs

Your request processed successfully.

Edit Permissions for Entry: Lorem Ipsum

« Back

	Name	Email Address	Permissions
☐	Joe Bloggs	test@liferay.com	Delete, Permissions, Update, View

As for the `modelResourceDescription` attribute, you can pass in anything that best describes this model instance. In the example, the blogs title was passed in, which is reflected in figure 3.3.2.1 with the blue underline. The `resourcePrimKey` attribute is simply the primary key of your model instance. The `var` attribute is the variable name this URL String will get assigned to. This variable is then passed to the `<liferay-ui:icon>` tag so the permission icon will have the proper URL link. There's also an optional attribute `redirect` that's available if you want to override the default behavior of the upper right arrow link shown in figure 3.3.2.1. That is all you need to do to enable users to configure the permission settings for model resources!!

Checking Permissions

The last major step to implementing permission to your custom portlet is to check permission. This may be done in a couple of places. For example, your business layer should check for permission before deleting a resource, or your user interface should hide a button that adds a model (e.g., a calendar event) if the user does not have permission to do so.

Checking Portlet Resource Permission

Similar to the other steps, the default permissions for the portlet resources are automatically checked for you. You do not need to implement anything for your portlet to discriminate whether a user is allowed to view or to configure the portlet itself. However, you do need to implement any custom permission you have defined in your resource-actions XML file. In the blogs portlet example, one custom supported action is `ADD_ENTRY`. There are two places in the source code that check for this permission. The first one is in the file `view_entries.jsp`. The presence of the add entry button is contingent on whether the user has permission to add entry (and also whether the user is in tab one).

```
<%
boolean showAddEntryButton = tabs1.equals("entries") &&
PortletPermission.contains(permissionChecker, plid, PortletKeys.BLOGS,
ActionKeys.ADD_ENTRY);
%>
```

The second place that checks for the add entry permission is in the file `BlogsEntryServiceImpl`. (Notice the difference between this file and the `BlogsEntryLocalServiceImpl`.) In the `addEntry(…)` method, a call is made to check whether the incoming request has permission to add entry.

```
PortletPermission.check(
        getPermissionChecker(), plid, PortletKeys.BLOGS,
        ActionKeys.ADD_ENTRY);
```

If the check fails, it throws a `PrincipalException` and the add entry request aborts. You're probably wondering what the PortletPermission class and the PermissionChecker class do. Let's take a look at these two classes.

PermissionChecker

The PermissionChecker class has a method called `hasPermission(…)` that checks whether a user making a resource request has the necessary access permission. If the user is not signed in (guest user), it checks for guest permissions. Otherwise, it checks for user permissions. This class is available to you in two places. First in your business logic layer, you can obtain an instance of the PermissionChecker by calling the `getPermissionChecker()` method inside your `ServiceImpl` class. This method is available because all `ServiceImpl` (not `LocalServiceImpl`) extends the `PrincipalBean` class, which implements the `getPermissionChecker()` method. The other place where you can obtain an instance of the PermissionChecker class is in your JSP files. If your JSP file contains the portlet tag `<portlet:defineObjects />` or includes another JSP file that does, you'll have an instance of the PermissionChecker class available to you via the `permissionChecker` variable. Now that you know what the `PermissionChecker` does and how to obtain an instance of it, let's take a look at Liferay's convention in using it.

PortletPermission

`PortletPermission` is a helper class that makes it easy for you to check permission on portlet resources (as oppose to model resources, covered later in section 3.4.2). It has two static methods called `check(…)` and another two called `contains(…)`. They are all essentially the same. The two differences between them are:

1. Only one `check(…)` method and one `contains(…)` method take in the portlet layout ID variable (`plid`).
2. The `check(…)` methods throw a new `PrincipalException` if user does not have permission, and the `contains(…)` methods return a boolean indicating whether user has permission.

The `contains(…)` methods are meant to be used in your JSP files since they return a boolean instead of throwing an exception. The check(…) methods are meant to be called in your business layer (ServiceImpl). Let's revisit the blogs portlet example below. (The addEntry(…) method is found in BlogsEntryServiceImpl.)

```
public BlogsEntry addEntry(
                String plid, String categoryId, String[] tags, String title,
                String content, int displayDateMonth, int displayDateDay,
                int displayDateYear, int displayDateHour, int
displayDateMinute,
                boolean addCommunityPermissions, boolean addGuestPermissions)
throws PortalException, SystemException {

        PortletPermission.check(
                getPermissionChecker(), plid, PortletKeys.BLOGS,
                ActionKeys.ADD_ENTRY);

        return BlogsEntryLocalServiceUtil.addEntry(
                getUserId(), plid, categoryId, tags, title, content,
                displayDateMonth, displayDateDay, displayDateYear,
displayDateHour,
                displayDateMinute, addCommunityPermissions,
addGuestPermissions);
}
```

Before the `addEntry(…)` method calls `BlogsEntryLocalServiceUtil.addEntry(…)` to add a blogs entry, it calls `PortletPermission.check(…)` to validate user permission. If the check fails, a `PrincipalException` is thrown and an entry will not be added. Note the parameters passed into the method. Again, the `getPermissionChecker()` method is readily available in all `ServiceImpl` classes. The `plid` variable is passed into the method by its caller (most likely from a `PortletAction` class). `PortletKeys.BLOGS` is just a static `String` indicating that the permission check is against the blogs portlet. `ActionKeys.ADD_ENTRY` is also a static String to indicate the action requiring the permission check. You're encouraged to do likewise with your custom portlet names and custom action keys.

Whether you need to pass in a portlet layout ID (plid) depends on whether your custom portlet supports multiple instances. Let's take a look at the message board portlet for example. A community may need three separate page layouts, each having a separate instance of the message board portlet. Only by using the portlet layout ID will the permission system be able to distinguish the three separate instances of the message board portlet. This way, permission can be assigned separately in all three instances. Though in general, most portlets won't need to use the portlet layout ID in relation to the permission system.

Service vs. LocalService

Since the `ServiceImpl` class extends the `PrincipalBean` class, it has access to information of the current user making the service request. Therefore, the `ServiceImpl` class is the ideal place in your business layer to check user permission. Liferay's convention is to implement the actual business logic inside the `LocalServiceImpl` methods, and then the `ServiceImpl` calls these methods via the `LocalServiceUtil` class after the permission check completes successfully. Your `PortletAction` classes should make calls to `ServiceUtil` (wrapper to `ServiceImpl`) guaranteeing that permission is first checked before the request is fulfilled.

Checking Model Resource Permission

Checking model resource permission is very similar to checking portlet resource permission. The only major difference is that instead of calling methods found in the PortletPermission class mention previously, you need to create your own helper class to assist you in checking permission. The next section will detail how this is done.

Custom Permission Class

It is advisable to have a helper class to help check permission on your custom models. This custom permission class is similar to the PortletPermission class but is tailored to work with your custom models. While you can implement this class however you like, we encourage you to model after the PortletPermission class, which contains four static methods. Let's take a look at the `BlogsEntryPermission` class.

```
public class BlogsEntryPermission {

        public static void check(
                        PermissionChecker permissionChecker, String entryId,
                        String actionId)
                throws PortalException, SystemException {

                if (!contains(permissionChecker, entryId, actionId)) {
                        throw new PrincipalException();
                }
        }

        public static void check(
                        PermissionChecker permissionChecker, BlogsEntry entry,
                        String actionId)
                throws PortalException, SystemException {

                if (!contains(permissionChecker, entry, actionId)) {
                        throw new PrincipalException();
                }
        }

        public static boolean contains(
                        PermissionChecker permissionChecker, String entryId,
                String actionId)
            throws PortalException, SystemException {

            BlogsEntry entry = 
BlogsEntryLocalServiceUtil.getEntry(entryId);

            return contains(permissionChecker, entry, actionId);
        }

        public static boolean contains(
                PermissionChecker permissionChecker, BlogsEntry entry,
                String actionId)
            throws PortalException, SystemException {

            return permissionChecker.hasPermission(
                entry.getGroupId(), BlogsEntry.class.getName(),
                entry.getPrimaryKey().toString(), actionId);
        }
}
```

Again, the two check(…) methods are meant to be called in your business layer, while the two contains(…) methods can be used in your JSP files. As you can see, it's very similar to the PortletPermission class. The two notable differences are:

1. Instead of having the `portletId` as one of the parameters, the methods in this custom class take in either an `entryId` or a `BlogsEntry` object.
2. None of the methods need to receive the portlet layout ID (`plid`) as a parameter. (Your custom portlet may choose to use the portlet layout ID if need be.)

Let's see how this class is used in the blogs portlet code.

```
public BlogsEntry getEntry(String entryId) throws PortalException, 
SystemException {

        BlogsEntryPermission.check(
            getPermissionChecker(), entryId, ActionKeys.VIEW);

        return BlogsEntryLocalServiceUtil.getEntry(entryId);
}
```

In the `BlogsEntryServiceImpl` class is a method called `getEntry(…)`. Before this method returns the blogs entry object, it calls the custom permission helper class to check permission. If this call doesn't throw an exception, the entry is retrieved and returned to its caller.

```
<c:if test="<%= BlogsEntryPermission.contains(permissionChecker, entry, 
ActionKeys.UPDATE) %>">
        <portlet:renderURL windowState="<%= WindowState.MAXIMIZED.toString() 
%>" var="entryURL">
                <portlet:param name="struts_action" value="/blogs/edit_entry" 
/>
                <portlet:param name="redirect" value="<%= currentURL %>" />
                <portlet:param name="entryId" value="<%= entry.getEntryId() 
%>" />
        </portlet:renderURL>

        <liferay-ui:icon image="edit" url="<%= entryURL %>" />
</c:if>
```

In the `view_entry_content.jsp` file, the `BlogsEntryPermission.contains(…)` method is called to check whether or not to show the edit button. That's all there is to it!

Summary

Let's review what we've just covered. Implementing permission into your custom portlet consists of four main steps. First step is to define any custom resources and actions. Next step is to implement code to register (or add) any newly created resources such as a `BlogsEntry` object. The third step is to provide an interface for the user to configure permission. Lastly, implement code to check permission before returning resources or showing custom features. Two major resources are portlets and Java objects. There is not a lot that needs to be done for the portlet resource to implement the permission system since Liferay Portal has a lot of that work done for you. You mainly focus your efforts on any custom Java objects you've built. You're now well on your way to implement security to your custom Liferay portlets! For other user guides, please visit the Liferay Developer Zone at: http://www.liferay.com/web/guest/devzone

Additional Information

Roles

If you're wondering how the Liferay user roles, community roles, and organization / location roles should be implemented in your custom portlet, this brief section will address that. The short answer is, nothing needs to be developed specifically for user, community, and organization / location roles to work with your custom portlets. Liferay's permission system has all that provided for you. When the `hasUserPermission(…)` method is called within the `PermissionChecker` class, Liferay checks all the roles the current user has, whether they're organization / location roles, community roles, or user roles.

Using your own Security System in Liferay

Here's a brief outline of how you can use your own security system in Liferay.

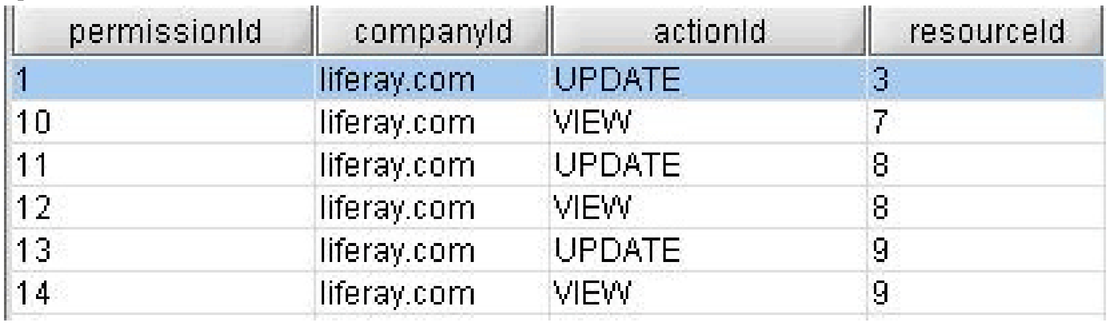
- Create your own `PermissionChecker` class that extends Liferay's `PermissionChecker` class.
- Register this new class in `portal.properties` (or `portal-ext.properties` for the EXT environment) under the `permissions.checker` property.
- Override the `hasUserPermission(…)` method and the `hasGuestPermission(…)` method with your own calls to your permission system.
- You can call the `setValues(…)` method to pull in parameters from the request object that your permission checker might need (e.g., `userId, projected`, etc).
- You can call the `resetValues(…)` method to wipe out old parameters.
- Override the `isAdmin(…)` method.

Database Schema View

Reviewing how Liferay stores all the permission information in the database may help you gain a better understanding to the entire permission system.

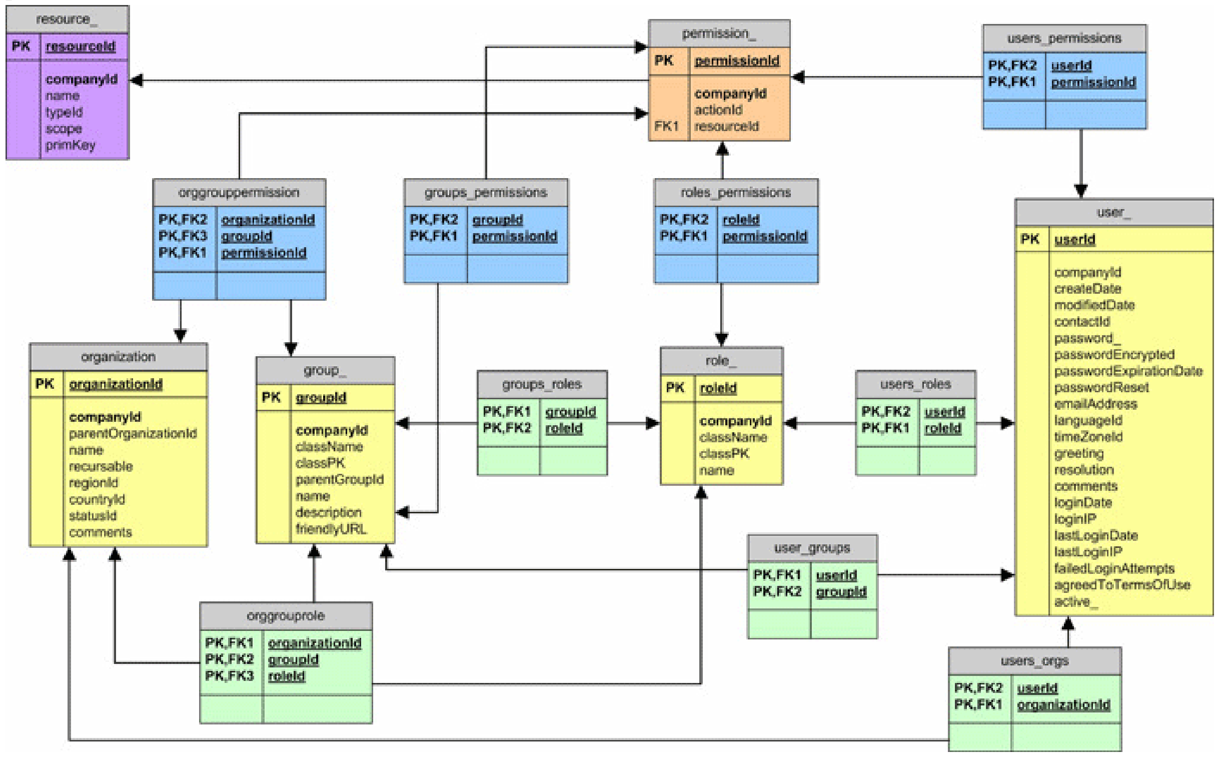
- The `resource_` table contains all the registered resources outlined in section 3.2.

- Every possible secure action that can be done to a resource will result in a row in the `permission_` table. For example, a `BlogsEntry` resource may have a row in `permission_` for the view action, and another for the update action.



permissionId	companyId	actionId	resourceId
1	liferay.com	UPDATE	3
10	liferay.com	VIEW	7
11	liferay.com	UPDATE	8
12	liferay.com	VIEW	8
13	liferay.com	UPDATE	9
14	liferay.com	VIEW	9

- Whether a user has permission to a resource depends on the roles the user has, or the community (groups) and organization the user is in (green tables). If those roles or groups contain the needed `permissionId` in the permissions table (in blue), then the user has access to the resource.



Chapter 8. Tutorials

Portlets

Liferay Portal Enterprise builds upon the Portlet API [http://www.jcp.org/en/jsr/detail?id=168] (JSR 168) to provide users with a rich set of portlets. The following are examples from the portlets bundled with Liferay. Follow the instructions found in Hot Deploy to learn how to hot deploy external WARs that are not bundled with Liferay.

Hello World

1. This portlet is defined in /portal-web/docroot/WEB-INF/portlet.xml.

```
        <portlet>
            <portlet-name>47</portlet-name>
            <display-name>Hello World</display-name>
            <portlet-class>
               com.liferay.portlet.helloworld.HelloWorldPortlet
            </portlet-class>
            <expiration-cache>0</expiration-cache>
            <supports>
                <mime-type>text/html</mime-type>
            </supports>
            <portlet-info>
                <title>Hello World</title>
                <short-title>Hello World</short-title>
                <keywords>Hello World</keywords>
            </portlet-info>
            <security-role-ref>
                <role-name>Power User</role-name>
            </security-role-ref>
            <security-role-ref>
                <role-name>User</role-name>
            </security-role-ref>
        </portlet>
```

   The unique id associated with this portlet is **47**. The HelloWorldPortlet class extends javax.portlet.GenericPortlet. The source of this class shows that it does nothing more than print out Hello World. This portlet is viewable by HTML browsers. The title is defined in **portlet-info**. Users must have either the Power User or User role to access this portlet. The roles can be changed at run time via the Admin portlet.

2. Additional definitions for this portlet are found in /portal-web/docroot/WEB-INF/liferay-portlet.xml.

```
<portlet id="47" struts-path="hello_world" narrow="true" />
```

   The **id** value in liferay-portlet.xml must match the **portlet-name** value in portlet.xml. The **struts-path** value tells Struts that all requests starting with http://localhost/c/hello_world/* are considered part of this portlet's scope. See the Mail portlet to better understand this feature. The **narrow** value, if set to true, means this portlet will be rendered in the narrow column. This value can be changed at run time via the Admin portlet.

3. Display information for this portlet is found in /portal-web/docroot/WEB-INF/liferay-display.xml and makes it possible for users to add this portlet via the personalize pages screen.

```
<category name="category.test">
    <portlet id="47" />
    <portlet id="48" />
</category>
```

When a user goes to personalize pages and clicks on a category to choose a portlet, the Hello World portlet is available under the category with the name that matches the key category.test. The value for this key is defined in /portal-ejb/classes/content/Language.properties.

```
category.test=Test
```

IFrame

1. This portlet is defined in /portal-web/docroot/WEB-INF/portlet.xml.

```
         <portlet>
             <portlet-name>48</portlet-name>
             <display-name>IFrame</display-name>
             <portlet-class>com.liferay.portlet.IFramePortlet</portlet-class>
             <expiration-cache>0</expiration-cache>
             <supports>
                 <mime-type>text/html</mime-type>
                 <portlet-mode>edit</portlet-mode>
             </supports>
<resource-bundle>com.liferay.portlet.StrutsResourceBundle</resource-bundle>
             <portlet-preferences>
                 <preference>
                     <name>src</name>
                     <value>http://www.gfa.org</value>
                 </preference>
                 <preference>
                     <name>auth</name>
                     <value>false</value>
                 </preference>
                 <preference>
                     <name>auth-type</name>
                     <value>basic</value>
                 </preference>
                 <preference>
                     <name>form-method</name>
                     <value>post</value>
                 </preference>
                 <preference>
                     <name>user-name</name>
                     <value></value>
                 </preference>
                 <preference>
                     <name>password</name>
                     <value></value>
                 </preference>
                 <preference>
                     <name>hidden-variables</name>
                     <value>var1=hello;var2=world</value>
                 </preference>
             </portlet-preferences>
             <security-role-ref>
                 <role-name>Power User</role-name>
```

```
            </security-role-ref>
            <security-role-ref>
                <role-name>User</role-name>
            </security-role-ref>
        </portlet>
```

The unique id associated with this portlet is **48**. The IFramePortlet class extends javax.portlet.GenericPortlet. The source [http://content.liferay.com/document/api/portal-ejb/com/liferay/portlet/IFramePortlet.java.html] of this class shows that this class prints an IFRAME tag that references an external site. This portlet is **editable** and viewable by HTML browsers. The preferences bind the name src with the default value of **ht-tp://www.gfa.org**.

The **auth** value, if set to true, will attempt to authenticate the user to the external IFrame application. The **auth-type** value can be set to basic or form. Basic authentication appends the login information to the URL and form authentication requires a post to the external IFrame application. The **form-method** value can be set to get or post. This is only used if you are using form authentication. The **user-name** value sets the user name for authentication. If you are using basic authentication, then you just need to set the user name. If using form authentication, you need to set the user name as a key value pair like acme_login=test@acme.com. The **password** value sets the password for authentication. If using basic authentication, you just need the password. If using form authentication, set the password as a key value pair like acme_password=password. The **hidden-variables** value is used for form authentication. Some forms require certain prepopulated fields in order to proceed with authentication. Separate each key and value with a = and each key value pair with a ;. Users must have either the *Power User* or *User* role to access this portlet. The roles can be changed at run time via the Admin portlet.

2. The title is fetched by StrutsResourceBundle and is configured in / portal-ejb/classes/content/Language.properties.

   ```
   javax.portlet.title.48=IFrame
   ```

3. Additional definitions for this portlet are found in /portal-web/docroot/WEB-INF/liferay-portlet.xml.

   ```
   <portlet id="48" struts-path="iframe" />
   ```

   The **id** value in liferay-portlet.xml must match the **portlet-name** value in portlet.xml. The **struts-path** value tells Struts that all requests starting with http://localhost/c/iframe/* are considered part of this portlet's scope. See the Mail portlet to better understand this feature.

4. Display information for this portlet is found in /portal-web/docroot/WEB-INF/liferay-display.xml and makes it possible for users to add this portlet via the personalize pages screen.

   ```
         <category name="category.test">
             <portlet id="47" />
             <portlet id="48" />
         </category>
   ```

   When a user goes to personalize pages and clicks on a category to choose a portlet, the IFrame portlet is available under the category with the name that matches the key category.test. The value for this key is defined in / portal-ejb/classes/content/Language.properties.

   ```
   category.test=Test
   ```

Calendar

1. This portlet is defined in /portal-web/docroot/WEB-INF/portlet.xml.

```
        <portlet>
            <portlet-name>8</portlet-name>
            <display-name>Calendar</display-name>
            <portlet-class>com.liferay.portlet.JSPPortlet</portlet-class>
            <init-param>
                <name>view-jsp</name>
                <value>/portlet/calendar/view.jsp</value>
            </init-param>
            <expiration-cache>0</expiration-cache>
            <supports>
                <mime-type>text/html</mime-type>
            </supports>
<resource-bundle>com.liferay.portlet.StrutsResourceBundle</resource-bundle>
            <security-role-ref>
                <role-name>Power User</role-name>
            </security-role-ref>
            <security-role-ref>
                <role-name>User</role-name>
            </security-role-ref>
        </portlet>
```

   The unique id associated with this portlet is 8. The JSPPortlet class extends javax.portlet.GenericPortlet. The source of this class shows that this class looks for the init parameters and forwards to the appropriate JSP depending on the portlet mode. This portlet is viewable by HTML browsers. Users must have either the Power User or User role to access this portlet. The roles can be changed at run time via the Admin portlet.

2. The title is fetched by StrutsResourceBundle and is configured in /portal-ejb/classes/content/Language.properties.

```
javax.portlet.title.8=Calendar
```

3. Additional definitions for this portlet are found in /portal-web/docroot/WEB-INF/liferay-portlet.xml.

```
        <portlet
            id="8"
            struts-path="calendar"
            scheduler-class="com.liferay.portlet.calendar.job.Scheduler"
            preferences-sharing-type="user"
            show-portlet-access-denied="true"
            show-portlet-inactive="true"
        />
```

   The id value in liferay-portlet.xml must match the portlet-name value in portlet.xml. The struts-path value tells Struts that all requests starting with http://localhost/c/calendar/* are considered part of this portlet's scope. See the Mail portlet to better understand this feature. The scheduler-class value must be a class that implements Scheduler and is called to schedule Quartz jobs for this portlet. The preferences-sharing-type value indicates the preferences sharing type of the portlet. If set to none, preferences are not shared and each page will have its own set of preferences. If set to user, preferences are shared by user if the portlet is in a personal page or by group if the portlet is in a group page. If set to company, preferences are shared across the entire company. The show-portlet-access-denied value, if set to true, means non-permissioned users are shown that they do not have access to the portlet. The default value is set in portal.properties. The show-portlet-inactive value, if set to true, means users are shown that the portlet is inactive (if the portlet is inactive). The default value is set in portal.properties.

4. Display information for this portlet is found in /portal-web/docroot/WEB-INF/liferay-display.xml and makes it possible for users to add this portlet via the personalize pages screen.

```
      <category name="category.community">
          ...
          <portlet id="8" />
          ...
      </category>
```

   When a user goes to personalize pages and clicks on a category to choose a portlet, the Calendar portlet is available under the category with the name that matches the key category.community. The value for this key is defined in /portal-ejb/classes/content/Language.properties.

```
category.community=Community
```

Message Boards

1. This portlet is defined in /portal-web/docroot/WEB-INF/portlet.xml.

```
          <portlet>
              <portlet-name>19</portlet-name>
              <display-name>Message Boards</display-name>
<portlet-class>com.liferay.portlet.messageboards.MBPortlet</portlet-class>
              <init-param>
                  <name>edit-jsp</name>
                  <value>/portlet/message_boards/edit.jsp</value>
              </init-param>
              <init-param>
                  <name>view-jsp</name>
                  <value>/portlet/message_boards/view.jsp</value>
              </init-param>
              <expiration-cache>0</expiration-cache>
              <supports>
                  <mime-type>text/html</mime-type>
                  <portlet-mode>edit</portlet-mode>
              </supports>
<resource-bundle>com.liferay.portlet.StrutsResourceBundle</resource-bundle>
              <portlet-preferences>
                  <preference>
                      <name>messages-per-page</name>
                      <value>25</value>
                  </preference>
              </portlet-preferences>
              <security-role-ref>
                  <role-name>Power User</role-name>
              </security-role-ref>
              <security-role-ref>
                  <role-name>User</role-name>
              </security-role-ref>
          </portlet>
```

   The unique id associated with this portlet is **19**. The MBPortlet class extends JSPPortlet. This portlet is editable and viewable by HTML browsers. The preferences bind the name **messages-per-page** with the default value of 25. Users must have either the Power User or User role to access this portlet. The roles can be changed at run time via the Admin portlet.

2. The title is fetched by StrutsResourceBundle and is configured in /portal-ejb/classes/content/Language.properties.

```
javax.portlet.title.19=Message Boards
```

3. Additional definitions for this portlet are found in /portal-web/docroot/WEB-INF/liferay-portlet.xml.

```
<portlet id="19" struts-path="message_boards"
indexer-class="com.liferay.portlet.messageboards.util.Indexer"
preferences-sharing-type="user" />
```

The **id** value in liferay-portlet.xml must match the **portlet-name** value in portlet.xml. The **struts-path** value tells Struts that all requests starting with http://localhost/c/message_boards/* are considered part of this portlet's scope. See the Mail portlet to better understand this feature. The **indexer-class** value must be a class that implements Indexer and is called to create or update a search index for this portlet. The **preferences-sharing-type** value indicates the preferences sharing type of the portlet. If set to none, preferences are not shared and each page will have its own set of preferences. If set to user, preferences are shared by user if the portlet is in a personal page or by group if the portlet is in a group page. If set to company, preferences are shared across the entire company.

4. Display information for this portlet is found in /portal-web/docroot/WEB-INF/liferay-display.xml and makes it possible for users to add this portlet via the personalize pages screen.

```
      <category name="category.community">
          ...
          <portlet id="19" />
          ...
      </category>
```

When a user goes to personalize pages and clicks on a category to choose a portlet, the Message Boards portlet is available under the category with the name that matches the key category.community. The value for this key is defined in /portal-ejb/classes/content/Language.properties.

```
category.community=Community
```

Mail

1. This portlet is defined in /portal-web/docroot/WEB-INF/portlet.xml.

```
      <portlet>
          <portlet-name>1</portlet-name>
          <display-name>Mail</display-name>
<portlet-class>com.liferay.portlet.mail.MailPortlet</portlet-class>
          <init-param>
              <name>view-jsp</name>
              <value>/portlet/mail/view.jsp</value>
          </init-param>
          <expiration-cache>0</expiration-cache>
          <supports>
              <mime-type>text/html</mime-type>
              <portlet-mode>edit</portlet-mode>
          </supports>
```

```
<resource-bundle>com.liferay.portlet.StrutsResourceBundle</resource-bundle>
          <portlet-preferences>
              <preference>
                  <name>html-format</name>
                  <value>true</value>
              </preference>
              <preference>
                  <name>forward-address</name>
              </preference>
              <preference>
                  <name>signature</name>
              </preference>
              <preference>
                  <name>col-order</name>
                  <value>fsdz</value>
              </preference>
              <preference>
                  <name>order-by-col</name>
                  <value>d</value>
              </preference>
              <preference>
                  <name>order-by-type</name>
                  <value>desc</value>
              </preference>
              <preference>
                  <name>messages-per-portlet</name>
                  <value>5</value>
              </preference>
              <preference>
                  <name>messages-per-page</name>
                  <value>25</value>
              </preference>
              <preference>
                  <name>message-headers</name>
                  <value>1</value>
              </preference>
              <preference>
                  <name>include-original</name>
                  <value>true</value>
              </preference>
              <preference>
                  <name>original-text-indicator</name>
                  <value>0</value>
              </preference>
              <preference>
                  <name>reply-to-address</name>
              </preference>
              <preference>
                  <name>new-mail-notification</name>
                  <value>2</value>
              </preference>
              <preference>
                  <name>folder-names</name>
              </preference>
              <preference>
                  <name>blocked</name>
              </preference>
          </portlet-preferences>
          <security-role-ref>
              <role-name>Power User</role-name>
          </security-role-ref>
          <security-role-ref>
```

```
            <role-name>User</role-name>
        </security-role-ref>
    </portlet>
```

The unique id associated with this portlet is **1**. The MailPortlet class extends JSPPortlet. This portlet is editable and viewable by HTML browsers. Users must have either the Power User or User role to access this portlet. The roles can be changed at run time via the Admin portlet.

2. The title is fetched by StrutsResourceBundle and is configured in /portal-ejb/classes/content/Language.properties.

```
javax.portlet.title.1=Mail
```

3. Additional definitions for this portlet are found in /portal-web/docroot/WEB-INF/liferay-portlet.xml.

```
<portlet id="1" struts-path="mail" preferences-sharing-type="user" />
```

The **id** value in liferay-portlet.xml must match the **portlet-name** value in portlet.xml. The **struts-path** value tells Struts that all requests starting with http://localhost/c/mail/* are considered part of this portlet's scope. If a user's request path falls inside the defined struts-path, then all of the portlet's objects will be available inside the forwarded Action classes and JSP pages as defined in /portal-web/docroot/WEB-INF/struts-config.xml and /portal-web/docroot/WEB-INF/tiles-defs.xml. For example, if the user requests ht-tp://localhost/c/mail/view_folder, then the user is forwarded to ViewFolderAction. This class has access to the javax.portlet.PortletConfig, javax.portlet.RenderRequest, and javax.portlet.RenderResponse objects through the user's request. These objects are stored as request attributes with the class names as keys. The JSPs that Struts forward to have access to these objects via the <portlet:defineObjects /> tag. User are not allowed to access any paths that fall in http://localhost/c/mail/* unless the have the required roles to access this portlet.

4. Display information for this portlet is found in **/portal-web/docroot/WEB-INF/liferay-display.xml** and makes it possible for users to add this portlet via the personalize pages screen.

```
    <category name="category.community">
        ...
        <portlet id="1" />
        ...
    </category>
```

When a user goes to personalize pages and clicks on a category to choose a portlet, the Mail portlet is available under the category with the name that matches the key **category.community**. The value for this key is defined in /portal-ejb/classes/content/Language.properties.

```
category.community=Community
```

Chapter 9. Code Conventions

We are proud and passionate about **super clean** code. Wrap lines at 80 spaces and use real tabs equivalent to 4 spaces. Follow the Java Code Conventions and read *The Elements of Java Style*. And **ABOVE** all else, follow the precedence of the rest of the code.