

Table of Contents

Liferay Presentation	1
Liferay Presentation	4
Slide 1	13
03-installing-liferay-bundle	19
Liferay Presentation	25
Slide 1	30
Liferay Training Template 3.0	38
Slide 1	52
Slide 1	65
Slide 1	74
Slide 1	80
Slide 1	85
Slide 1	92
Slide 1	115
03-ipc-exercise	127
Slide 1	148
01-library-project-overview	159
02-library-project-service-layer	171
03-library-project-controller-overview	189
04-library-project-portlet-actions	204
05-library-project-feedback-validation-localization	216
06-library-project-permissions	228
07-control-panel-portlets	243
Slide 1	248
Slide 1	262
03-layout-templates-overview-and-exercise	269
05-advanced-theme-and-layout-template-development	276
06-best-practices	283
01-hooks-overview-and-exercise	286
Slide 1	299
Slide 1	307
03-overriding-struts-actions	324
Liferay Training Template 3.0	330
Slide 1	338
Slide 1	344
03-dev-strategy	350



COURSE TOPICS

Copyright © 2000 – 2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

DAY 1

Introduction to Liferay

- Liferay's user interface
- Introduction to Plugins
- Development Strategy
- Installing Liferay Developer Studio

Development

- Introduction to Portlets
- Java Standard Portlets
- Inter-Portlet Communication



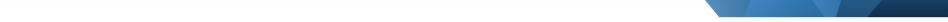
DAY 2

Developing Portlet Plugins

- ❖ Portlets and Web Application Frameworks
- ❖ MVC Portlets
 - Design Approach
 - Service Builder
 - The MVC Portlet Framework
 - Portlet Actions
 - Feedback, Validation, and Localization
 - Portlet Permissions
 - Control Panel Portlets

WWW.LIFERAY.COM





DAY 3

Developing Theme and Layout Template Plugins

- ❖ Liferay Themes Overview & Exercise
- ❖ Liferay Layout Template Overview & Exercise
- ❖ Advanced Themes, Layout Topics, and Best Practices

Developing Hook Plugins

- ❖ Customizing Properties
- ❖ Creating a cookie after logging in
- ❖ Customizing language keys
- ❖ Customizing core Liferay JSPs
- ❖ Overriding Terms of Use

WWW.LIFERAY.COM





DAY 3 CONTINUED

Developing EXT Plugins

- ❖ EXT Plugin Overview
- ❖ Extending user management
- ❖ Overriding Struts Actions

Advanced Topics

- ❖ Spring in Liferay
- ❖ Using Liferay's Web Services
- ❖ Summary and Conclusion



Q&A

- ❖ We'll have 10 minutes for Q&A after each section
This is a good time to ask questions about recently-covered topics or how covered topics relate to each other.
- ❖ We'll also have 30 – 60 minutes for Q&A at the end of each day
This is a good time to ask questions that are not covered in the course topics



BUILDING A PORTAL WITH LIFERAY

Copyright © 2000 – 2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated,
copied, sold, resold, or otherwise exploited for any commercial
purpose
without express written consent of Liferay, Inc.

WHAT IS A PORTAL?

- ❖ A portal is a web-based gateway which allows users to locate and create relevant content and use the applications they commonly need to be productive.
- ❖ Portals offer compelling benefits to today's enterprises: reduced operational costs, improved customer satisfaction, and streamlined business processes.

WHAT IS LIFERAY PORTAL?

- ❖ Liferay Portal is one of the most mature portal frameworks on the market
- ❖ Liferay Portal is open source and fully customizable
- ❖ Liferay Portal was the first to provide an easy-to-use, drag-and-drop interface
- ❖ Liferay Portal is server-agnostic, and is designed to fit into your environment

WWW.LIFERAY.COM



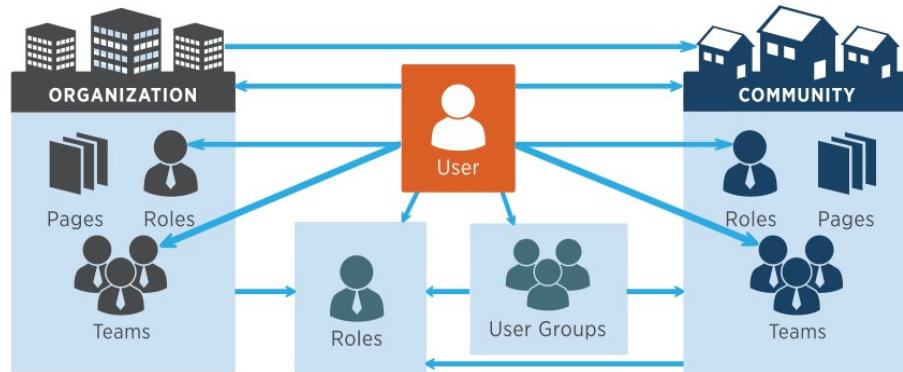
PORTAL ARCHITECTURE

- ❖ Portals are accessed by *Users*
- ❖ *Users* can be collected into *User Groups*
- ❖ *Users* can belong to *Organizations*
- ❖ *Organizations* can be grouped into hierarchies, such as Home Office
-> Regional Office -> Satellite Office
- ❖ *Users*, *Groups*, and *Organizations* can belong to *Communities* having a common interest

WWW.LIFERAY.COM



LIFERAY PORTAL MEMBERSHIP



WWW.LIFERAY.COM

 LIFERAY.

ORGANIZATIONS

- ❖ Organizations are hierarchical collections of Users
- ❖ Organizations can have portal pages
- ❖ Users can be members of multiple organizations
- ❖ Organizations can have *Roles*, such as Security, granting members of that organization particular permissions
- ❖ Organizations can also be members of Communities

WWW.LIFERAY.COM

 LIFERAY.

COMMUNITIES

- ❖ Communities are collections of Users who have a common interest
- ❖ Liferay's default pages are in the liferay.com community, because everyone—whether they are anonymous or members of the portal—has a common interest in the public portal pages
- ❖ Communities can be Open, Restricted, or Private

COMMUNITIES

- ❖ An Open community allows portal users to join and leave that community at will
- ❖ A Restricted community requires that users be added by a community administrator
- ❖ A Private community does not show up at all in the Communities portlet

USER GROUPS

- ❖ *User Groups* are simple, arbitrary collections of users
- ❖ They can be members of *Communities* or *Roles*
- ❖ User Groups can have Page Templates, allowing certain users access to certain applications automatically.

USERS

- ❖ Users can be collected in multiple ways:
 - Users can be members of *Organization* hierarchies, such as Liferay, Inc → Security
 - Users can be collected into arbitrary *User Groups*, such as **Writers**.
 - Users can be members of *Communities* which draw together common interests
 - Users can have *Roles* which describe their function in the system, and these roles can be scoped by Portal, Organization, or Community

ROLES

- ❖ There are three kinds of *Roles*:
 - Portal Roles
 - Organization Roles
 - Community Roles
- ❖ Users, User Groups, Communities, or Organizations can be members of a Role
- ❖ Roles are used to define permissions across their scope (portal, organization, or community)

WWW.LIFERAY.COM



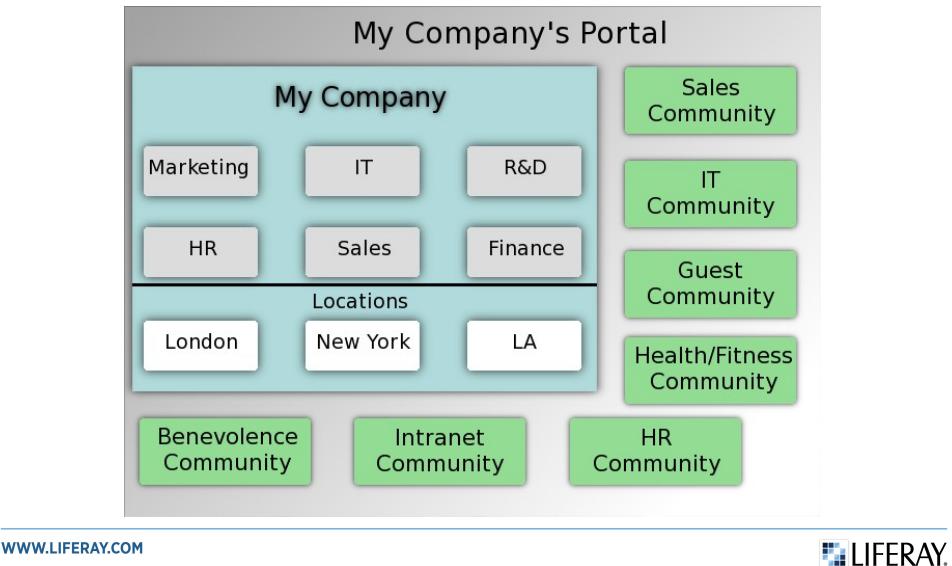
ROLES

- ❖ For example, consider a Role granting access to create a Message Board category
- ❖ A Portal role would grant that access across the portal, wherever there was a Message Board portlet
- ❖ A Community role would grant that access only within a community
- ❖ An Organization role would grant that access only within an organization

WWW.LIFERAY.COM



EXAMPLE PORTAL DESIGN



EXAMPLE PORTAL DESIGN EXPLAINED

- ❖ In the diagram, the outside *My Company's Portal* box represents the portal as a whole.
- ❖ The *My Company* box inside that represents the top level organization.
- ❖ Each box inside of *My Company* represents a suborganization of My Company.
- ❖ Communities don't have a formal structure, they can be configured so that anyone can access them.
- ❖ In some use cases, you'll want to make use of Organization pages, for members of that department, and have separate Community pages where members of that department interact with other users.

SCENARIO 1: MARKETING USER

- ❖ Logs in to Intranet, checks Cafe Menu. LA menu appears
 - ❖ Heads over to Marketing Organization pages to check To Dos on a dashboard application
- My Company's Portal

My Company

Marketing

Locations

LA

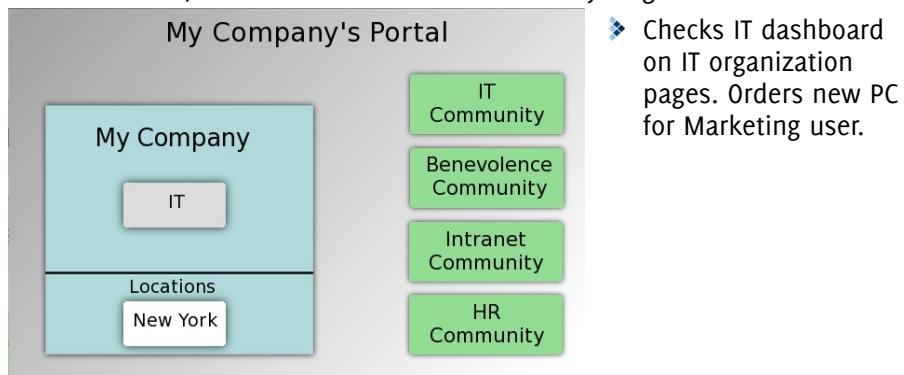
- IT Community
 - Health/Fitness Community
 - Intranet Community
 - HR Community

WWW.LIFERAY.COM

LIFERAY.

SCENARIO 2: IT USER

- ❖ Logs into Intranet. Sees company news on 5K charity run in New York
- ❖ Clicks link, heads to benevolence community. Registers for run



WWW.LIFERAY.COM

LIFERAY.

Notes:



INTRODUCTION TO LIFERAY

Copyright © 2000 – 2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

LIFERAY OVERVIEW

- ❖ Runs on all major application servers & servlet containers, databases, and operating systems with over 700 deployment combinations
- ❖ JSR-168 & JSR-286 Compliant
- ❖ Out-of-the-box usability with over 60 portlets pre-bundled.
- ❖ Built in Content Management System (CMS)

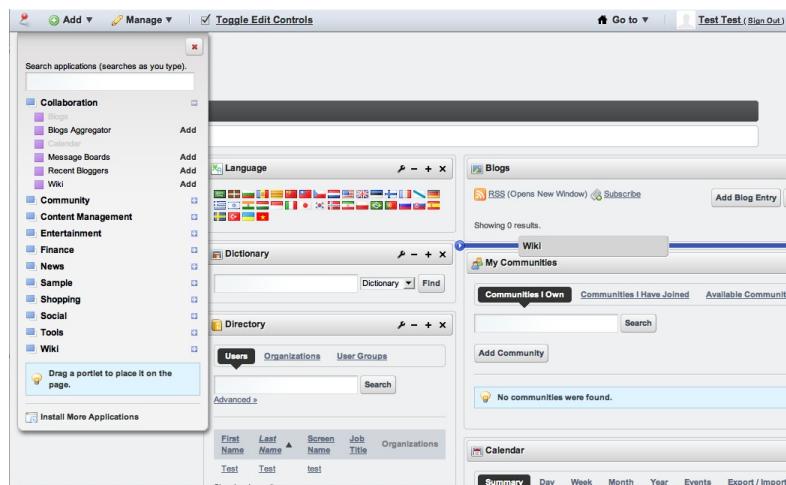
LIFERAY OVERVIEW

- ❖ Collaboration suite
- ❖ Personalized pages for all users
- ❖ Benchmarked as among the most secure portal platforms using LogicLibrary's Logiscan suite

WWW.LIFERAY.COM



DRAG AND DROP USER INTERFACE



WWW.LIFERAY.COM



USER-FRIENDLY FEATURES

- ❖ Easy AJAX UI
- ❖ Hot-Deployable Theme Architecture with Online Software Catalog
- ❖ Freeform / WebOS layout
- ❖ Just In Time Portlet Rendering
- ❖ Fine-Grained Permissions System

WWW.LIFERAY.COM



CONTENT MANAGEMENT SYSTEM

A screenshot of the Liferay Content Management System. At the top, a grey bar displays the title 'Web Content'. Below it, a form titled 'New Web Content' is shown. The 'Name' field is highlighted with a yellow background. A checked checkbox labeled 'Localized' is present. Under 'Language', there are two dropdown menus both set to 'English (United States)'. A rich-text editor window is open, showing various toolbar icons for styling text, including bold, italic, underline, and various alignment and list options. The editor has a light gray background and a dark gray border.

WWW.LIFERAY.COM



DEVELOPER FRIENDLY FEATURES

- ❖ Service Builder: Liferay's Code Generator
- ❖ CSS Compliance
- ❖ Liferay Archive (LAR) Import / Export
- ❖ Write portlets with Plugins SDK
- ❖ Completely customize Liferay with EXT Plugins

WWW.LIFERAY.COM



SERVICE BUILDER

Auto-generate Hibernate objects, Spring mappings, web services, and business logic wrappers using a single XML file

```
<service-builder root-dir=". " package-path="com.ext.portlet">
<portlet name="Library" short-name="Library" />
<entity name="Book" local-service="true">
    <!-- PK fields -->
    <column name="bookId" type="String" primary="true" />
    <!-- Other fields -->
    <column name="title" type="String" />
</entity>
</service-builder>
```

WWW.LIFERAY.COM



DATABASE AND SERVER AGNOSTIC

Operating Systems	Application Servers	Databases
✓ Linux (CentOS, RHEs, SUSE, Ubuntu, and others) ✓ Unix (AIX, HP-UX, Mac OS X, Solaris, and others) ✓ Windows	✓ Apache Geronimo ✓ Sun GlassFish ✓ JBoss ✓ JOnAS ✓ OracleAS ✓ SUN JSAS ✓ WebLogic ✓ WebSphere	✓ Apache Derby ✓ IBM DB2 ✓ Firebird ✓ Hypersonic ✓ Informix ✓ InterBase ✓ JDataStore ✓ MySQL ✓ Oracle ✓ PostgreSQL ✓ SAP ✓ SQL Server ✓ Sybase
Servlet Containers		
✓ Jetty ✓ Resin ✓ Tomcat		

WWW.LIFERAY.COM



STANDARDS COMPLIANT

JSR 127 (JSF)
JSR 168 (Portlet Specification)
JSR 286 (Portlet 2.0 Specification)
JSR 170 (Content Repository)
JSR 208 (Java Business Integration)
AJAX, Spring, Struts, Tiles,
Velocity, WSRP

WWW.LIFERAY.COM



Notes: _____



INSTALLING LIFERAY

Copyright © 2000 – 2011 Liferay, Inc.
All Rights Reserved.

No material may be reproduced electronically or in print, duplicated,
copied, sold, resold, or otherwise exploited for any commercial purpose
without express written consent of Liferay, Inc.

INSTALLING LIFERAY

- Liferay bundles the Java Runtime Environment (JRE) in the Liferay-Tomcat bundle, so for Windows users installing Liferay is just a matter of unzipping the Liferay-Tomcat Bundle.
- If you are not using Windows, you will need to install the appropriate JRE for your operating system and set up the JAVA_HOME or JRE_HOME environment variable (see optional slides).
- Unzip the Liferay-Tomcat Bundle

OVERVIEW

- This presentation describes the installation procedure for a Liferay bundle.
- It covers how to quickly get Liferay up and running using the Liferay-Tomcat bundle.
- The intended audience is anyone who wants to use Liferay Portal.

WWW.LIFERAY.COM



DOWNLOAD THE JRE (OPTIONAL)

- If necessary, download the latest Java Runtime Environment from <http://java.oracle.com>
- Use the version for your platform
- Download the installer to your local system

WWW.LIFERAY.COM



JRE VS. JDK (OPTIONAL)

- JRE = Java Runtime Environment
Required to run Java applications
- JDK = Java Development Kit
Required to develop Java applications
- The JDK comes with the JRE, but for development the JDK is required

SET JAVA_HOME (OPTIONAL)

- The next step is to set the [JAVA_HOME Environment Variable](#) and point it to the location of the JRE
- Tomcat needs to know the location of the JRE in order to run properly
- Once you have set the JAVA_HOME Environment Variable, you will need to add JAVA_HOME/bin to the path
- Consult documentation on the <http://java.oracle.com> web site for additional details.

For further study, take a look at catalina.bat to see why we need to set JAVA_HOME
catalina.bat is located in the /bin directory in the Liferay-Tomcat bundle

LIFERAY-TOMCAT BUNDLE

- The Liferay-Tomcat bundle contains all of the settings and files that are needed to run Liferay
- Just unzip the bundle, double click startup.bat and Liferay is up and running!
- A bundle of Liferay Portal Enterprise Edition (Bundled with Tomcat for JDK 6.0) can be found within the provided material
- The file should look something like this
liferay-portal-[version].zip

LIFERAY-TOMCAT BUNDLE

- Create the following directory structure:
C:\liferay\bundles
- Unzip Tomcat (liferay-portal-[version].zip) to C:\liferay\bundles\
- Go to C:\liferay\bundles\liferay-portal-[version]\tomcat-6.0.26\bin
- Double click startup.bat

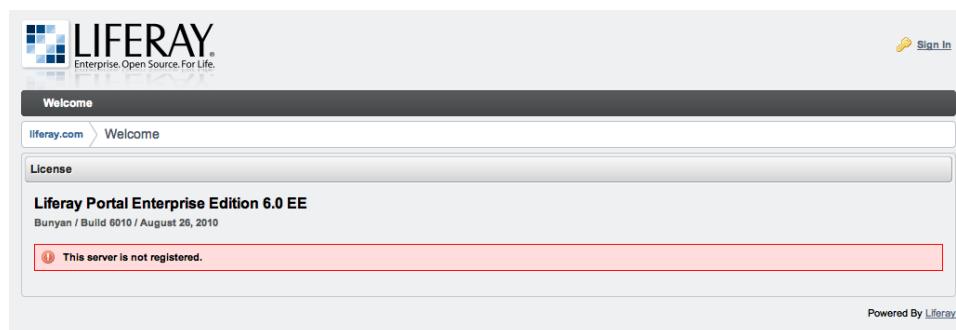
CHECKPOINT!

- You should see something like the following output in your console when Liferay starts sucessfully:

```
19:14:23,497 INFO [PluginPackageUtil:1109] Reloading repositories
19:14:26,488 INFO [HotDeployUtil:69] Initializing hot deploy manager 70841254
19:14:27,182 INFO [AutoDeployDir:105] Auto deploy scanner started for
/Users/stephenkostas/java/liferay/bundles/liferay-portal-6.0-ee/deploy
Sep 23, 2010 7:14:32 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory tunnel-web
Sep 23, 2010 7:14:36 PM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
Sep 23, 2010 7:14:36 PM org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009
Sep 23, 2010 7:14:36 PM org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/25 config=null
Sep 23, 2010 7:14:36 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 68574 ms
```

REGISTERING LIFERAY EE

- The first time you start the server, you will be presented with a message saying *This server is not registered.*



ENTERING YOUR LICENSE KEY

- On earlier versions of Liferay Enterprise Edition, an administrator would need to manually enter a license key the first time they started Liferay.
- With Liferay 6 we've changed the method so that you simply deploy an XML file which contains the license key.
- Copy the *license-portal-developer-6.0-liferaycom.xml* that you received into the <liferay-bundle-version>/deploy folder.
- Wait a few seconds and refresh the page. You are now ready to start using Liferay.

Notes:



LIFERAY USER INTERFACE

Copyright © 2000 – 2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

LIFERAY PORTAL

- ❖ Liferay is a Portal Server.
- ❖ All application functionality is in portions of the page called *portlets*.
- ❖ The look and feel of the pages themselves can be changed by the use of *themes*.
- ❖ Themes and Portlets are collectively called *plugins* and are installed by portal administrators.

PORTLETS

- Portlets are web applications that appear in windows on a page.
- Portlets are written by developers, and can have any functionality a web application has.
- Many portlets can be added to a single page, allowing convenient access to many applications in one place.

WWW.LIFERAY.COM

LIFERAY.

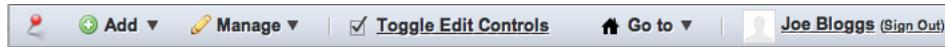


ADMINISTRATIVE PORTLETS

- Liferay's administration interface is itself implemented as portlets.
- From Liferay 5.2, all the administrative portlets are accessible through the Control Panel.
 - They are classified in 4 areas: Personal, Content, Portal and Server.

LIFERAY.

NAVIGATING LIFERAY



- ❖ The Dockbar, which appears across the top of the screen, is a new feature in Liferay 6.
- ❖ Previous versions of Liferay used the Dock, which was a single drop down type menu which contained all of the navigation and administration options now contained in the Dockbar,
- ❖ The Dockbar is the key to navigating in Liferay.
- ❖ The Dockbar enables you to navigate the portal, create new pages, modify your current page, and access administrative functions.

WWW.LIFERAY.COM

 LIFERAY.

DEFAULT ADMINISTRATIVE USER

- ❖ Click the *Sign In* link.
- ❖ Liferay by default uses email addresses as user names.
- ❖ Log in with the name *test@liferay.com*
- ❖ The password is *test*.

WWW.LIFERAY.COM

 LIFERAY.

TERMS OF USE

- ❖ The first time a user logs in, a terms of use page is displayed.
- ❖ Users cannot access the portal without agreeing to the terms of use.
- ❖ The terms of use can be modified by a developer.
- ❖ The terms of use page can be turned off by a portal administrator.

PASSWORD REMINDER QUERIES

- ❖ The first time a user logs in, the user must choose a password reminder question and answer.
- ❖ The question and answer is used to help reset a user's password
- ❖ Password reminder queries can be turned off by a portal administrator.

THE DOCKBAR



- ❖ Control Panel shows you all the administrative functions.
- ❖ Add → More allows you to add portlets to the page.
- ❖ Page Layout lets you change the layout of the page.
- ❖ Manage Page takes you to the page administration screen.
- ❖ Toggle Edit Controls allows you to view the page as would a visitor who is not logged in.
- ❖ Go to shows you the community and organization pages to which you can go.
- ❖ Clicking on your name allows you to edit your user information.
- ❖ Sign Out logs you out.

WWW.LIFERAY.COM

LIFERAY

Notes:



PLUGINS SDK

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

PLUGINS

- ❖ In version 4.3.0, Liferay introduced the idea of hot-deployable *plugins*.
- ❖ Later versions have added more features and new types of plugins.

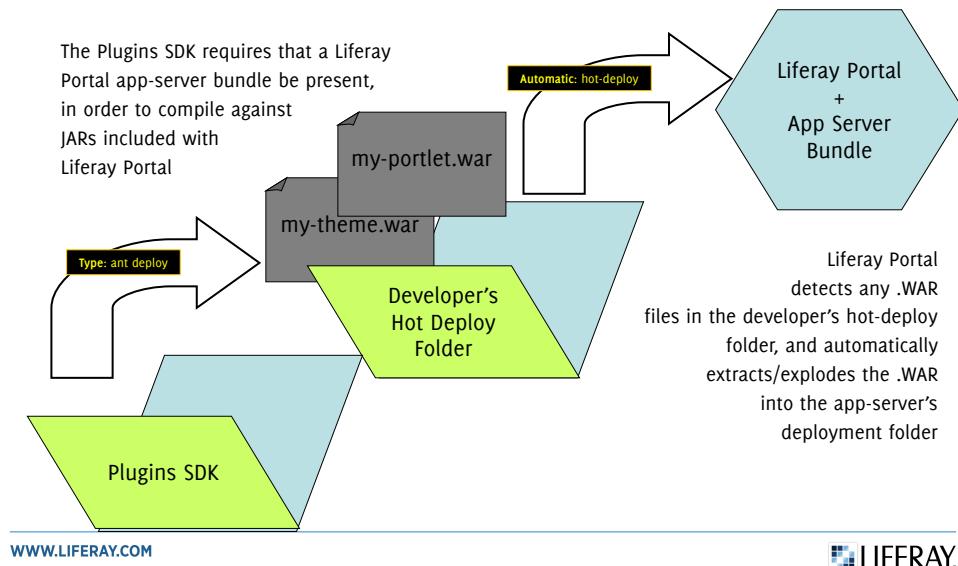
Currently there are six types of plugins:

- ❖ Portlets
- ❖ Themes
- ❖ Layout Templates
- ❖ Web Modules
- ❖ Hooks
- ❖ EXT

PLUGINS SDK

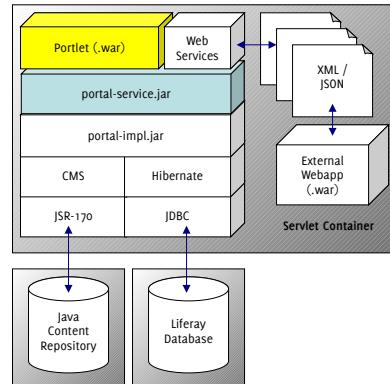
- The Plugins SDK is a simple environment for the development of Liferay plugins.
- Portlet, theme and layout development used to take place in the *ext* environment.
- Starting with Liferay version 6.0, the EXT environment is no longer supported.
- The Plugins SDK is to be used instead to create hot-deployable portlets, themes and layouts.

HOW IT WORKS

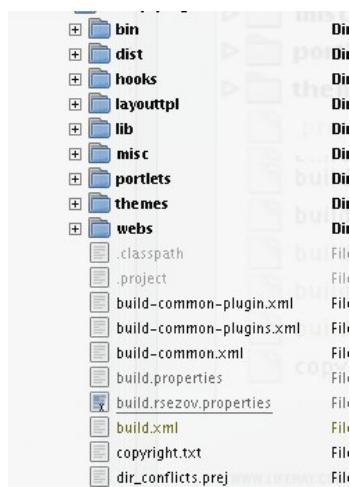


PORTLET DEPENDENCIES

- ❖ Portlets developed in the Plugins SDK may only import classes from *portal-service.jar* and other JARs contained in the portlet's WEB-INF/lib folder
- ❖ This forces portlets to rely completely on the Portal's API, and not to depend on implementation classes defined in *portal-impl.jar*

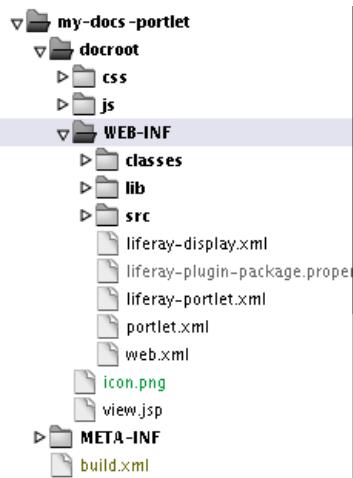


PLUGINS SDK LAYOUT



- ❖ Portlets go in the *portlets* folder
- ❖ Themes go in the *themes* folder
- ❖ Layout templates go in the *layouttpl* folder
- ❖ Web applications go in the *webs* folder
- ❖ Hooks go in the *hooks* folder
- ❖ EXT plugins go in the *ext* folder
- ❖ Ant scripts are used to build and deploy plugins to a local application server

PORTLETS

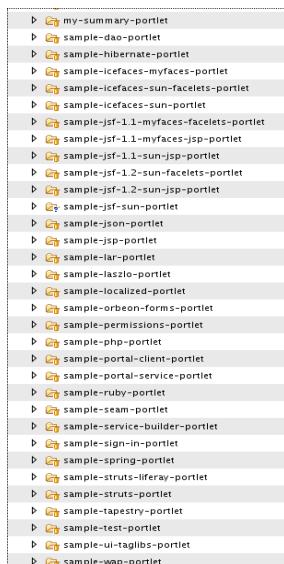


- A simple *create* script (for Windows, Mac, or Linux) is used to generate a new, blank portlet project

WWW.LIFERAY.COM

LIFERAY.

PORTLETS



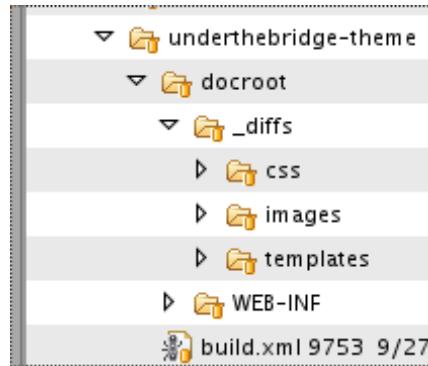
- Portlets can make use of any application framework that Liferay supports
- For a complete list of sample portlets, see <http://svn.liferay.com/repos/public/plugins/>
- Username: *guest*
- Password: *<blank>*

WWW.LIFERAY.COM

LIFERAY.

THEMES

- ❖ Themes are generated with a similar *create* script
- ❖ Themes are based on *differences* from the default theme
- ❖ Everything is customizable

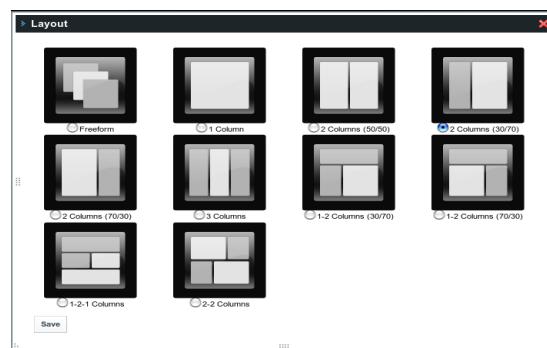


WWW.LIFERAY.COM

 LIFERAY.

LAYOUT TEMPLATES

- ❖ Layout Templates are the simplest of plugins.
- ❖ They are also generated with a *create* script.
- ❖ They go in the *layouttpl* folder.



WWW.LIFERAY.COM

 LIFERAY.

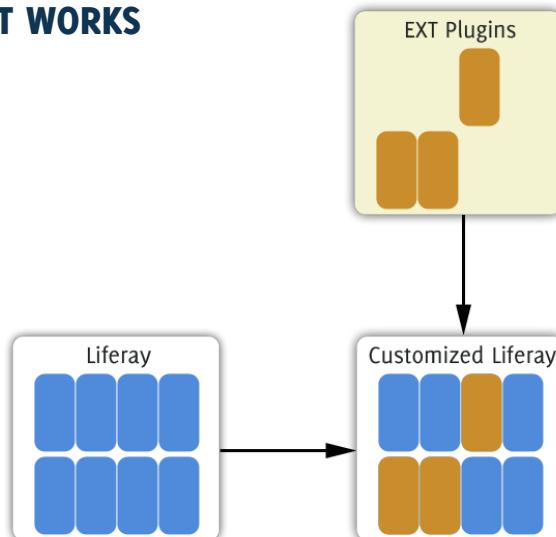
HOOKS

- ❖ A hook plugin can adapt and extend Liferay Portal's functionality by:
 - Providing a predefined configuration
 - Providing implementations for the available extension points (hooks) defined in portal.properties
 - Overriding portal JSPs (use with care!)
- ❖ Examples:
 - StartupAction
 - Login hooks

EXT

- ❖ An EXT plugin can modify the portal's core classes and behavior in ways that are not possible with a Hook.
 - Overriding portal implementation classes (portal-impl)
 - Overriding portal util implementations (portal-util)
 - Overriding portal web configuration files
- ❖ Examples:
 - Modify Struts Actions
 - Override Liferay Service using Spring

HOW IT WORKS



WWW.LIFERAY.COM

LIFERAY.

WEB APPLICATIONS

- ❖ Liferay can also integrate with certain web applications.
- ❖ This will not be covered in this course, but you can find examples of this in Liferay's Subversion repository.
- ❖ There, you will find
 - Search engine integration plugins
 - Enterprise Service Bus integration plugins
 - ...and more

WWW.LIFERAY.COM

LIFERAY.

Notes: _____



LIFERAY TRAINING SETUP

Copyright © 2000-2010 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

INSTALL JAVA

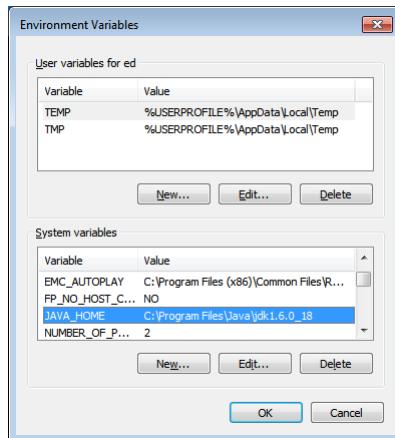
- Install the JDK from your CD. Make sure you use the correct version for your platform (i.e., Windows 64-bit or Windows 32-bit).
[/software/java](#)
(Use the default options)

- Alternatively, you can download the latest JDK 6 here
<http://java.sun.com>

- JRE vs JDK
 - JRE = Java Runtime Environment
Required to run Java applications
 - JDK = Java Development Kit
Required to develop Java applications

CHECKPOINT!

Verify that JAVA_HOME is correct



WWW.LIFERAY.COM

LIFERAY

CHECKPOINT!

- ❖ You must open a **new command prompt** for the Environment Variables to take effect.
- ❖ Click Start → Run
- ❖ Type **cmd** and press Enter
- ❖ Type **path**
- ❖ Make sure there is **only one JDK in the Path**.

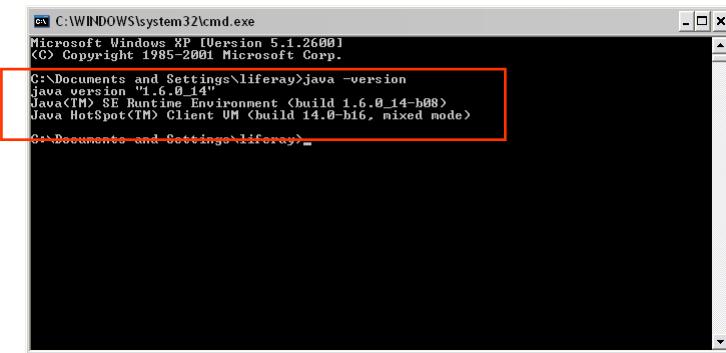
A screenshot of a Windows command prompt window titled 'C:\WINDOWS\system32\cmd.exe'. The window displays the output of the 'path' command. The output shows the current PATH environment variable, which includes 'C:\Java\jdk1.6.0_18\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program Files\Shri Technologies\ATI Control Panel;'. The path 'C:\Java\jdk1.6.0_18\bin' is highlighted with a red box.

WWW.LIFERAY.COM

LIFERAY

CHECKPOINT!

- ❖ Click Start → Run...
- ❖ Type cmd
- ❖ Type java -version
- ❖ The following message should be displayed:



A screenshot of a Windows Command Prompt window titled 'C:\WINDOWS\system32\cmd.exe'. The window shows the following text:
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\liferay>java -version
java version "1.6.0_14"
Java(TM) SE Runtime Environment (Build 1.6.0_14-b08)
Java HotSpot(TM) Client VM (Build 14.0-b16, mixed mode)
C:\Documents and Settings\liferay>

MYSQL

- ❖ MySQL is a leading open source database
- ❖ It is widely used to power many web sites
- ❖ It is small and fast
- ❖ Its small footprint makes it ideal for a developer's machine

INSTALL MYSQL

- ❖ A copy of MySQL can be found within the provided material
- ❖ If necessary, you can also download MySQL from
<http://dev.mysql.com/downloads/mysql/5.0.html#downloads>
- ❖ Run the installer
- ❖ Under install type, select *Custom*

MYSQL INSTALL

- ❖ Change the default path to C:\mysql



MYSQL INSTALL

- ❖ Click *Next* through the remainder of the installer
- ❖ When it completes, leave the *Configure the MySQL Server Now* box checked, and click *Finish*
- ❖ In the configuration wizard, select *Standard Configuration* and click *Next*
- ❖ Keep the defaults on the next page (install as a service) and click *Next*

MYSQL INSTALL

- ❖ Create a root password (i.e., *liferay*) and check the “root access from remote machines” box. Click *Next*



Note: in MySQL 5.1, when asked for the current password it should be left blank

MYSQL INSTALL

- Select Execute
- Select Finish
- MySQL is now successfully installed



CREATE MYSQL DATABASE

- From a command prompt, type

mysql -u root -p

- From the MySQL prompt, type

create database lportal character set utf8;

LIFERAY DEVELOPER STUDIO

- Liferay Developer Studio is the Integrated Development Environment for writing applications for the Liferay platform.
- It provides many conveniences which make it easy for you to get started implementing your site on Liferay.
- The version we are providing to you as a bonus for attending training is exactly the same as the commercial version, with a few additions to help with the training exercises.

WWW.LIFERAY.COM



LIFERAY IS TOOL AGNOSTIC

- Though we are using Liferay Developer Studio in training, it is important to note that Liferay is tool agnostic.
- You can use anything from a command prompt and text editor to a full blown IDE to develop for Liferay's platform.
- We use Liferay Developer Studio in training to streamline the setup process and to help the exercises to go more smoothly.
- If you are interested in learning more about setting up Liferay manually, there are slide decks in your book explaining how to do that which we can go over.

WWW.LIFERAY.COM



SETTING UP LIFERAY DEVELOPER STUDIO

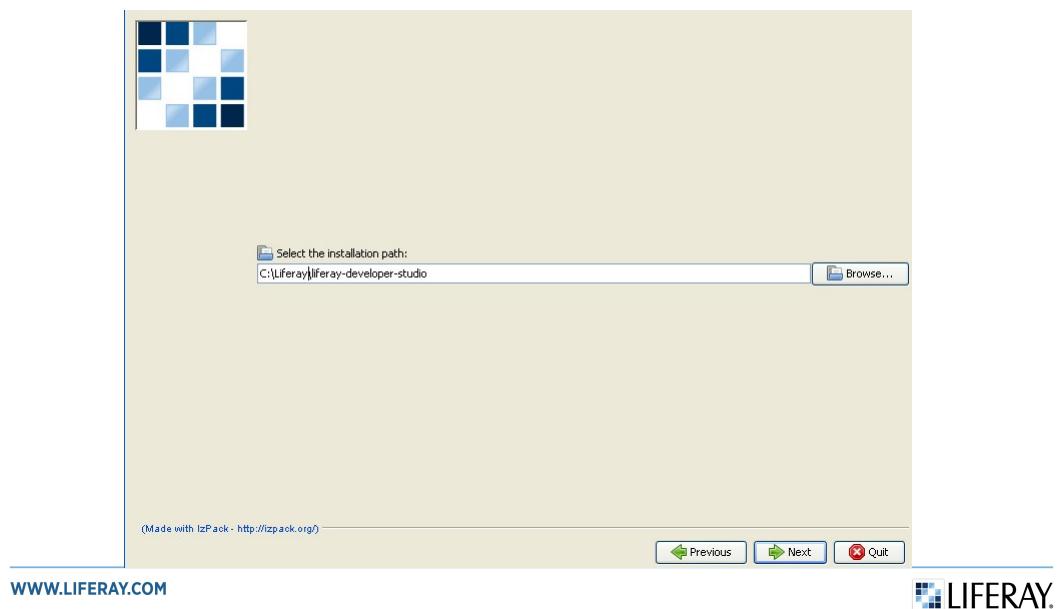
- Now that you have Java installed, it is easy to install Liferay Developer Studio.
- Find the Liferay Developer Studio installer for your platform. It will be clearly labeled by OS and platform (32-bit or 64-bit).
- Launch the installer by double-clicking on it.

WWW.LIFERAY.COM



INSTALLING LIFERAY DEVELOPER STUDIO

- Install in C:\Liferay\liferay-developer-studio as shown.

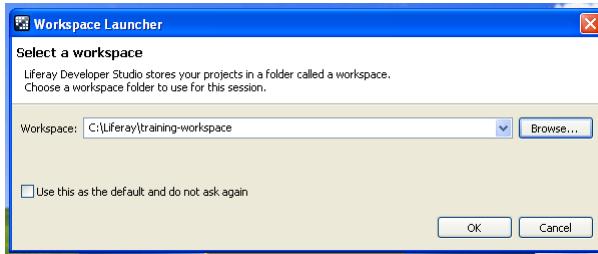


WWW.LIFERAY.COM



LAUNCHING LIFERAY DEVELOPER STUDIO

- Once installed, you can launch Liferay Developer Studio from your Start menu.
- Liferay Developer Studio is based on Eclipse, so it needs a workspace.
- Place your workspace in the C:\Liferay folder, as shown below.

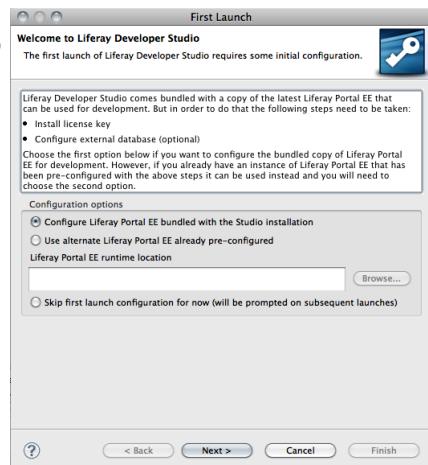


WWW.LIFERAY.COM

LIFERAY

LIFERAY DEVELOPER STUDIO FIRST RUN

- Liferay Developer Studio will run a wizard to help you get started on the first run.
- On the first screen, choose the default option to use the embedded Liferay.
- When prompted for a license key, select browse, and locate the license key provided with your training materials. Click Next.

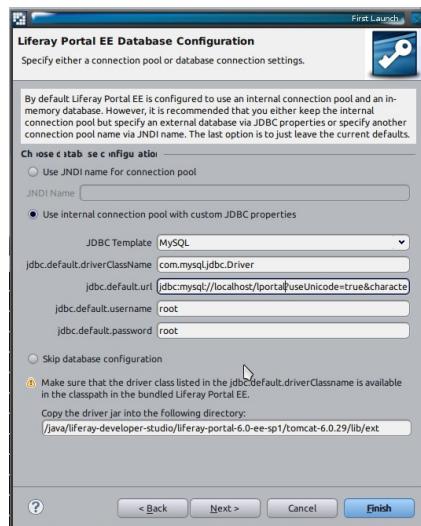


WWW.LIFERAY.COM

LIFERAY

CONNECT TO MYSQL DATABASE

- Select *Use internal connection pool*.
- Choose MySQL as the database.
- Check the URL to be sure it has the correct database name
- Provide the MySQL user name and password you selected earlier.
- Click *Finish*.



WWW.LIFERAY.COM

LIFERAY

WHAT HAPPENED BEHIND THE SCENES

- Studio created a file called *portal-ext.properties* in your bundle folder (C:\Liferay\liferay-developer-studio\liferay-portal-[version]) with the following contents:

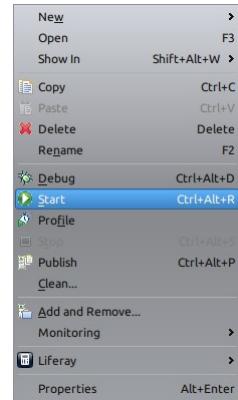
```
jdbc.default.driverClassName=com.mysql.jdbc.Driver  
jdbc.default.url=jdbc:mysql://localhost/lportal?  
useUnicode=true&characterEncoding=UTF-8&useFastDateParsing=false  
jdbc.default.username=root  
jdbc.default.password=liferay
```

WWW.LIFERAY.COM

LIFERAY

START THE LIFERAY SERVER

- Right-click your Liferay server in the bottom left corner of LDS.
- Select Start.
- The first time Liferay starts, it takes a while because it needs to create the Liferay database.



WWW.LIFERAY.COM

LIFERAY

REGISTER A PLUGINS SDK WITH LIFERAY DEVELOPER STUDIO

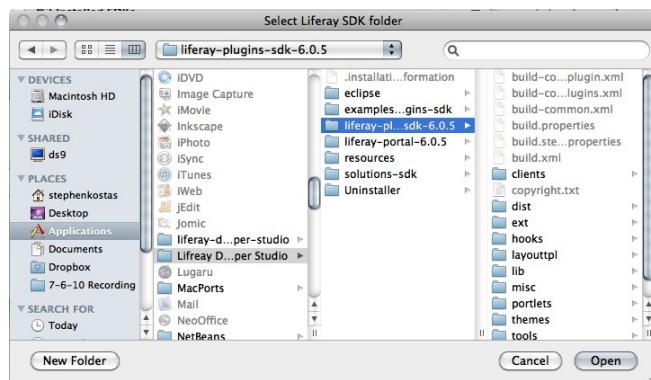
- We have grouped all the solutions to the exercises in this course into a separate Plugins SDK which we've called Solutions SDK.
- Let's register this SDK with our IDE so that we can import its projects.

WWW.LIFERAY.COM

LIFERAY

REGISTERING THE SOLUTIONS SDK

- In Liferay Developer Studio, click Window → Preferences → Liferay → Installed SDKs
- Click Add.
- Browse to the *solutions-sdk* folder in the LDS installation.
- Select and click OK.
- Click OK again.

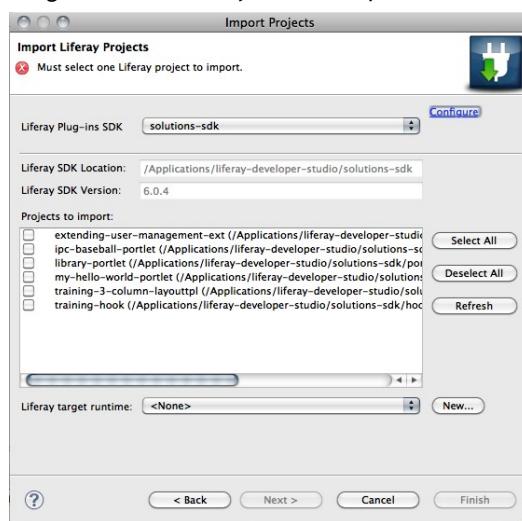


WWW.LIFERAY.COM

LIFERAY

IF YOU GET STUCK

- If you get stuck during the course of the training, you will now be able to import the training solutions into your workspace to examine them.

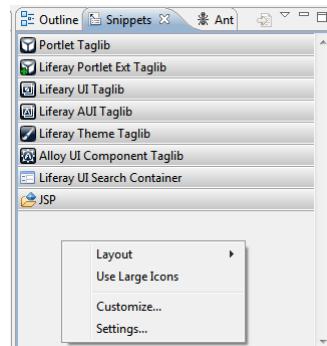


WWW.LIFERAY.COM

LIFERAY

MANUALLY IMPORTING SNIPPETS

- Liferay Developer Training Edition comes bundles with snippets for the code that you will need to complete the class exercises.
- If for some reason these fail to import automatically you can add them manually.
- Right click in the *Snippets* view, and click *Customize*.

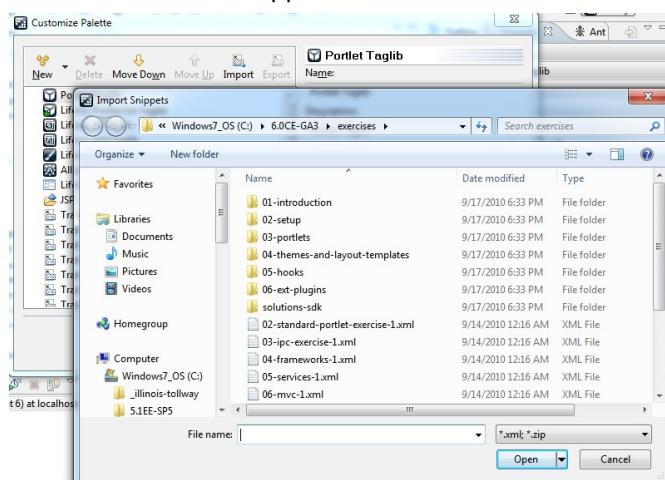


WWW.LIFERAY.COM

LIFERAY

SELECTING SNIPPETS TO IMPORT

- Click *Import*, navigate to the src directory and import the XML files that correspond to the relevant snippets.



WWW.LIFERAY.COM

LIFERAY

Notes:



BASIC SETUP

Copyright © 2000 – 2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

OVERVIEW

- ❖ This presentation describes the tools necessary for development on a Liferay Portal platform.
- ❖ It covers how to quickly get Liferay up and running for a developer using the Liferay-Tomcat bundle.
- ❖ You will learn how to install Ant, which is the build tool Liferay uses.
- ❖ You will also learn how to install MySQL and connect Liferay to it.
- ❖ The intended audience is Java developers.
- ❖ A prerequisite for this is to already have the Liferay-Tomcat bundle installed and working.

INSTALL JAVA

- ❖ Install the JDK from your CD
[/software/java](#)
(Use the default options)

- ❖ Alternatively, you can download the latest JDK 6 here
<http://java.sun.com>

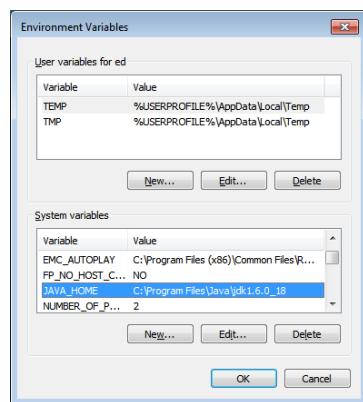
- ❖ JRE vs JDK
 - JRE = Java Runtime Environment
Required to run Java applications
 - JDK = Java Development Kit
[Required to develop Java applications](#)

WWW.LIFERAY.COM



CHECKPOINT!

Verify that JAVA_HOME is correct

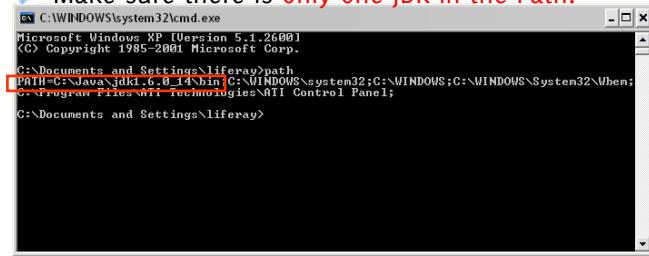


WWW.LIFERAY.COM



CHECKPOINT!

- ❖ You must open a **new command prompt** for the Environment Variables to take effect!
- ❖ Click Start → Run...
- ❖ Type **cmd** and press Enter
- ❖ Type **path**
- ❖ Make sure there is **only one JDK in the Path!**



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\liferay>path
PATH=C:\Java\jdk1.6.0_14\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;
C:\Program Files\WMI Technologies\WMI Control Panel;

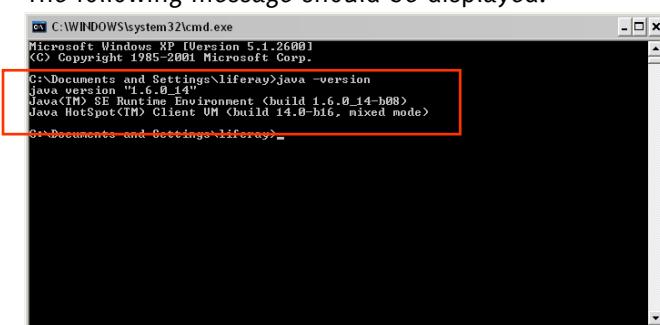
C:\Documents and Settings\liferay>
```

WWW.LIFERAY.COM

 LIFERAY.

CHECKPOINT!

- ❖ Click Start → Run...
- ❖ Type **cmd**
- ❖ Type **java -version**
- ❖ The following message should be displayed:



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\liferay>java -version
java version "1.6.0_14"
Java(TM) SE Runtime Environment (build 1.6.0_14-b08)
Java HotSpot(TM) Client VM (build 14.0-b16, mixed mode)

C:\Documents and Settings\liferay>
```

WWW.LIFERAY.COM

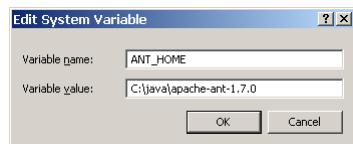
 LIFERAY.

INSTALL ANT

- ❖ Ant uses build scripts to automate tasks like cleaning out directories, compiling classes and moving files
- ❖ A copy of Ant can be found within the provided material
- ❖ The file should look something like this apache-ant-[version]-bin.zip
- ❖ If necessary, you can also download Ant from <http://ant.apache.org/bindownload.cgi>
- ❖ Unzip Ant (apache-ant-1.7.0-bin.zip) to C:\java

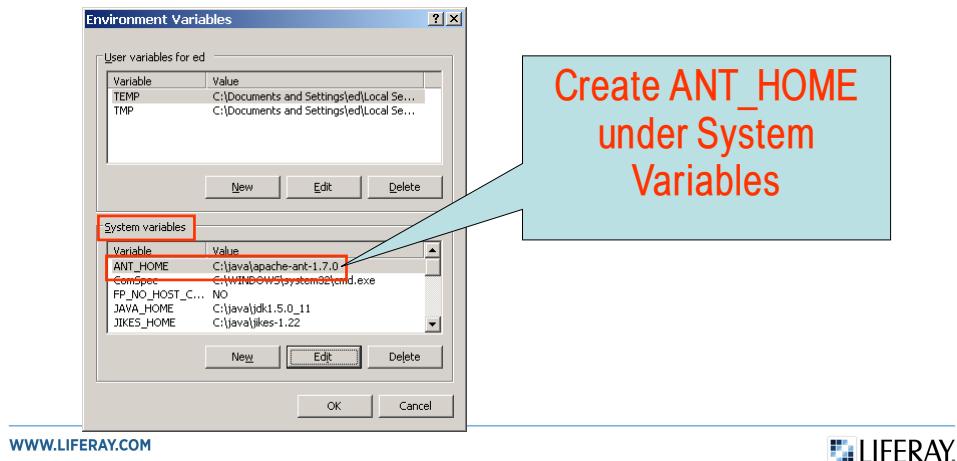
SET ANT_HOME

- ❖ Click Start → Settings → Control Panel → System → Advanced → Environment Variables
- ❖ Under System Variables click New
- ❖ **Variable name:** ANT_HOME
Variable value: C:\java\apache-ant-1.7.1



CHECKPOINT!

Verify that ANT_HOME is correct



ADD ANT_HOME TO THE PATH

- ❖ Under System Variables click Path then Edit
 - ❖ Type %ANT_HOME%\bin; after %JAVA_HOME%\bin;
 - ❖ Example:
- ```
%JAVA_HOME%\bin;%JAVA_HOME%\bin;
%ANT_HOME%\bin;%SystemRoot%\system32;%SystemRoot%;%SystemRoot%
%\System32\Wbem;C:\Program Files\ATI Technologies\ATI Control Panel;
```

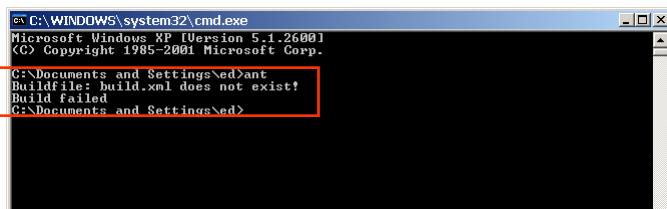
Don't put spaces in the  
Path, and don't forget  
the semi-colon!

---

## CHECKPOINT!

You must open a new command prompt for the Environment Variables to take effect!

- ❖ Click Start → Run...
- ❖ Type *cmd*
- ❖ Type *ant*
- ❖ The following message should be displayed:



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\ed>ant
Buildfile build.xml does not exist!
Build failed
C:\Documents and Settings\ed>
```

---

## MYSQL

- ❖ MySQL is a leading open source database
- ❖ It is widely used to power many web sites
- ❖ It is small and fast
- ❖ Its small footprint makes it ideal for a developer's machine

## INSTALL MYSQL

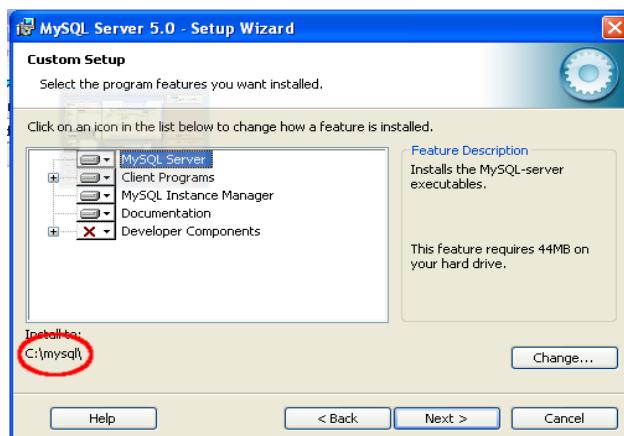
- ❖ A copy of MySQL can be found within the provided material
- ❖ If necessary, you can also download MySQL from  
<http://dev.mysql.com/downloads/mysql/5.0.html#downloads>
- ❖ Run the installer
- ❖ Under install type, select *Custom*

WWW.LIFERAY.COM

LIFERAY.

## MYSQL INSTALL

- ❖ Change the default path to C:\mysql



WWW.LIFERAY.COM

LIFERAY.

---

## MYSQL INSTALL

- ❖ Click *Next* through the remainder of the installer
- ❖ When it completes, leave the *Configure the MySQL Server Now* box checked, and click *Finish*
- ❖ In the configuration wizard, select *Standard Configuration* and click *Next*
- ❖ Keep the defaults on the next page (install as a service) and click *Next*

WWW.LIFERAY.COM

LIFERAY.

---

## MYSQL INSTALL

- ❖ Create a root password (i.e., *liferay*) and check the “root access from remote machines” box. Click *Next*



Note: in MySQL 5.1, when asked for the current password it should be left blank

WWW.LIFERAY.COM

LIFERAY.

## MYSQL INSTALL

- ❖ Select Execute
- ❖ Select Finish
- ❖ MySQL is now successfully installed



WWW.LIFERAY.COM

LIFERAY.

## CREATE MYSQL DATABASE

- ❖ From a command prompt, type

```
mysql -u root -p
```

- ❖ From the MySQL prompt, type

```
create database lportal character set utf8;
```

WWW.LIFERAY.COM

LIFERAY.

---

## CONFIGURE RESOURCES TO USE MYSQL

- ❖ Create a file called portal-ext.properties in your bundle folder (liferay-portal-[version]) and paste the following contents into it:

```
jdbc.default.driverClassName=com.mysql.jdbc.Driver
jdbc.default.url=jdbc:mysql://localhost/lportal?
useUnicode=true&characterEncoding=UTF-8&useFastDateParsing=false
jdbc.default.username=root
jdbc.default.password=****
```

- Be careful to use the appropriated values for username and password

- ❖ You can find this file in the provided material in the exercises folder *exercises/03-portal-dev/02-setup/01-portal-ext-properties.txt*

---

## EXECUTE THE SERVER

- ❖ Go to tomcat-6.0.26/bin
- ❖ Execute startup.bat

## INCREASE THE SIZE OF THE CONSOLE

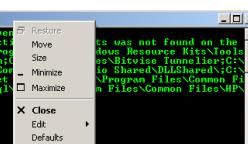
- ❖ Increase the window size and the screen buffer size of the console
- ❖ Increasing the window size makes it easier to read the console.
- ❖ Increasing the screen buffer size allows us to see the full stack trace when an error is thrown
- ❖ As a developer you must know how to look through the stack trace to determine the cause of an error

WWW.LIFERAY.COM

LIFERAY

## INCREASE THE SIZE OF THE CONSOLE

- ❖ Right click the console and select *Properties*



```
Feb 20, 2009 2:02:04 PM org.apache.catalina.core.ContainerBase lifecycleEvent
INFO: The Apache Tomcat Native library which allows optimal performance in production mode was not found on the classpath. You can either download Resource Kits\Tools\Native from the Apache website or build your own using the Apache Ant 1.7.1 buildscript located at C:\Program Files\Apache Software Foundation\Apache Ant-1.7.1\bin\apache-ant-1.7.1-bin.xml. If you are running on a Mac OS X system, you may need to use the Homebrew package manager to install the native library.
Feb 20, 2009 2:02:04 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 406 ms
Feb 20, 2009 2:02:04 PM org.apache.catalina.core.StandardService start
INFO: Starting service Catalina
Feb 20, 2009 2:02:04 PM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: JBoss Seam v2.2.0.GA
Feb 20, 2009 2:02:04 PM org.apache.catalina.core.StandardHost start
INFO: XML validation disabled
loading file:/C:/liferay/bundles/liferay-portal-tomcat-5.5-5.1.2/websapps/ROOT/WEB-INF/lib/portal-impl.jar!/system.properties
Loading file:/C:/liferay/bundles/liferay-portal-tomcat-5.5-5.1.2/websapps/ROOT/WEB-INF/classes/system-ext.properties
14:02:24,881 INFO [TomcatDeployment] loading file:/C:/liferay/bundles/liferay-portal-tomcat-5.5-5.1.2/websapps/ROOT/WEB-INF/lib/portal-impl.jar!/portal.properties
loading file:/C:/liferay/bundles/liferay-portal-tomcat-5.5-5.1.2/websapps/ROOT/WEB-INF/classes/portal-ext.properties
14:02:24,914 INFO [TomcatDeployment] (PortletImpl@228) Portlet lib directory C:/liferay/bundles/liferay-portal-tomcat-5.5-5.1.2/websapps/ROOT/WEB-INF/lib/
14:02:24,914 INFO [TomcatDeployment] Detected server tomcat 5.5.1.2 (Galvin / Build 5102 / October 3, 2008)
14:02:24,914 INFO [TomcatDeployment] (ServerDetector@76) Detected server tomcat
14:02:24,981 INFO [TomcatDeployment] (HotDeployer@1176) Initializing hot deploy manager 23442754
14:02:25,009 INFO [TomcatDeployment] (HotDeployer@1176) Registering themes for liferay-jedi-theme
14:02:25,059 INFO [TomcatDeployment] (HotDeployListener@92) Registering themes for liferay-jedi-theme registered successfully
14:02:25,059 INFO [TomcatDeployment] (HotDeployer@1176) Themes for liferay-jedi-theme found for deployment
14:02:25,074 INFO [TomcatDeployment] (HotDeployListener@104) Themes for liferay-jedi-theme registered successfully
14:02:25,077 INFO [TomcatDeployment] (HotDeployListener@104) Initializing Spring FrameworkServlet 'Spring Servlet'
14:02:25,077 INFO [TomcatDeployment] (HotDeployListener@104) Registering portletHttpBaseProtocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
Feb 20, 2009 2:02:27 PM org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13-0.0.0.0-8009
Feb 20, 2009 2:02:27 PM org.apache.jk.server.JkMain start
INFO: JK running in 0.0 time=0/16 config=null
14:02:27,000 INFO [TomcatDeployment] (HotDeployer@1176) StoreLoader load
INFO: Find registry server-registry.xml at classpath resource
Feb 20, 2009 2:02:27 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 2247 ms
```

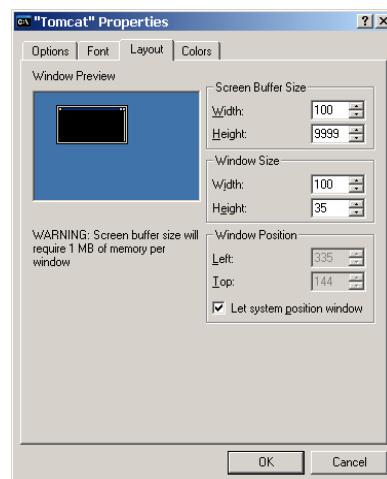
WWW.LIFERAY.COM

LIFERAY

---

## INCREASE THE SIZE OF THE CONSOLE

- ❖ Click the Layout tab
- ❖ Screen Buffer Size
  - Width: 100
  - Height: 9999
- ❖ Window Size
  - Width: 100
  - Height: 35
- ❖ At the prompt, click Save properties for future windows with the same title



WWW.LIFERAY.COM

 LIFERAY.

---

## LAST STEP!

- ❖ Your default browser should automatically open this url:  
<http://localhost:8080/web/guest>
  - Login: [test@liferay.com](mailto:test@liferay.com)
  - Password: test

WWW.LIFERAY.COM

 LIFERAY.

**Notes:** \_\_\_\_\_



# PLUGINS SDK SETUP

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

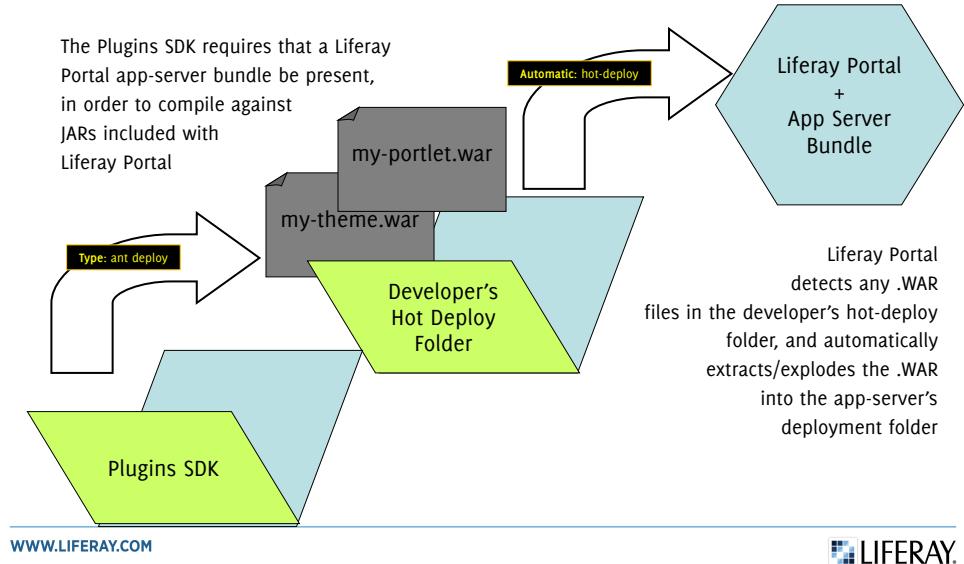
No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

## PLUGINS SDK

The Plugins SDK is a simple environment for the development of Liferay plugins.

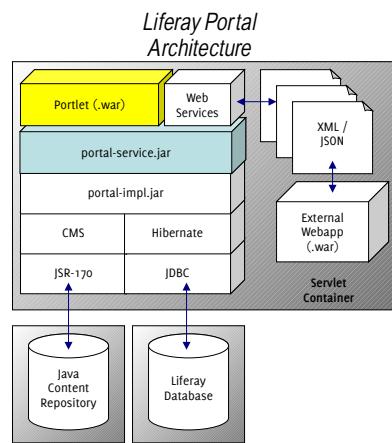
- Portlet, theme and layout development used to take place in the *ext* environment.
- The *ext* plugin does not support portlet and theme development, except for modifying/extending Liferay's out-of-the-box portlets.
- The Plugins SDK is to be used instead to create hot-deployable portlets, themes and layouts.

## HOW IT WORKS



## PORTLET DEPENDENCIES

- Portlets developed in the Plugins SDK may only import classes from **portal-service.jar**, and other JARs contained in the portlet's **WEB-INF/lib** folder
- This forces portlets to rely completely on the Portal's API, and **not to depend on** implementation classes defined in **portal-impl.jar**



---

## SETUP

- ❖ The Plugins SDK can be found in the provided material with the name: liferay-plugins-sdk-[version]
- ❖ Unzip the file into C:\liferay\plugins

---

WWW.LIFERAY.COM



---

## SETUP

- ❖ The first thing we need to do is tell the plugins SDK where to deploy the plugins (so that they are deployed to the server)
- ❖ We do this by setting `app.server.dir` in `build.${user.name}.properties`

---

WWW.LIFERAY.COM



---

## WHAT IS app.server.dir?

- ❖ app.server.dir (Application Server Directory) is the property that tells the Plugins SDK where to deploy the plugins
- ❖ For further study, take a look at build.properties located in the Plugins SDK directory

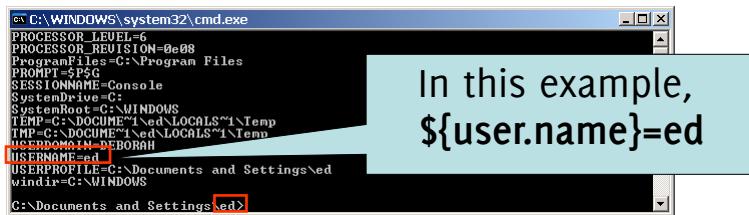
---

## WHAT IS BUILD.\${USER.NAME}.PROPERTIES?

- ❖ build.\${user.name}.properties is a file that we will use to override the default properties (*build.properties*). This way you do not have to modify any of the defaults
- ❖ \${user.name} is the user name for your account on your computer

## FIND OUT YOUR USER NAME

- Click Start -> Run
- Type cmd and press Enter
- Type set in the cmd prompt
- Write down your user name



```
ex C:\WINDOWS\system32\cmd.exe
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=0e08
ProgramFiles=C:\Program Files
PROMPT=PG
SESSIONNAME=Console
SystemDrive=C:
SystemRoot=C:\WINDOWS
TEMP=C:\DOCUMENTS\ed\LOCALS\Temp
TMP=C:\DOCUMENTS\ed\LOCALS\Temp
USERDOMAIN=DEBORAH
USERNAME=ed
USERPROFILE=C:\Documents and Settings\ed
windir=C:\WINDOWS
C:\Documents and Settings\ed>
```

In this example,  
\${user.name}=ed

## CREATE THE PROPERTIES FILE

- Go to the top level folder of the plugins SDK and **create** the build.  
\${user.name}.properties file.
- In the previous example, \${user.name}=ed
- This user would create a file called **build.ed.properties**

### Example 1:

If your user name is **Administrator**, create a file called  
**build.Administrator.properties**

### Example 2:

If your user name is **john.doe**, create a file called  
**build.john.doe.properties**

---

## SET THE APPLICATION SERVER DIRECTORY

- ❖ Enter the following in build.\${user.name}.properties

```
app.server.dir=C:/liferay/bundles/<liferay bundle name>/<application-server>
```

Example:

```
app.server.dir=C:/liferay/bundles/liferay-portal-6.0-ee-sp1/tomcat-6.0.26
```

---

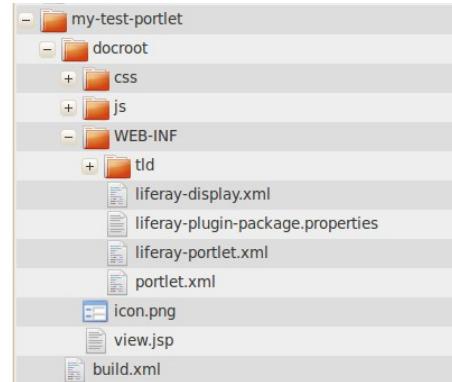
## GENERATE YOUR FIRST PORTLET

- ❖ Creating a blank portlet application with the Plugins SDK is easy
- ❖ One simple command creates a whole project skeleton
- ❖ Navigate to c:/liferay/plugins/portlets and enter the command
  - (Windows)
    - create.bat my-test "My Test"
  - (Mac/Linux)
    - Give execution permission to create.sh
    - ./create.sh my-test "My Test"

---

## MY TEST PORTLET

- ❖ The skeleton is a fully functional portlet
- ❖ It can be deployed as-is in order to see the functionality



[WWW.LIFERAY.COM](http://WWW.LIFERAY.COM)

 LIFERAY.

---

## DEPLOYING PORTLETS

- ❖ To deploy your portlet, navigate to the root of your project and **type:**  
*ant deploy*
- ❖ Your project will be built and deployed to your application server

[WWW.LIFERAY.COM](http://WWW.LIFERAY.COM)

 LIFERAY.

## VERIFY

- Open Windows Explorer and find the deploy folder found in your bundle directory, i.e.:

C:/liferay/bundles/<liferay bundle name>/deploy

- Verify that you see this file there:

my-hello-world-portlet-<version number>.war

## TOMCAT CONSOLE OUTPUT

Start Tomcat and verify that the console indicates that the .war archive was auto-detected, exploded, copied, and registered:

```
INFO: Server startup in 20563 ms
INFO [com.liferay.portal.kernel.deploy.auto.AutoDeployDir] Processing my-hello-world-portlet-4.4.2.1.war
INFO [com.liferay.portal.deploy.auto.PortletAutoDeployListener] Copying portlets for C:\Documents and Settings\yourname\liferay\deploy\my-hello-world-portlet-4.4.2.1.war
Expanding: C:\Documents and Settings\yourname\liferay\deploy\my-hello-world-portlet-4.4.2.1.war into
C:\training\tomcat\temp\20070909221216703
Copying 17 files to C:\training\tomcat\bin..\webapps\my-hello-world-portlet
Copying 1 file to C:\training\tomcat\bin..\webapps\my-hello-world-portlet
Deleting directory C:\training\tomcat\temp\20070909221216703
INFO [com.liferay.portal.deploy.auto.PortletAutoDeployListener] Portlets for C:\Documents and Settings\yourname\liferay\deploy\my-hello-world-portlet-4.4.2.1.war copied successfully
log4j:WARN No appenders could be found for logger (org.apache.commons.digester.Digester).
log4j:WARN Please initialize the log4j system properly.
INFO [com.liferay.portal.deploy.hot.PluginPackageHotDeployListener] Reading plugin package for my-hello-world-portlet
INFO [com.liferay.portal.deploy.hot.PluginPackageHotDeployListener] Plugin package liferay/my-hello-world-portlet/4.4.2.1/war registered successfully
INFO [com.liferay.portal.deploy.hot.PortletHotDeployListener] Registering portlets for my-hello-world-portlet
INFO [com.liferay.portal.deploy.hot.PortletHotDeployListener] Portlets for my-hello-world-portlet registered successfully
```

## TRY IT OUT!

- Login to Liferay Portal
- Dockbar → Add → More
- Expand the “Sample” category
- Add the “My Test” portlet
- My Test portlet appears on page



WWW.LIFERAY.COM

LIFERAY.

Notes: \_\_\_\_\_



# DEVELOPMENT ENVIRONMENT

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

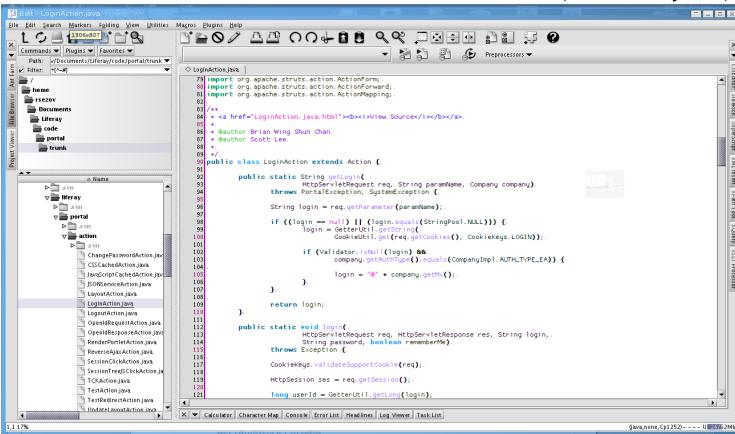
No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

## TOOLS

- ❖ Liferay is tool-agnostic.
- ❖ Anything from a text editor to a full-blown IDE can be used.
- ❖ Liferay core developers use everything from command line Ant and EditPlus to Eclipse, Netbeans, or IntelliJ.

## TEXT EDITOR

- A text editor and command line Ant can be used (below is JEdit)



The screenshot shows the JEdit text editor interface. The title bar says "JEdit (LogAction.java)". The menu bar includes File, Edit, View, Markers, Refactor, Plugins, Favorites, Macros, Debug, Help. The toolbar has icons for Open, Save, Cut, Copy, Paste, Find, Replace, etc. The left sidebar shows a project tree with "Portal" selected, containing "action" and "lib". The main editor area contains the following Java code:

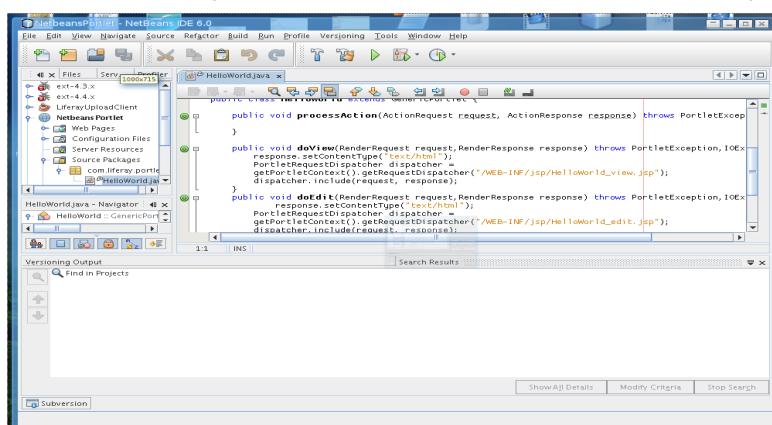
```
1 import org.apache.struts.action.ActionForm;
2 import org.apache.struts.action.ActionMapping;
3 import org.apache.struts.action.Parameter;
4 /**
5 *
6 */
7 Author: Brian Wing Shun Chan
8 Author: Scott Lee
9 *
10 */
11
12 public class LogAction extends Action {
13
14 public static String getLogin(
15 HttpServletRequest req, String paramString, Company company)
16 throws PortletException, SystemException {
17
18 String login = req.getParameter(paramString);
19
20 if ((login == null) || (login.equals(StringPool.NULL))) {
21 login = GetterUtil.getString(company.getAuthType());
22
23 if (Validator.isNull(login)) {
24 login = GetterUtil.getString(company.getAuthType());
25 }
26
27 login = " " + company.getAuthType();
28
29 }
30
31 return login;
32 }
33
34 public static void setLogin(
35 HttpServletRequest req, HttpServletResponse res, String login,
36 String password, boolean rememberMe)
37 throws PortletException {
38
39 HttpSession session = req.getSession(true);
40
41 CookieUtil.validateSupportCookie(req);
42
43 HttpSession reses = req.getSession();
44
45 long userId = GetterUtil.getLong(login);
46
47 }
48 }
```

WWW.LIFERAY.COM



## NETBEANS

- Netbeans is an open source IDE which can be used with Liferay



The screenshot shows the Netbeans IDE interface. The title bar says "Netbeans 6.0 - HelloWorld.java". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Build, Run, Profile, Versioning, Tools, Window, Help. The toolbar has icons for Open, Save, Run, Stop, etc. The left sidebar shows a project tree with "HelloWorld" selected, containing "src" and "Web Pages". The main editor area contains the following Java code:

```
1 package com.liferay.portlet.helloworld;
2
3 import javax.portlet.ActionRequest;
4 import javax.portlet.ActionResponse;
5 import javax.portlet.RenderRequest;
6 import javax.portlet.RenderResponse;
7
8 import com.liferay.portal.kernel.util.GetterUtil;
9 import com.liferay.portal.kernel.util.StringPool;
10
11
12 public class HelloWorld extends GenericPortlet {
13
14 public void processAction(ActionRequest request, ActionResponse response) throws PortletException {
15
16 }
17
18 public void doView(RenderRequest request, RenderResponse response) throws PortletException, IOException {
19 response.setContentType("text/html");
20 getPortletContext().getPortletDispatcher().setPortletName("HelloWorld");
21 dispatcher.include(request, response);
22 }
23 public void doEdit(RenderRequest request, RenderResponse response) throws PortletException, IOException {
24 PortletRequestDispatcher dispatcher =
25 getPortletContext().getPortletDispatcher("WEB-INF/jsp/helloworld_edit.jsp");
26 dispatcher.include(request, response);
27 }
28 }
```

WWW.LIFERAY.COM



---

## ECLIPSE

- ❖ Eclipse is an open source IDE which can be used with Liferay.
  - Because of its prevalence in the marketplace and its free availability we use it for training.
  - This does not mean we endorse it as the *best* solution; Liferay developers use a number of different tools based on their preference.
- ❖ Poll: what do you use?
- ❖ We will next go over how to set up a Liferay development environment in Eclipse.
  - For consistency in class, we will install a fresh copy of the latest version of Eclipse.

WWW.LIFERAY.COM



---

## INSTALLING ECLIPSE

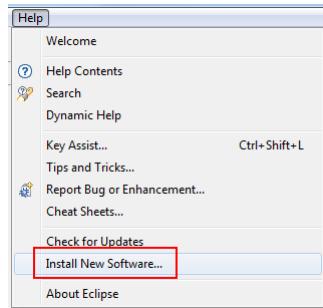
- ❖ You should find a copy of Eclipse in the *files* folder of your training materials.
- ❖ Unzip this file to a location on your hard drive suitable for running it.
- ❖ We recommend any of the following three options:
  - C:\eclipse
  - C:\Programs\eclipse
  - C:\java\eclipse
- ❖ Once you have unzipped the file, navigate to the Eclipse folder in your file manager.
- ❖ Using the right mouse button, drag the eclipse.exe file to your desktop and select *Create Shortcut Here*.
- ❖ You can now start Eclipse by double-clicking on the shortcut.

WWW.LIFERAY.COM



## INSTALLING LIFERAY IDE

- Once Eclipse opens, go to Help > Install New Software



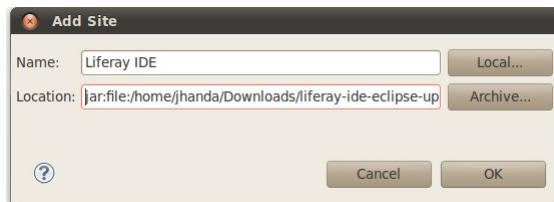
- Click Add... button to open the Add Site dialog

WWW.LIFERAY.COM

LIFERAY.

## INSTALLING LIFERAY IDE

- Type Liferay IDE for name
- Click the "Archive" button and browse to files provided on thumb drive liferay-ide-eclipse-updatesite-[version].zip



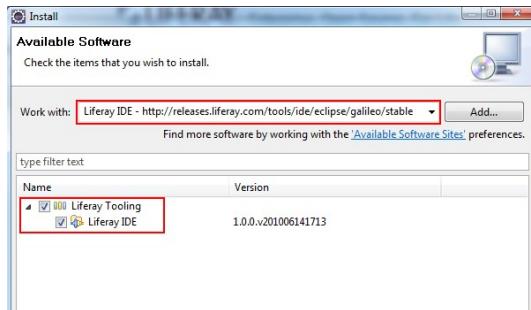
- Click Ok

WWW.LIFERAY.COM

LIFERAY.

## INSTALLING LIFERAY IDE

- ❖ Expand *Liferay Tooling* and select the *Liferay IDE*
- ❖ Click Next and then click Finish to begin the install



- ❖ (Note) You may see several Mylyn features that need to be installed as well, that is expected, as there has been a Mylyn update since the Galileo SR2 release

WWW.LIFERAY.COM

 LIFERAY.

## INSTALLING LIFERAY IDE

- ❖ After plugins download and install you will have to accept that the content is unsigned and then restart eclipse.
- ❖ After you restart, go to *Help > About Eclipse* and you should see a icon badge for Liferay IDE that shows you have it properly installed.



WWW.LIFERAY.COM

 LIFERAY.

**Notes:** \_\_\_\_\_



# SETTING UP THE LIFERAY IDE

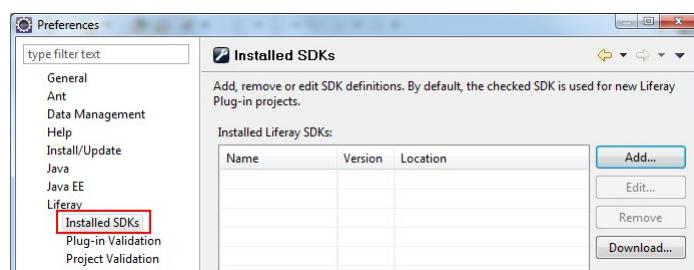
Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

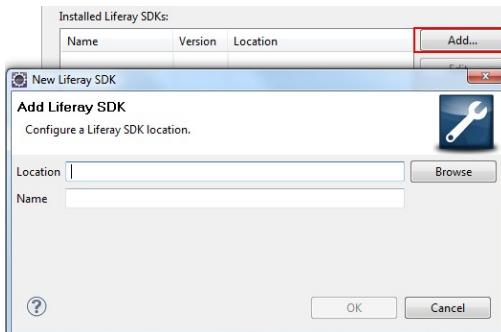
## LIFERAY PLUGINS SDK SETUP

- ❖ Before we can use the Liferay IDE, we need to configure it to recognize the Plugins SDK and the installed Bundle
- ❖ Open Preference page for Liferay > Installed SDKs
  - Go to Window > Preferences
  - Expand Liferay > Installed SDKs



## LIFERAY PLUGINS SDK SETUP

- ❖ Click *Add...*
- ❖ For the location, click *Browse* and select C:\liferay\plugins
- ❖ For name, enter *liferay-plugins-sdk*



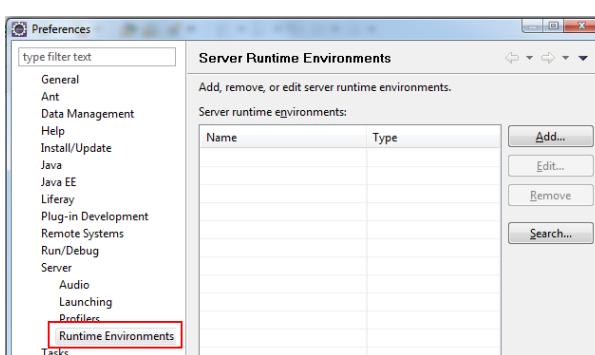
- ❖ Click *Ok*

WWW.LIFERAY.COM

LIFERAY.

## LIFERAY BUNDLE TOMCAT SETUP

- ❖ While still in the *Preferences* dialog, open the Runtime environments preference page
  - Go to Server > Runtime environments
- ❖ Click *Add...*

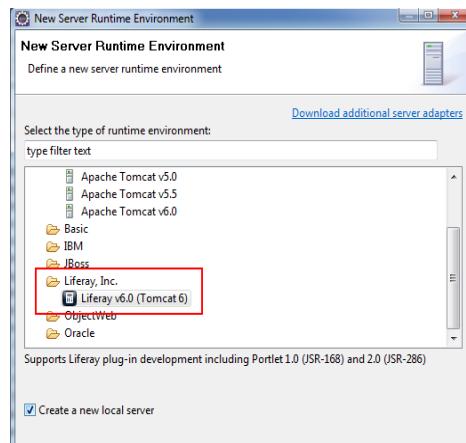


WWW.LIFERAY.COM

LIFERAY.

## LIFERAY BUNDLE TOMCAT SETUP

- ❖ Expand *Liferay, Inc.*
- ❖ Select *Liferay v6.0(Tomcat6)*
- ❖ Check the *Create a new local server* check box
- ❖ Click *Next*

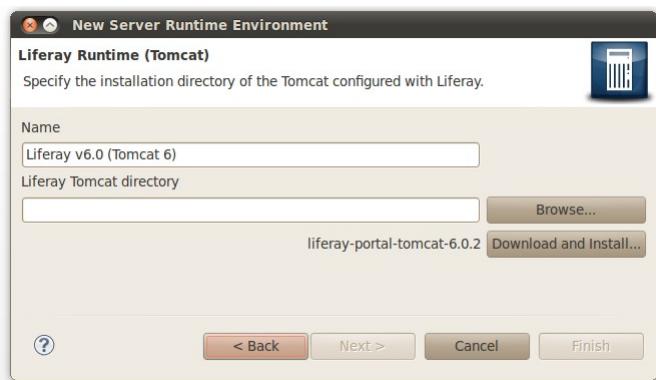


WWW.LIFERAY.COM

LIFERAY.

## LIFERAY BUNDLE TOMCAT SETUP

- ❖ Browse and select the location of the Liferay bundle

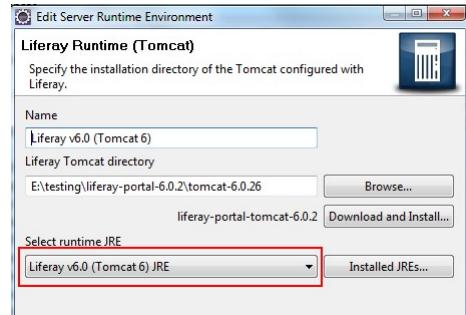


WWW.LIFERAY.COM

LIFERAY.

## LIFERAY BUNDLE TOMCAT SETUP

- ❖ Once you have selected the Liferay portal directory if it has a bundled JRE then that bundled JRE will be automatically selected as the JRE to use for launching the server. However, if there is no bundled JRE (Mac and Linux users) then you will need to select the JRE to use for launch.



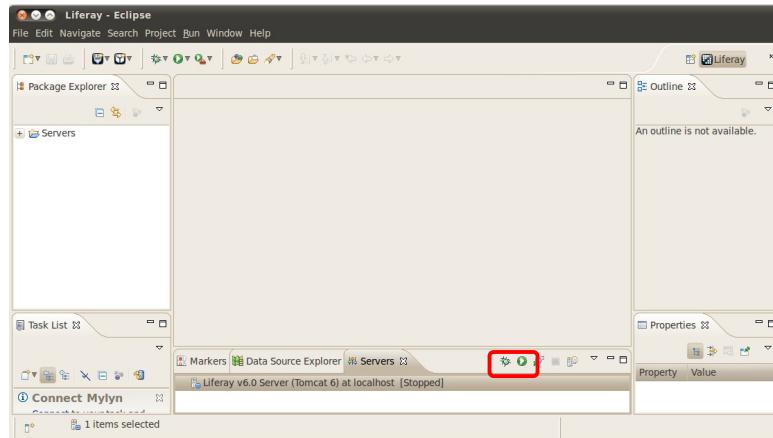
- ❖ Click *Finish*
- ❖ Click *Ok*

WWW.LIFERAY.COM

 LIFERAY.

## LIFERAY BUNDLE TOMCAT SETUP

- ❖ Go to the servers view and you should see the new server that was created. right click and choose "Start" or "Debug"

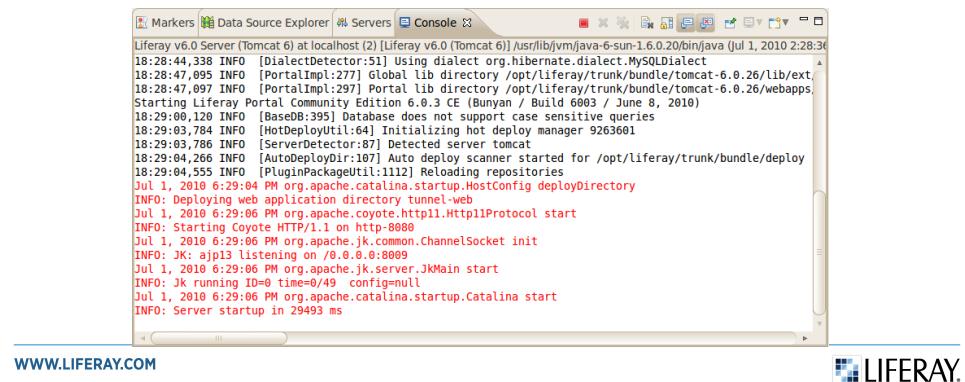


WWW.LIFERAY.COM

 LIFERAY.

## CHECKPOINT

- ❖ You should see a *Server startup . . .* message appear in the Console view
- ❖ Once it starts, the servers view will update to show that it is "Started" and then, right-click the server and select the (Liferay Portal > Open Portal Home) action.



WWW.LIFERAY.COM

 LIFERAY

### Notes:



# CONFIGURING REFERENCE PROJECTS AND DEBUGGING

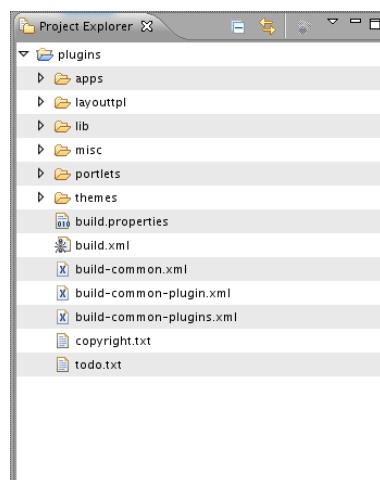
Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

## CREATE THE PLUGINS PROJECT

- ❖ Click File → New → Project
- ❖ Select Java → Java Project → Next
- ❖ Type *plugins* and select Create Project from Existing Source
- ❖ Browse to the project location and click Finish
- ❖ The plugins project will be opened in Eclipse

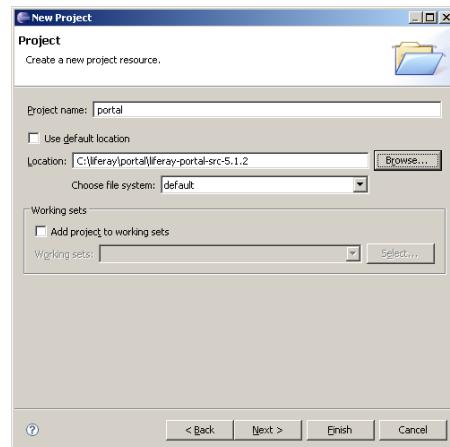


## CREATE THE DEBUG PROJECT

The portal directory is for reference purposes only. As an EE customer, you would have access to the Portal source code.

We will use it when we want to look at the Liferay Portal source code.

- Click File → New → Other → General → Project
- Project name: debug
- Browse to project location
- Click Next → Finish

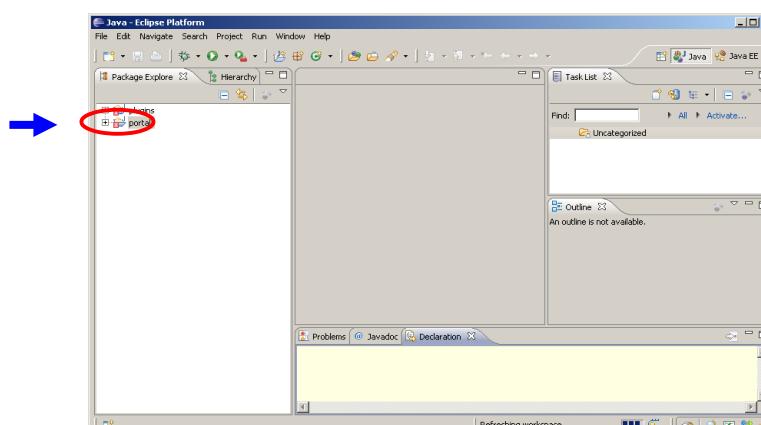


WWW.LIFERAY.COM

LIFERAY.

## CREATE THE DEBUG PROJECT (CONT.)

- Click debug
- Press F5 to refresh

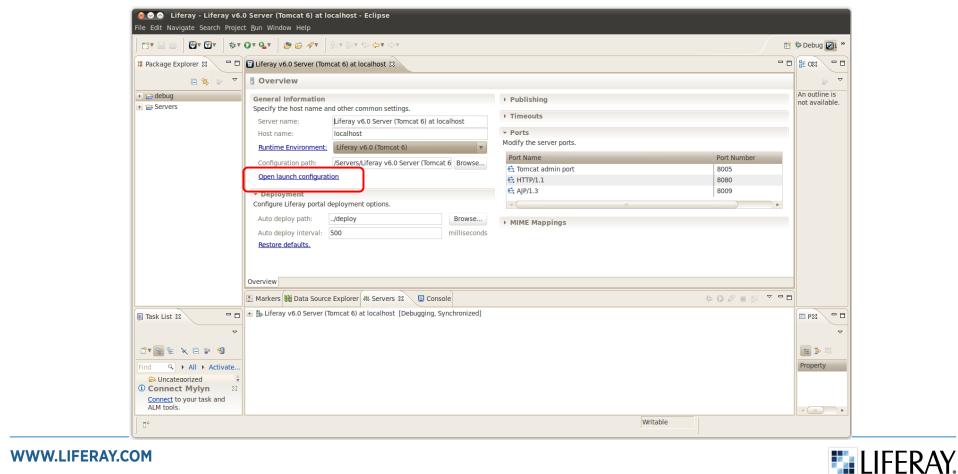


WWW.LIFERAY.COM

LIFERAY.

## ADD DEBUG TO SOURCE LOOKUP

- In the Server tab, double click on Liferay v6.0 Server
- Select *Open launch configuration*



WWW.LIFERAY.COM

LIFERAY.

## ADD DEBUG TO SOURCE LOOKUP

- Select *Source* tab and click *Add...*
- Select Java Project
- Select *debug*
- Click *Ok*, then *Ok* again

WWW.LIFERAY.COM

LIFERAY.

## CONGRATULATIONS!

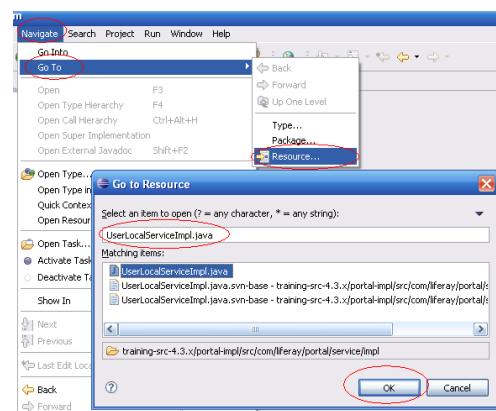
Congratulations, you've just set up the plugins and debug projects in Eclipse!

plugins → Plugins SDK

debug → Partial Liferay Portal source code

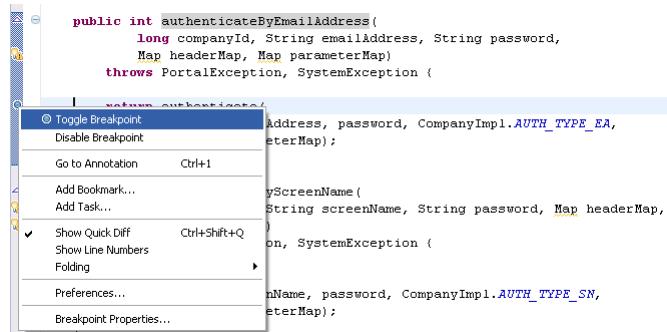
## DEBUG PORTAL SOURCE

- ❖ To test our ability to debug Liferay Portal source code we will be putting a break point into the authentication process
- ❖ Open the Open Resources dialog, you can use either of the following methods
  - *Navigate → Open Resource*
  - *Ctrl-Shift-R*
- ❖ Start typing *UserLocalServiceImpl.java* and click *Ok* once you match the correct file



## SET BREAKPOINT

- Search for "authenticateByEmailAddress"
- Set breakpoint on the only line of code in the method

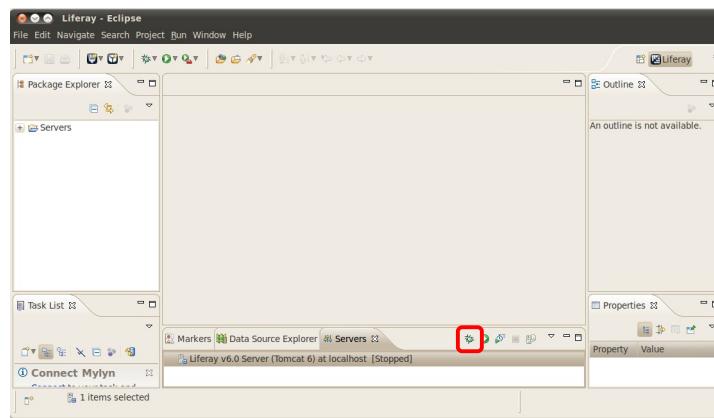


WWW.LIFERAY.COM

LIFERAY.

## START SERVER IN DEBUG MODE

- If server is not running in Debug mode, stop the server and restart in Debug mode.

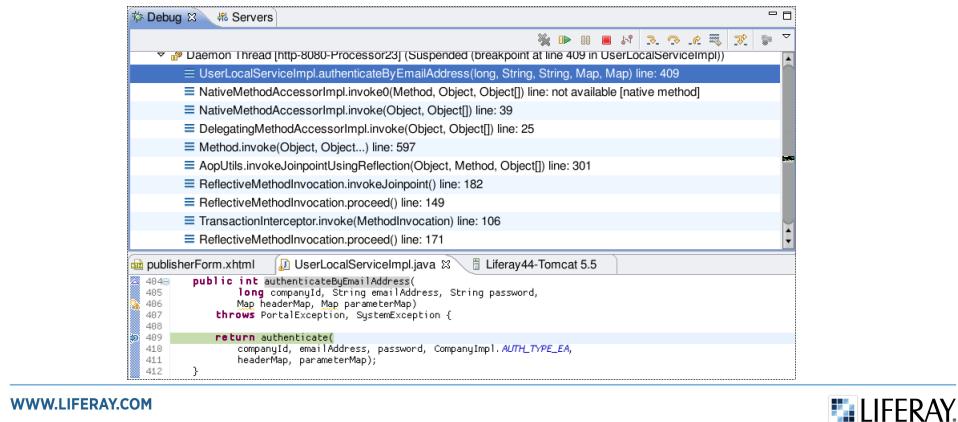


WWW.LIFERAY.COM

LIFERAY.

## LOG IN TO LIFERAY

- ❖ Go to your browser, to the address <http://localhost:8080>
- ❖ Log in to Liferay with the user name [test@liferay.com](mailto:test@liferay.com) and the password test



## CONGRATULATIONS!

Congratulations, you can now debug the source code in Eclipse!



---

## TROUBLESHOOTING

- If your Project JRE does not point to the SDK:
  - Click *Window* → *Preferences* → *Java* → *Installed JREs*
  - Click *Add*
  - Enter a name for your SDK  
Browse for the location of your SDK  
For example: *C:/Java/j2sdk1.5.0\_15*
  - Remove all entries that point to the JRE
- Your Project JRE should now point to the SDK
- If Eclipse shuts down Tomcat before it starts:
  - Open the Server Configuration and expand the *Timeouts* section.
  - Change the Start time to 500 seconds.

Notes: \_\_\_\_\_



# JAVA PORTLET OVERVIEW

Copyright © 2000 – 2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

## OVERVIEW

- This presentation will give you an overview of JSR-168 and JSR-286 and explain the benefits of using a standards compliant portal.
- It covers the portlet life cycle and the characteristics of a portlet.
- For this discussion (and the rest of the course), you will need to be familiar with web development in Java.

---

## OBJECTIVES

- ❖ What is a Portlet?
- ❖ Benefits of Java Standard Portlets
- ❖ Understand differences between Portlets and Servlets
- ❖ Portlet Lifecycle
- ❖ Portlet Modes
- ❖ Window States
- ❖ Portlet Preferences

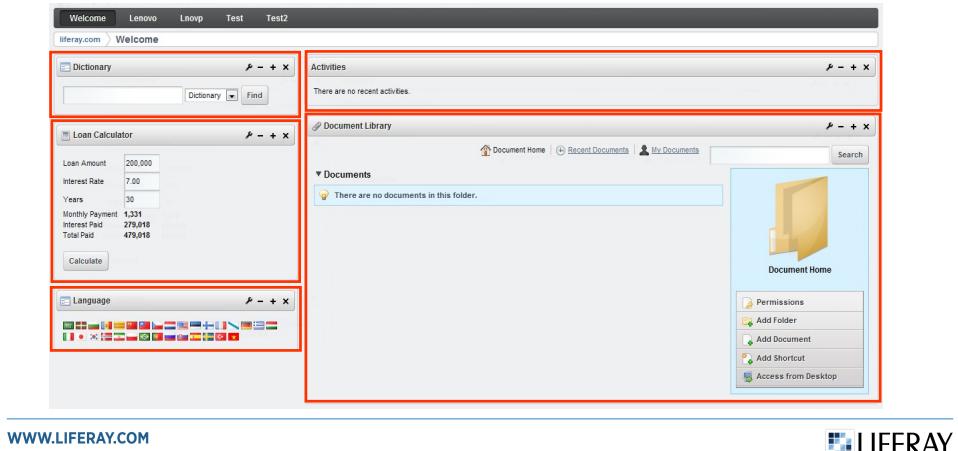
---

## WHAT IS A PORTLET?

- ❖ In the simplest terms, a portlet is a web application.
- ❖ A portlet accepts a request and generates a response.
- ❖ The request will be generated by your web browser and sent to the portlet via HTTP (or HTTPS).
- ❖ After processing the request, the portlet will return content (e.g. HTML, XHTML, WML) that is delivered to your browser and then displayed.

## WHAT IS A PORTLET?

- The content generated by a portlet is a fragment of the total web page and can be aggregated with other fragments to form a complete document.



WWW.LIFERAY.COM

LIFERAY.

## WHAT IS A JAVA PORTLET?

- There are two JSR standards that govern portal/portlet behavior.
- JSR-168 is the Portlet 1.0 Specification.
- It was created out of a need to have a specification for displaying multiple applications on the same page.
- JSR-286 is the Portlet 2.0 Specification which retains backwards compatibility with 1.0 (JSR-168) portlets.
- It adds new features based on the experience of the different portal vendors and the needs of the industry.
- The portlet specification defines the life cycle of a portlet as well as its characteristics.

WWW.LIFERAY.COM

LIFERAY.

---

## BENEFITS OF JAVA STANDARD PORTLETS

- Core Architecture
  - The core architecture is the same across Java Standard compliant portals.
  - Portlets written to the standard will run on all compliant portlet containers.
- Documentation
  - The Portlet Specification will not change, and you can reliably refer to it for information.
  - Standard Portlet API for developers.

WWW.LIFERAY.COM



---

## PORTLETS FOR SERVLET DEVELOPERS

- If you are familiar with Java servlet technology, you will see many similarities between servlets and portlets.
- Portlets are packaged as a Web Archive (.WAR) file and deployed to an application server or servlet container.
- The servlet container considers a portlet application to be just another web application that lives inside a Web ARchive (.WAR).
- Portlets must provide a valid web.xml descriptor file, although the Liferay hot-deploy process does most of the work with configuring this file.
- In addition to the standard web.xml deployment descriptor file, portlet applications require an additional descriptor file named portlet.xml.
- This descriptor tells the portlet container what portlets are included in the application, what roles they support, what portlet modes they support, and more.

WWW.LIFERAY.COM



---

## PORLETS FOR SERVLET DEVELOPERS

- However, because a portlet's response is designed to be aggregated with responses from other portlets on the page, there are some key differences.
  - Portlets are only responsible for rendering a fragment of the total page, so certain tags are not allowed (`<html>`, `<head>`, `<body>`).
  - Portlets may need to re-render, even if a user is not interacting with that portlet directly.
  - At design time, there is no way of knowing what page a portlet may be deployed on and what other portlet's may be on the page, this makes it impractical to create portlet URLs directly.
  - The Portlet API provides methods for creating all of the necessary Portlet URLs.

---

## PORLETS FOR SERVLET DEVELOPERS

- Differences (Continued)
  - Servlets have direct access to the `ServletRequest` object. According to the portlet specification, Portlets do not have access to the `ServletRequest` object but rather to a `PortletRequest` object
  - Because Portlets don't have access to the `ServletRequest` object, they are unable to read query parameters directly from the URL. The portlet specification only provides a mechanism for a Portlet to read its own URL parameters or other URL parameter that have been declared as Public Render Parameters.
  - Liferay provides utility methods that can be used to access the `ServletRequest` and query parameters, but these methods can only be used when Portlets are deployed to Liferay.

---

## PORTLET PHASES

- ❖ With servlets, the service() method processes all requests.
- ❖ Because Portlets are designed to coexist on the page with other portlets, a user may not be interacting with a portlet directly, but that portlet may still generate a request based on interaction with another portlet on the page.
- ❖ As a result, portlets have more than one type of method to process requests. These methods result in corresponding portlet phases:

  - Render Phase
  - Action Phase

- ❖ JSR-286 (Portlet 2.0) adds two more phases:
  - Event Phase
  - Resource Serving Phase

WWW.LIFERAY.COM



---

## PORTLET LIFE CYCLE

- ❖ init()
  - Initializes the portlet
- ❖ render()
  - Renders the content
- ❖ processAction()
  - Called when the user performs an action
- ❖ processEvent()
  - Called when an event has been triggered
- ❖ serveResource()
  - Called when a ResourceURL is requested
- ❖ destroy()
  - Releases the portlet object so it is eligible for garbage collection.

WWW.LIFERAY.COM



---

## PORTLET LIFE CYCLE IN ACTION

- To see the Portlet Life Cycle in Action, we will use a demonstration portlet that's been included in your exercise folder.
- Copy the *jsr286-demo-portlet* directory from:  
*/liferay/exercises/developer/o3-portlets/o1-intro/o1-java-portlet-overview/*  
to  
*/liferay/liferay-developer-studio/plugins[version]/portlets/*
- Start LDS, if it's not already running, and select your training workspace.
- Select File → Import...

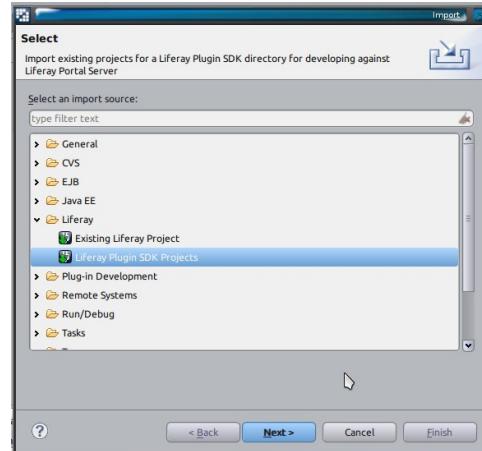
WWW.LIFERAY.COM

LIFERAY.

---

## PORTLET LIFE CYCLE IN ACTION

- Expand the Liferay Folder
- Select Liferay Plugin SDK Projects
- Select Next

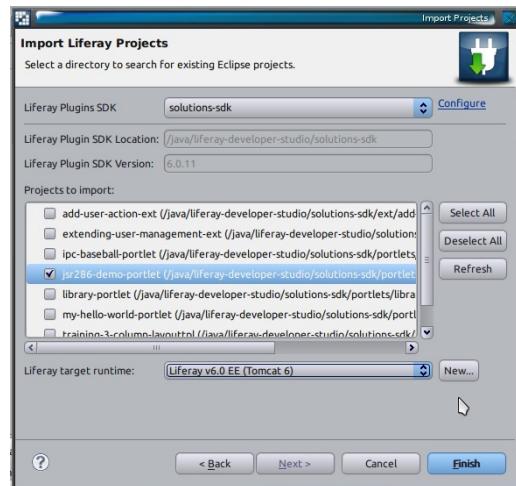


WWW.LIFERAY.COM

LIFERAY.

## PORTLET LIFE CYCLE IN ACTION

- ❖ Select the Solutions SDK
- ❖ Select the  
*jsr286-demo-portlet*
- ❖ From the drop down,  
select the *Liferay target runtime*
- ❖ Select *Finish*

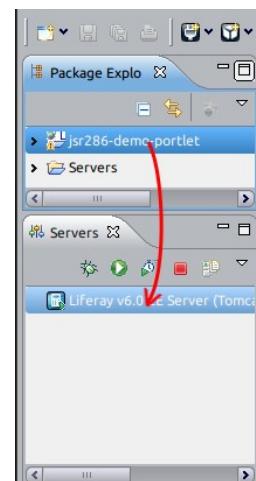


WWW.LIFERAY.COM

 LIFERAY.

## DEPLOY THE PORTLET

- ❖ Deploy the portlet by dragging it from the Package Explorer and dropping it on your Liferay server.



WWW.LIFERAY.COM

 LIFERAY.

---

## INIT PHASE

- ❖ The *init()* method is called by the Liferay portlet container during the deployment of the portlet.
- ❖ During this phase of the portlet life cycle, portlets typically initialize any back end resources or perform *one-time* activities.
- ❖ Liferay portlets typically use the *init()* method to read initialization parameters from the *portlet.xml*.
- ❖ These initialization parameters are used to describe the page flow of the portlet.
- ❖ Because we've added some logging to this *DemoPortlet*, during the deployment and any subsequent re-deployment of this portlet, you should see a message in the Tomcat console indicating that the *init()* method has been called.

---

## RENDER PHASE

- ❖ During the Render Phase, the portlet generates content based on its current state.
- ❖ The Render Phase is called on all of the portlets on a page when that page is rendered.
- ❖ The Render Phase is also called when any of the portlets on that page complete the Action phase or Event Processing phase.
- ❖ Create a new page in the portal and add the *Demo portlet* to the page. You'll find it in the *Sample* category
- ❖ Watch the Tomcat console for message indicating the render method is being called.

---

## RENDER PHASE

- ❖ Portlets typically enter the Render Phase as a result of a page refresh or after the completion of the Action Phase. However, there times when you may want trigger the Render Phase directly.
- ❖ This can be accomplished by invoking a Render URL.
- ❖ Open the `/docroot/html/demo/view.jsp` file in Eclipse
- ❖ Find the following line and remove the comments
  - `<!--<p><a href="> . . . </p>-->`
- ❖ LDS automatically redeploys the portlet,
- ❖ Refresh the page, and click on the new link that is displayed.
- ❖ The portlet renders again, and you should see another render message.

---

## ACTION PHASE

- ❖ The portlet enters the Action Phase as a result of the user interacting with that portlet.
- ❖ Specifically, the user interaction should result in a change of state in the portlet.
- ❖ Only one portlet can go through the Action Phase during a single request/response cycle.
- ❖ Once the Action Phase has completed, the portlet will then process any Events that may have been triggered by the Action Phase.
- ❖ Once the Events have been processed or if there are no events triggered the portal will then call the Render Phase on all of the portlets on the page.

---

## ACTION PHASE

- ❖ Portlets enter the Action Phase by invoking an Action URL.
- ❖ Again, open the `/docroot/html/demo/view.jsp` file in Eclipse.
- ❖ Find the following line and remove the comments  
`<!--<p><a href="> . . . </p>-->`
- ❖ Save the file, refresh the page, and click on the new link that is displayed.

---

## EVENT PHASE

- ❖ The Event Phase is used to process any Events triggered during the Action phase of the portlet lifecycle.
- ❖ Events are used for Inter Portlet Communication (IPC).
- ❖ Events can, in turn, trigger additional events.
- ❖ Once all of the Events have been processed the portal will then call the Render Phase on all of the portlets on the page.
- ❖ We'll see an example of the Event Phase in the IPC Portlet exercise.

---

## RESOURCE SERVING

- ❖ The Resource Serving Phase allows portlets to serve dynamic content without the need calling the Render Phase on all of the portlets on the page.
- ❖ In Portlet 1.0, portlet requests always returned a full portal page.
- ❖ A Resource is requested via a ResourceURL.
- ❖ The bytes written to the response will be sent directly to the client.
- ❖ Resource serving is very useful for:
  - Creating images and other binary resources dynamically
  - Returning XML, JSON, HTML fragments for AJAX calls
  - Etc.

---

## RESOURCE SERVING PHASE

- ❖ Portlets enter the Resource Serving Phase by invoking a Resource URL.
- ❖ Again, open the `/docroot/html/demo/view.jsp` file in Eclipse
- ❖ Find the following line and remove the comments  
`<!--<p><a href="" onclick="loadXMLDoc . . . </p>-->`
- ❖ Save the file, refresh the page, and click on the new link that is displayed.

---

## DESTROY PHASE

- ❖ The *destroy()* method is called by the Liferay portlet container when it is removed from service.
- ❖ This phase of the portlet life cycle is designed to allow the portlet to release any resources and to save state if necessary.
- ❖ To trigger the *destroy()* method, we will need to undeploy the portlet.
- ❖ Because we've added some logging to our demo portlet, we can see a message when the portlet enters the *destroy()* method.

---

## PORTLET CHARACTERISTICS

- ❖ Portlets have additional characteristics that make them different from Servlets
- ❖ Portlet Modes
- ❖ Window States
- ❖ Portlet Preferences
- ❖ Standard Inter Portlet Communication (IPC)
  - Public render parameters
  - Events
- ❖ Resource Serving
  - AJAX support
  - Binary data support
- ❖ Portlet Filters

---

## PORTLET MODES

- ❖ Each portlet has a current mode, which indicates the function the portlet is performing.
- ❖ All Java Standards compliant portals must support the View, Edit and Help modes.
- ❖ Portlet modes are defined in the portlet.xml file.
- ❖ Custom modes may be created by developers.

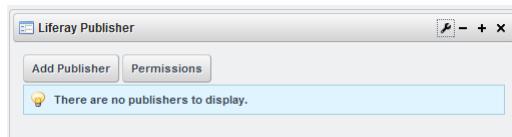
WWW.LIFERAY.COM

 LIFERAY.

---

## VIEW MODE

- ❖ The screenshot below represents a typical portlet when it is first rendered in View Mode:

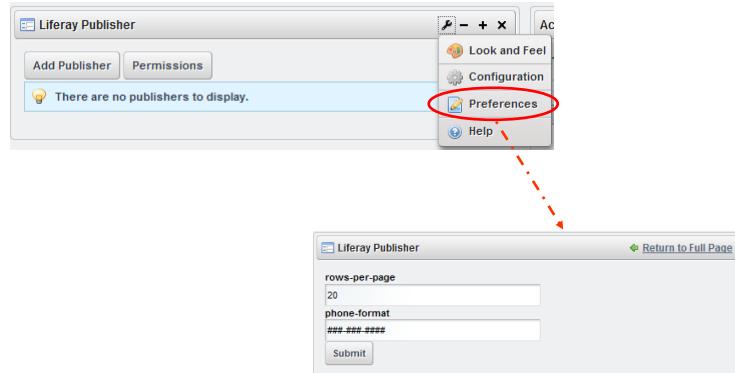


WWW.LIFERAY.COM

 LIFERAY.

## EDIT MODE

- When the user clicks on the “Preferences” icon, the portlet switches to Edit Mode:



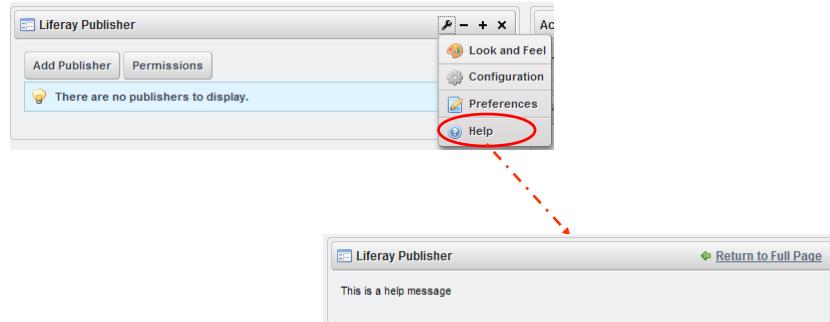
## EDIT MODE (PORTLET PREFERENCES)

- Portlets can be configured to provide a custom view or behavior for different users.
- For example, a weather portlet can show the temperature in Chicago for one user and the temperature in LA for another user.
- These configurations are represented as a persistent set of name-value pairs and are referred to as portlet preferences.

---

## HELP MODE

- When the user clicks on the *Help* icon, the portlet switches to Help Mode:



WWW.LIFERAY.COM

LIFERAY.

---

## LIFERAY PORTLETS

- Liferay allows for two additional descriptors named *liferay-portlet.xml* and *liferay-display.xml* that extend the features of *portlet.xml*. They are optional, but allow for Liferay-specific features to be implemented.
- You can optionally provide a *liferay-plugin-package.properties* file in order to specify the version(s) of Liferay that are known to be compatible with the portlet as well as other options.

WWW.LIFERAY.COM

LIFERAY.

---

## SAMPLE LIFERAY-PORTLET.XML

```
<?xml version="1.0"?>
<!DOCTYPE liferay-portlet-app PUBLIC "-//Liferay//DTD Portlet
Application 6.0.0//EN" "http://www.liferay.com/dtd/liferay-
portlet-app_6_0_0.dtd">
<liferay-portlet-app>
 <portlet>
 <portlet-name>demo-portlet</portlet-name>
 <instanceable>true</instanceable>
 <render-weight>1</render-weight>
 <ajaxable>false</ajaxable>
 </portlet>
 <role-mapper>
 <role-name>power-user</role-name>
 <role-link>Power User</role-link>
 </role-mapper>
 <role-mapper>
 <role-name>user</role-name>
 <role-link>User</role-link>
 </role-mapper>
</liferay-portlet-app>
```

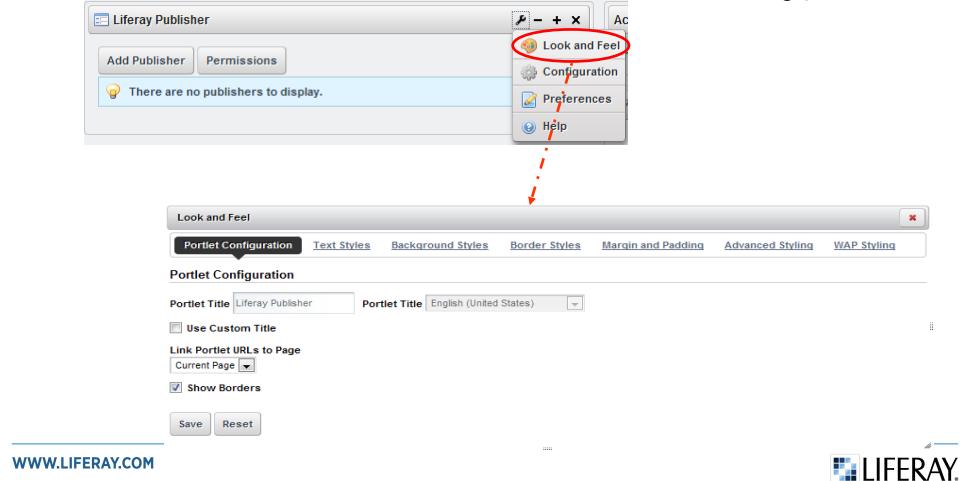
---

## SAMPLE LIFERAY-DISPLAY.XML

```
<?xml version="1.0"?>
<!DOCTYPE display PUBLIC "-//Liferay//DTD Display 6.0.0//EN"
"http://www.liferay.com/dtd/liferay-display_6_0_0.dtd">
<display>
 <category name="category.sample">
 <portlet id="demo-portlet" />
 </category>
</display>
```

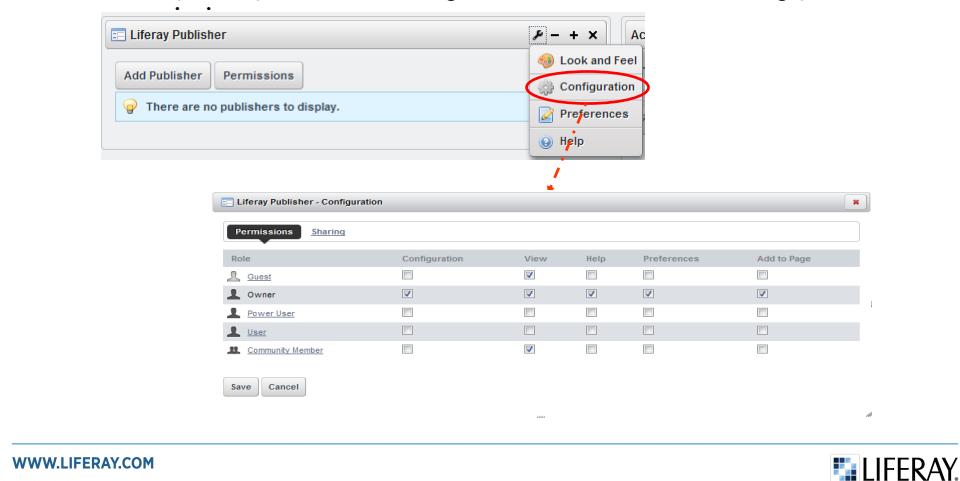
## LOOK AND FEEL MODE (LIFERAY EXTENSION)

- ❖ Liferay also provides a Look and Feel icon for customizing portlet



## CONFIGURATION MODE (LIFERAY EXTENSION)

- ❖ Liferay also provides a Configuration icon for customizing portlet



---

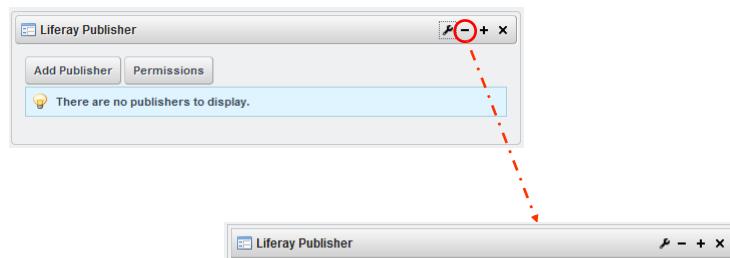
## WINDOW STATES

- ❖ Window states indicate the amount of space that will be assigned to a portlet.
- ❖ All Java Standards compliant portals must support *minimized*, *maximized* and *normal*.

---

## MINIMIZED WINDOW STATE

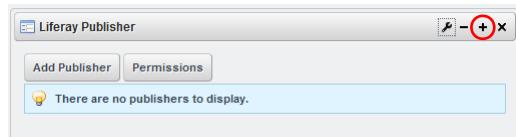
- ❖ When the user clicks on the *Minimize* icon, only the portlet titlebar is displayed:



---

## MAXIMIZED WINDOW STATE

- When the user clicks on the *Maximize* icon, the portlet will take up the entire width of the page, and become the only portlet rendered on the page:



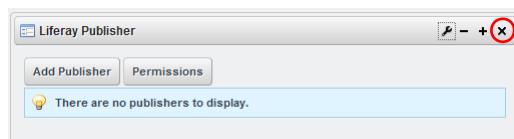
WWW.LIFERAY.COM

LIFERAY.

---

## REMOVE WINDOW (LIFERAY EXTENSION)

- When the user clicks on the *Remove* icon, the portlet is removed from the page:



WWW.LIFERAY.COM

LIFERAY.

---

## PORLET FILTERS

- ❖ Filters are Java components that allow on the fly transformations of information in both the request to and the response from a portlet.
- ❖ They allow chaining reusable functionality before or after any phase of the portlet Lifecycle:
  - processAction
  - processEvent
  - serveResource
  - render
- ❖ Modeled after the filters of the Servlet specification

---

## OTHER

- ❖ Better caching control: Etags, resource caching, ...
- ❖ Better servlet introspection
  - Very useful for framework developers
- ❖ Support for writing HTTP headers
- ❖ Annotations to simplify configuration
- ❖ P3P Profile attributes (The Platform for Privacy Preferences Project (P3P) enables Websites to express their privacy practices in a standard format that can be retrieved automatically and interpreted easily by user agents.)

---

## FINAL THOUGHTS

- ❖ Portlets are becoming increasingly important as the demand increases for aggregation of value-added services.
- ❖ Just as JSR-154 is the standard for servlets, JSR-168 (Portlet 1.0) and JSR-286 (Portlet 2.0) are the standards for portlets.

---

## FINAL THOUGHTS

- ❖ All of the major products including BEA/Oracle Weblogic Portal, IBM WebSphere Portal, Oracle Portal, JBoss Portal, and Sun OpenPortal adhere to the Java Portlet specification, either 1.0 or 2.0.
- ❖ Liferay Portal 4.x adheres to Portlet 1.0 (JSR-168).
- ❖ Liferay Portal 5.x and above adheres to Portlet 2.0 (JSR-286).
- ❖ Any knowledge that you gain regarding the Java standard portlet specification will be applicable to any commercial or open source portal available today.

---

## REFERENCES

- ❖ Java Portlet Specification
- ❖ <http://jcp.org/aboutJava/communityprocess/final/jsr286/index.html>

Notes: \_\_\_\_\_



# JAVA STANDARD PORTLET EXERCISE

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

## MY HELLO WORLD PORTLET

- ❖ In this exercise, you use the Liferay IDE to create a new portlet.
- ❖ Initially, this portlet will display a simple JSP.
- ❖ You will modify this portlet so that it accepts a name, stores it in PortletPreferences, and displays it.

## CREATE NEW LIFERAY PLUGIN PROJECT

- ❖ Go to File > New... > Liferay Project
- ❖ Project name:
  - my-hello-world
- ❖ Display name:
  - My Hello World
- ❖ Select the SDK and Liferay runtime you have configured
- ❖ Select the plug-in type (portlet is default)
- ❖ Click *Next*



WWW.LIFERAY.COM

LIFERAY.

## CREATE NEW LIFERAY PLUGIN PROJECT

- ❖ On this page you can choose a framework to use for the project.
- ❖ Leave the defaults, and check *Create custom portlet class*.
- ❖ Click *Next*.

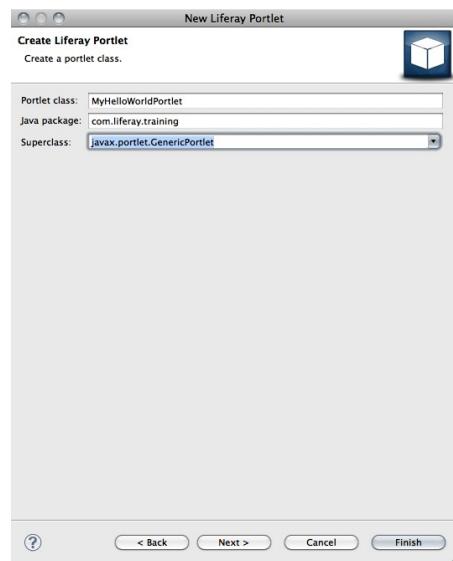


WWW.LIFERAY.COM

LIFERAY.

## CREATE NEW LIFERAY PORTLET

- ❖ Enter the following
  - Portlet class: MyHelloWorldPortlet
  - Java package: com.liferay.training
  - SuperClass: javax.portlet.GenericPortlet
- ❖ Click *Next*

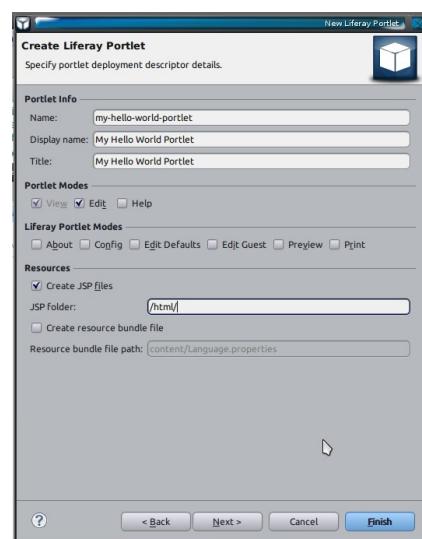


WWW.LIFERAY.COM

LIFERAY.

## SPECIFY PORTLET DEPLOYMENT DESCRIPTOR

- ❖ Enter the following
  - Name: my-hello-world-portlet
  - Display name: My Hello World Portlet
  - Title: My Hello World Portlet
- ❖ Check *Edit* check box
- ❖ Change JSP folder to: /html/
- ❖ Click *Next*

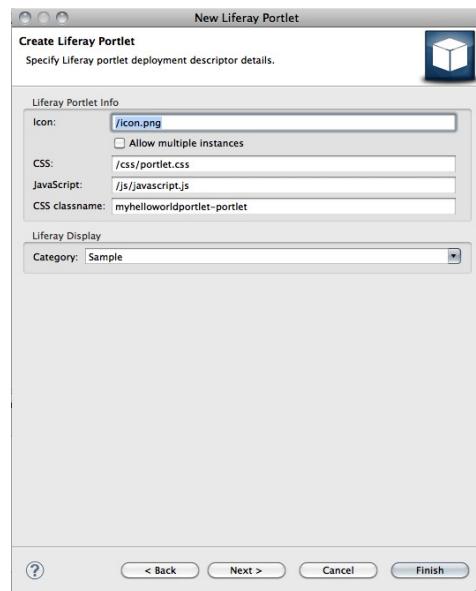


WWW.LIFERAY.COM

LIFERAY.

## SPECIFY LIFERAY PORTLET DEPLOYMENT DESCRIPTOR

- ❖ Modify the CSS classname to conform to a unique namespace for your portlet.
- ❖ Click *Next*

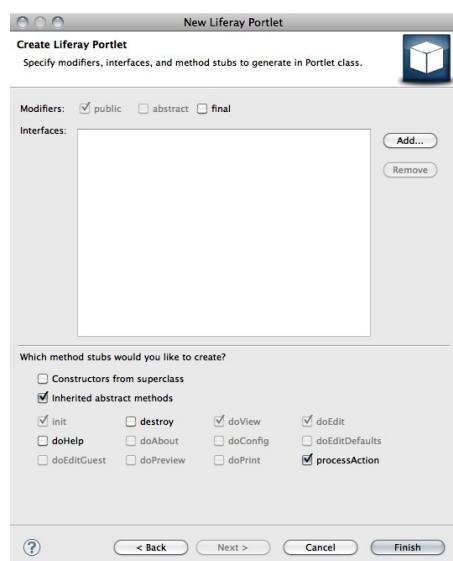


WWW.LIFERAY.COM

LIFERAY.

## SPECIFY ADDITIONAL DETAILS

- ❖ Place a check mark in the *processAction* check box
- ❖ Click *Finish*



WWW.LIFERAY.COM

LIFERAY.

---

## CHECK POINT

- ❖ The Liferay Developer Studio Wizard created the shell of our project which includes:
  - MyHelloWorldPortlet class which extends GenericPortlet
  - JSR-286 deployment descriptor (portlet.xml)
  - Liferay specific deployment descriptor (liferay-portlet.xml)
  - Additional Liferay specific files (liferay-display.xml,.liferay-plugin-package.properties)
  - View and edit jsp files
- ❖ This is a valid portlet that could be deployed to the portal, but at this point it only displays static view and edit modes.
- ❖ Next we will add our own logic to our portlet class.

---

## PORTLET PREFERENCES

- ❖ As defined in the specification, portals support a mechanism to store basic configuration data for portlets called *portlet preferences*.
- ❖ *Portlet preferences* are name-value pairs that are stored by the portal and available through the *PortletPreferences* interface.
- ❖ In this exercise, you will be enhancing the static view page to display a name that's been entered by the end user and stored as a *portlet preference*.
- ❖ You will also be enhancing the static edit page to display a form to capture the display name and persist it as a *portlet preference*.

## THE processAction METHOD

- ❖ First, modify the *processAction* method that is called when the form is submitted.
- ❖ Open the *MyHelloWorldPortlet.java* file and navigate to the *processAction* method.
- ❖ Replace the existing *processAction* method with the following code. You'll find it in the first snippet for this exercise in your IDE.

```
public void processAction(ActionRequest actionRequest, ActionResponse actionResponse)
 throws IOException, PortletException {

 String name = actionRequest.getParameter("name");
 PortletPreferences prefs = actionRequest.getPreferences();
 prefs.setValue("NAME", name);
 prefs.store();
 actionResponse.setPortletMode(PortletMode.VIEW);
}
```

❖ You also need to add the following import statements:

```
import javax.portlet.PortletPreferences;
import javax.portlet.PortletMode;
```

## MODIFY doView in MyHelloWorldPortlet.java

- ❖ *doView* is executed when we enter the view mode, *doEdit* when we enter the edit mode and *doHelp* when we enter the help mode (through the portlet icons).
- ❖ Replace the existing *doView* method with the following code (*03-doView()*):

```
public void doView(RenderRequest renderRequest, RenderResponse
 renderResponse)
 throws IOException, PortletException {

 PortletPreferences prefs = renderRequest.getPreferences();
 String defaultValue = "";
 String name = (String) prefs.getValue("NAME", defaultValue);
 renderRequest.setAttribute("ATTRIBUTE_NAME", name);
 include(viewJSP, renderRequest, renderResponse);
}
```

---

## CREATE edit.jsp

- ❖ Modify the *docroot/html/edit.jsp* file that was created by the wizard (o2-edit-jsp):

```
<%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %>

<form action="" method="post" name=">fm">
<label for="name">Name:</label>
<input name=">name" type="text" />

<input type="submit" />
</form>
```

---

## MODIFY view.jsp

- ❖ Change the content of the *view.jsp* in the *docroot* folder (o4-view-jsp):

```
<%
String name = (String)request.getAttribute("ATTRIBUTE_NAME");
%>

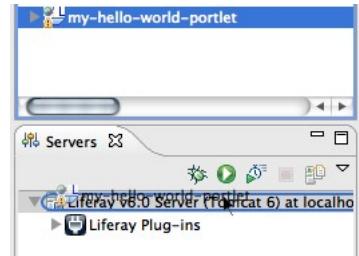
<p>This is the Hello World portlet.</p>

<p>Hello <%= name %>!</p>
```

## DEPLOY THE PROJECT

- ❖ To deploy the project, click on the root of the project and drag it from the package explorer onto the your Liferay runtime in the server view.
- ❖ Start Liferay, look for the following:

```
20:08:01,625 INFO [PluginPackageHotDeployListener:74] Reading plugin package
for hello-world-portlet
20:08:01,636 INFO [PluginPackageHotDeployListener:187] Plugin package
liferay/hello-world-portlet/4.4.2.1/war registered successfully
20:08:01,638 INFO [PortletHotDeployListener:130] Registering portlets for
hello-world-portlet
20:08:01,673 INFO [PortletHotDeployListener:393] Portlets for hello-world-
portlet registered successfully
```

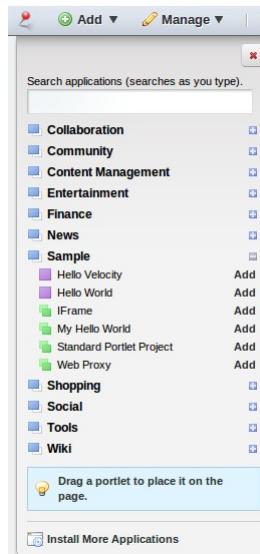


## OTHER DEPLOY OPTIONS

- ❖ Under the Ant view, you will see a variety of options for building your project. We'll use some of these later in the course or for troubleshooting issues.
- ❖ Click *Window -> Show View* and pick the Ant view.
- ❖ Open the Ant view and drag the *build.xml* file from the Navigator to the Ant view.
- ❖ You'll now be able to access the other build options through this view.

## ADD THE PORTLET

- Once Liferay has Point your web browser at *localhost:8080*.
- Login to the portal.
  - Username: *test@liferay.com*
  - Password: *test*
- On any page, click *Add* in the dock bar and select *More...*
- Open the *Sample* category
- Click the *Add* button next to the *My Hello World* portlet



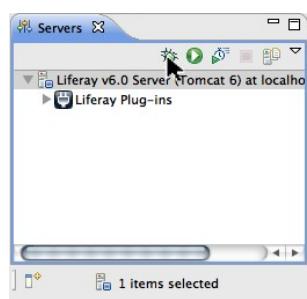
WWW.LIFERAY.COM

LIFERAY

## SET A BREAKPOINT

- In *MyHelloWorldPortlet.java*, set a breakpoint by double-clicking in the gutter on the left side of the editor in the *processAction* method and restart the server in debugger mode.

```
public void processAction(ActionRequest request, ActionResponse response) throws IOException, PortletException {
 String name = request.getParameter("name");
 PortletPreferences prefs = request.getPreferences();
 prefs.setValue("NAME", name);
 prefs.store();
 response.setPortletMode(PortletMode.VIEW);
}
```



WWW.LIFERAY.COM

LIFERAY

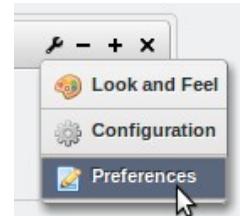
---

## USE THE PORTLET'S EDIT MODE

- In the upper right hand corner of the My Hello World portlet window, click the Configuration icon



- Then click Preferences



WWW.LIFERAY.COM

LIFERAY.

---

## SUBMIT THE FORM

- The form you created in your edit.jsp will be displayed. Put in your name and click Add Name.

A screenshot of the "My Hello World" portlet. It contains a single text input field with the placeholder "Name:" and the value "Joe Blogs". Below the input field is a button labeled "Add Name". At the top of the portlet, there is a "Return to Full Page" link.

WWW.LIFERAY.COM

LIFERAY.

## DEBUGGER

- The Debugger will be displayed
- You can view the variables and step through the code as it executes
- Hit the Play button to continue

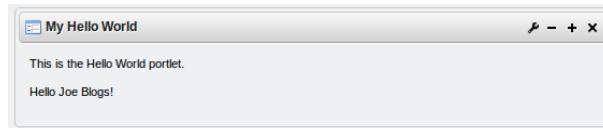
```
89 }
90 req.setAttribute("userName", username);
91 include(viewJSP, req, res);
92 }
93
94
95 public void processAction(ActionRequest req, ActionResponse res)
96 throws IOException, PortletException {
97 String addName = req.getParameter("addName");
98 if (addName!=null) {
99 PortletPreferences prefs = req.getPreferences();
100 prefs.setValue("name",req.getParameter("username"));
101 prefs.store();
102 res.setPortletMode(PortletMode.VIEW);
103 }
```

WWW.LIFERAY.COM

LIFERAY.

## IT WORKS!

- The name you entered should now be displayed in your portlet



WWW.LIFERAY.COM

LIFERAY.

**Notes:** \_\_\_\_\_



# LIFERAY. **INTER-PORTLET COMMUNICATION IN PORTLET 2.0**

Copyright © 2008-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated,  
copied, sold, resold, or otherwise exploited for any commercial purpose  
without express written consent of Liferay, Inc.

---

## **PORTLET 2.0 INTER-PORTLET COMMUNICATION**

- ❖ A serious limitation in Portlet 1.0 was the absence of a standard method for portlets to communicate with each other.
- ❖ To address the issue, various portal vendors (including Liferay) came up with means of overcoming this limitation.
- ❖ These implementations generally tied portlets to a particular vendor's portal.
- ❖ A standard way of doing IPC, therefore, was an important and much-anticipated feature.
- ❖ Portlet 2.0 (JSR-286) addresses the issue with two different methods of IPC.

---

## WHY IS IPC IMPORTANT?

- IPC becomes important with *Portlet Applications*. These are applications which are composed of more than one portlet for their functionality.
- Consider an order invoice application with one portlet at the top the screen which allows users to search for customers.
- When a customer is selected in the top portlet, a portlet on the bottom of the screen shows a list of all of that customer's invoices.
- With IPC, this functionality can be done in a standard way—and the two portlets don't even need to be on the same portlet page.

---

## PUBLIC RENDER PARAMETERS

- The simplest method for Standard Inter Portlet Communication
- A developer can declare a list of public parameters for a portlet application in portlet.xml:
- The parameter names are namespaced to avoid naming conflicts

```
<portlet-app>
 <public-render-parameter>
 <identifier>foo</identifier>
 <qname xmlns:x="http://foo.com/p">x:foo2</qname>
 </public-render-parameter>
 ...
</portlet-app>
```

---

## PUBLIC RENDER PARAMETERS (Cont.)

- Portlets must declare which public params they want to read by using the supported-public-render-parameter element:
- Those parameters will be available through all the lifecycle (processAction, processEvent, render, serveResource) of the portlet.

```
<portlet>
 <portlet-name>portletA</portlet-name>
 ...
 <supported-public-render-parameter>foo</supported-public-
 render-parameter>
</portlet>
```

---

## PUBLIC RENDER PARAMETERS (Cont.)

- A portlet can read a public render parameter by using:  
`request.getPublicParameterMap()`
- Public parameters are merged with regular parameters so can also be read using  
`getParameter(name)` and `getParameterMap()`
- A portlet can remove a public render param by invoking:  
`response.removePublicRenderParameter(name)`  
`portletURL.removePublicRenderParameter(name)`

---

## EVENTS

- ❖ Very powerful, highly decoupled method for Inter Portlet Communication
- ❖ Uses a Producer-Listener pattern
  - One portlet generates an event.
  - Zero or more portlets may be listening and acting upon it.
- ❖ Allows communication between portlets in different applications
- ❖ Additionally the container may also generate its own events
  - No specific container events have been standardized yet
- ❖ But beware of the added complexity!

---

## EVENTS (Cont.)

- ❖ Portlets can publish events from its processAction code using:  
`actionResponse.setEvent(event, eventPayload)`
- ❖ Publishing an event causes one or more invocations of the new processEvent method in this or other portlets.
- ❖ From the implementation of processEvents new events may also be published using:  
`eventResponse.setEvent(event, eventPayload)`
- ❖ Note that there is no guarantee in the order of delivery of events.

---

## EVENTS (Cont.)

- ❖ Sample event processing:

```
public void processEvent(
 EventRequest request, EventResponse response)
throws PortletException, IOException {
 String eventName = request.getEvent().getQName().toString();
 if (eventName.equals(...)) {
 ...
 } else if (eventName.equals(...)) {
 ...
 }
}
```

---

## EVENTS (Cont.)

- ❖ But it's cleaner and easier to use annotations (Requires inheriting from GenericPortlet)

```
/**
 * This method receives the event called "MyEvent"
 */
@ProcessEvent(qname={http://foo.com}MyEvent)
public void processMyEvent(
 EventRequest request, EventResponse response)
throws PortletException, IOException {
 ...
}
```

## A SIMPLE EXAMPLE

- ❖ We will go over a simple example of inter-portlet communication that consists of two portlets.
- ❖ Our example will simulate a segment of a baseball game: one portlet will “pitch” a ball, and another portlet will “catch” it.
- ❖ This will illustrate how you can send an event with a payload to another portlet, and that portlet can then perform some action on it.

WWW.LIFERAY.COM



## CREATE IPC BASEBALL PROJECT

- ❖ Go to File > New Project... > Liferay > Liferay Plug-in Project
- ❖ Project name:
  - > ipc-baseball
- ❖ Display name:
  - > IPC Baseball
- ❖ Select the SDK and Liferay runtime you have configured
- ❖ Select the plug-in type (portlet is default)
- ❖ Click *Next*



WWW.LIFERAY.COM



## CHOOSE A PORTLET FRAMEWORK

- Check *Create custom portlet class*.
- Leave the defaults.
- Click *Next* to continue.

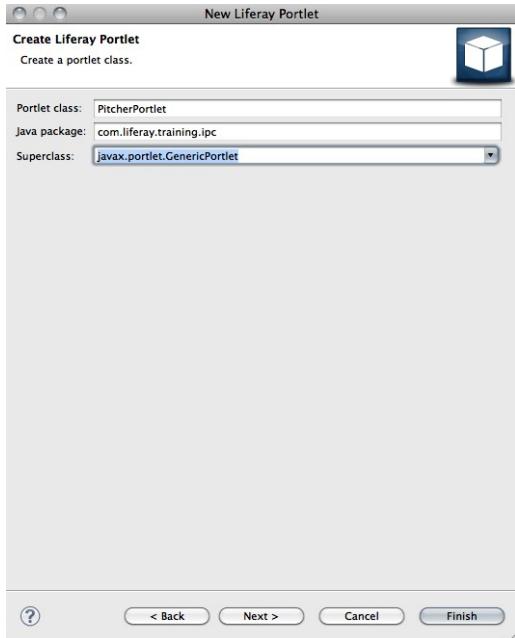


WWW.LIFERAY.COM

LIFERAY

## CREATE PITCHER PORTLET

- Enter the following
  - Portlet class: PitcherPortlet
  - Java package: com.liferay.training.ipc
  - SuperClass: javax.portlet.GenericPortlet
- Click *Next*

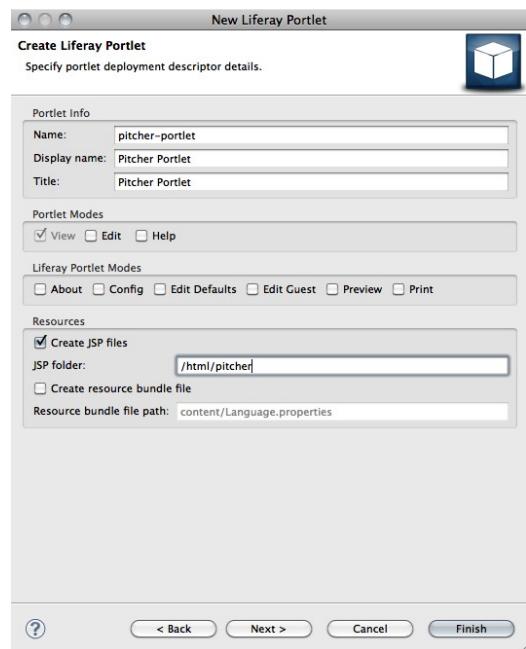


WWW.LIFERAY.COM

LIFERAY

## SPECIFY PORTLET DEPLOYMENT DESCRIPTOR

- ❖ Enter the following
  - Name: pitcher-portlet
  - Display name: Pitcher Portlet
  - Title: Pitcher Portlet
- ❖ Change JSP folder to: /html/pitcher
- ❖ Click Next

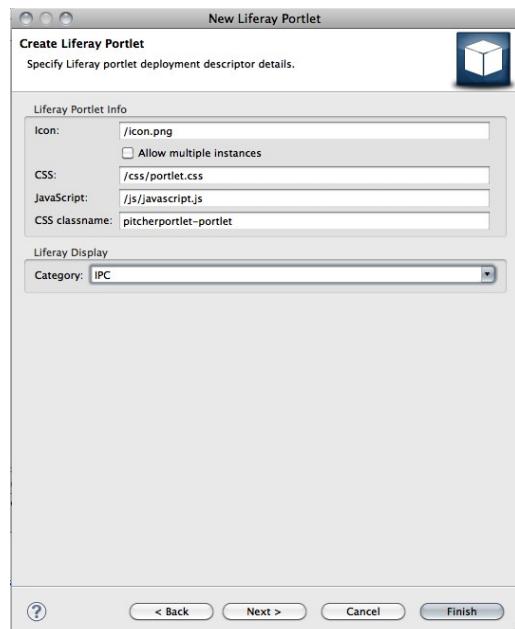


WWW.LIFERAY.COM

LIFERAY

## SPECIFY LIFERAY PORTLET DEPLOYMENT DESCRIPTOR

- ❖ Change the Category to IPC
- ❖ Click Next

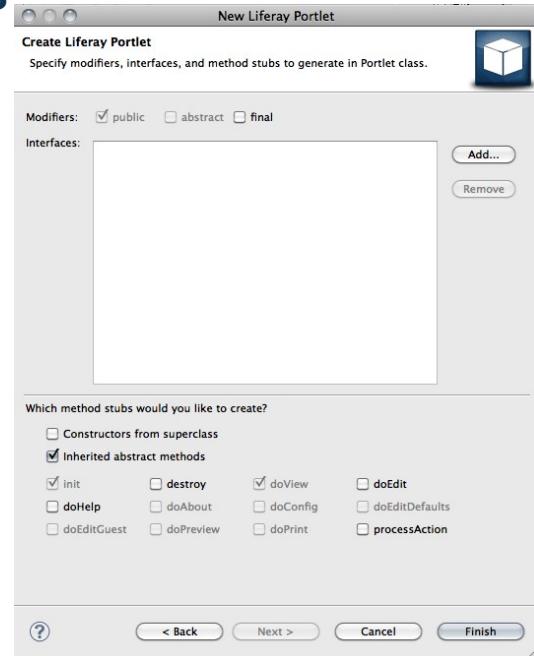


WWW.LIFERAY.COM

LIFERAY

## SPECIFY ADDITIONAL DETAILS

- Accept the Defaults and click *Finish*

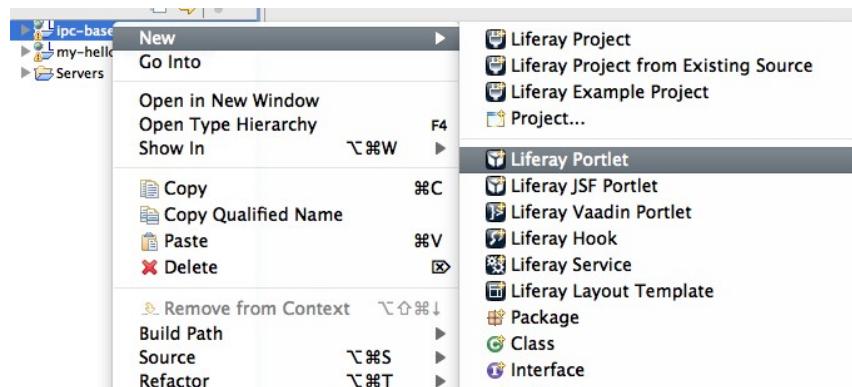


WWW.LIFERAY.COM

LIFERAY

## CREATING A NEW PORTLET IN THE PROJECT

- Right click on *ipc-baseball-portlet* in the package explorer.
- Select *New* → *Liferay Portlet*

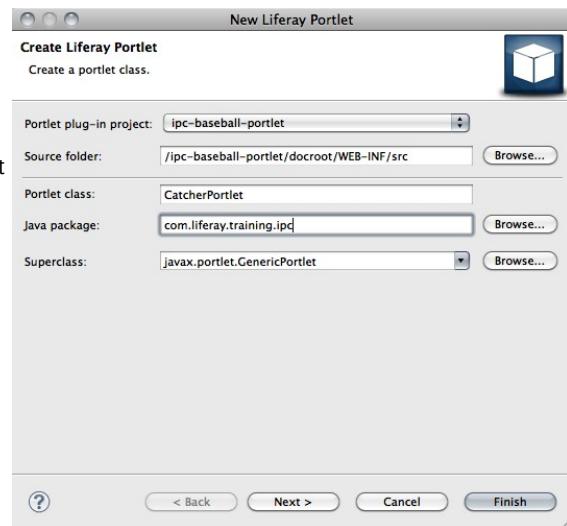


WWW.LIFERAY.COM

LIFERAY

## CREATE CATCHER PORTLET

- ❖ Enter the following
  - Portlet class: CatcherPortlet
  - Java package: com.liferay.training.ipc
  - SuperClass: javax.portlet.GenericPortlet
- ❖ Click *Next*

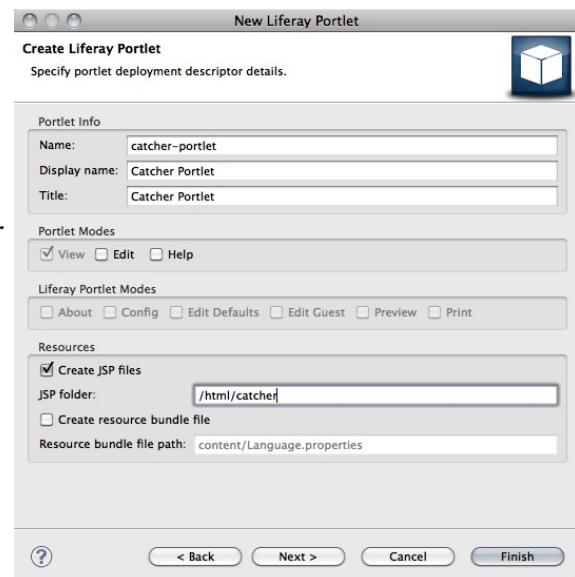


WWW.LIFERAY.COM

LIFERAY

## SPECIFY PORTLET DEPLOYMENT DESCRIPTOR

- ❖ Enter the following
  - Name: catcher-portlet
  - Display name: Catcher Portlet
  - Title: Catcher Portlet
- ❖ Change JSP folder to: /html/catcher
- ❖ Click *Next*

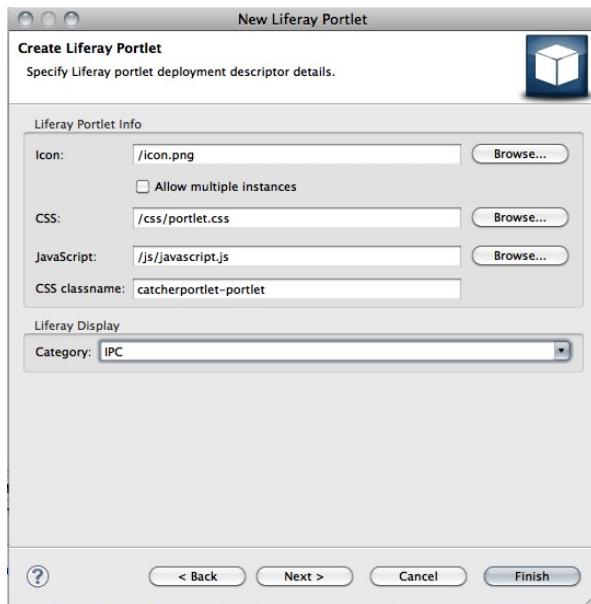


WWW.LIFERAY.COM

LIFERAY

## SPECIFY LIFERAY PORTLET DEPLOYMENT DESCRIPTOR

- Change the Category to IPC
- Click *Next*

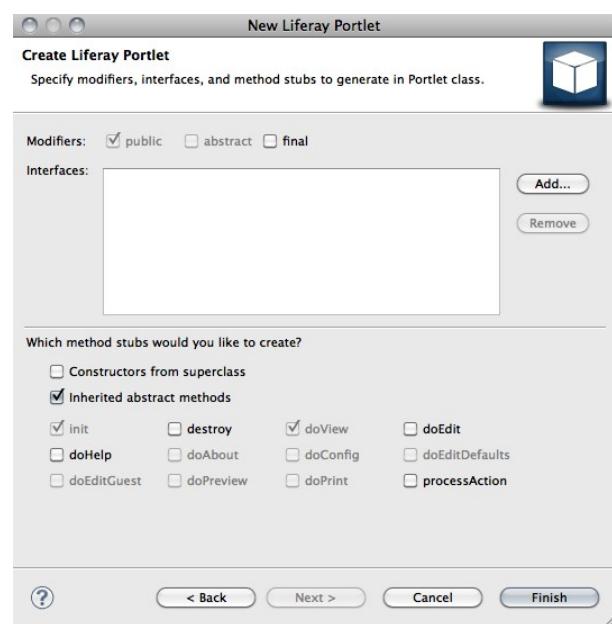


WWW.LIFERAY.COM

LIFERAY

## SPECIFY ADDITIONAL DETAILS

- Accept the Defaults and click *Finish*



WWW.LIFERAY.COM

LIFERAY

---

## MODIFY portlet.xml

- In order for our portlet application to participate in Event processing, we need to let the Liferay Portlet Container know which events we are interested in.
- Additionally, we need to add the **event** to our descriptor so Liferay knows which portlet can send the event and which can receive it.

---

## ADDING THE EVENT DEFINITION

- We do this by placing an event definition at the bottom of the portlet.xml file, outside of the two portlets.
- Our event will be a simple one: the PitcherPortlet will randomly generate a pitch, which will be placed into a String object.
- The object will be sent as the payload for our Pitch event.
- The PitcherPortlet will send the event, and the CatcherPortlet will receive it.
- First, we define the event itself, and then we define which portlet sends the event and which portlet receives it.

---

## THE EVENT DEFINITION

- Below the last portlet, **add** the following code (*o1-event-definitions*):

```
<event-definition>
 <qname xmlns:x="http://liferay.com/events">x:ipc.pitch</qname>
 <value-type>java.lang.String</value-type>
</event-definition>
```

- A QName is a *Qualified Name*.
  - Using a QName allows events to be namespaced properly so that no two event names are identical.
  - Here, we are defining an event called *ipc.pitch* within the name space <http://liferay.com/events>.
- Our payload for this event is a String containing the type of pitch that the PitcherPortlet generated.

---

## IDENTIFYING THE EVENT SENDER

- We need to be able to tell the portlet container (Liferay) which portlet sends the event.
- We have already defined this as the PitcherPortlet.
- **Add** the below code in the PitcherPortlet definition in *portlet.xml*, below the last *<security-role-ref>* tag (*o2-Publishing Event*):

```
<supported-publishing-event>
 <qname xmlns:x="http://liferay.com/events">x:ipc.pitch</qname>
</supported-publishing-event>
```

---

## IDENTIFYING THE EVENT RECEIVER

- ❖ One or more portlets can be configured to receive events. For our example, we have one: the CatcherPortlet.
- ❖ Add the following code below the last `<security-role-ref>` tag in the CatcherPortlet definition (*o3-Processing Event*):

```
<supported-processing-event>
 <qname xmlns:x="http://liferay.com/events">x:ipc.pitch</qname>
</supported-processing-event>
```

- ❖ Save the file.

---

## PitcherPortlet.java

- ❖ Next, we need to implement our pitch event.
- ❖ One of the nice things about Portlet 2.0 is GenericPortlet's reliance on Java SE 5 annotations.
  - These can be used to create methods which process specific actions without having to override the `processAction` method.
- ❖ The PitcherPortlet is simple: the JSP displays a URL to the Pitch action.
  - When a user clicks on the URL, the Pitch action is called.
  - This action generates a pitch and then publishes the pitch as an event.

---

## PITCH METHOD

- Add the contents of *o2-pitchball-method* into your PitcherPortlet.java file.
- Import all the needed classes:
  - import javax.portlet.ProcessAction;
  - import javax.portlet.ActionRequest;
  - import javax.portlet.ActionResponse;
  - import javax.xml.namespace.QName;
  - import java.util.Random;

---

## PITCH METHOD EXPLAINED

- The method starts with an annotation:  
`@ProcessAction(name="pitchBall")`
- Since we are extending GenericPortlet, we are making use of that portlet's processAction method. This calls any method with the same name as the Action name.
- Our pitchBall method then generates a random number between 1 and 3.
- Based on this number, a String representing the kind of pitch is created.
- We then send an event with the name we defined in portlet.xml containing the String payload of the type of pitch:  
`QName qName = new QName("http://liferay.com/events", "ipc.pitch");  
response.setEvent(qName, pitchType);`

---

## REMOVE processAction METHOD

- Because we want to use GenericPortlet's implementation of processAction, we told the Liferay IDE not to add the processAction method during the last step of the wizard.
- If you included the processAction method, you will need to remove it.
  - Highlight the processAction method and completely **remove** it.
    - This will cause us to use the implementation of processAction that is in the super class.

---

## CatcherPortlet.java

- Our CatcherPortlet doesn't do anything except receive the event sent to it by the PitcherPortlet.
- When it receives the event, it can examine the payload and then perform some action on it.
  - In our case, all CatcherPortlet does is display the type of pitch the PitcherPortlet generated.

---

## CATCH BALL METHOD

- As you did with the PitcherPortlet, **add** the following method below the include method in CatcherPortlet (o3-catchball-method):

```
@ProcessEvent(qname="{http://liferay.com/events}ipc.pitch")
public void catchBall(EventRequest request, EventResponse response) {
 Event event = request.getEvent();
 String pitch = (String)event.getValue();
 response.setRenderParameter("pitch", pitch);
}
```

- Import all of the needed classes:
  - import javax.portlet.Event;
  - import javax.portlet.EventRequest;
  - import javax.portlet.EventResponse;
  - import javax.portlet.ProcessEvent;

---

## CATCH BALL METHOD EXPLAINED

- We are using GenericPortlet's processEvent method to direct processing to our annotated method.
- In this case, the qname of the event is the parameter.
- The only processing we do is to take the value of the event payload and set it as a render parameter so it may be displayed to the user during the render phase of the portlet.
- Save the file.

---

## THE VIEW

- ❖ Our view for this application consists of two .jsp files that implement the View mode for each portlet.
- ❖ The Liferay IDE created two folders under the *docroot/html* folder of your project:
  - catcher
  - pitcher
- ❖ The Liferay IDE also created two *view.jsp* files in these folders.

---

## PITCHER VIEW

- ❖ Replace the contents of the *docroot/html/pitcher/view.jsp* file with *04-view-pitcher*:

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects />
<p>Click the link below to pitch the ball. </p>
</portlet:actionURL>">Pitch!
```

- ❖ As you can see, all this .jsp does is display a URL with the action name of *pitchBall* which maps to our method name annotation.
- ❖ Clicking this URL calls our action.

---

## CATCHER VIEW

- ❖ Replace the contents of the *docroot/html/catcher/view.jsp* file with *05-view-catcher*:

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects />
<%String pitch = (String)renderRequest.getParameter("pitch");%>
<p>And the pitch is....</p>
<p>
<% if (pitch!=null) { %>
<%=pitch %>!
<% } else { %>
... waiting for pitch.
<% } %>
```

- ❖ This .jsp displays the render parameter, or if it's null, displays a message that it's waiting for a pitch.

---

## OTHER DESCRIPTORS

- ❖ This is all that's necessary to deploy this project on any Java Standards compliant portlet container.
- ❖ However, the Liferay IDE created some extra files that describe additional Liferay extensions to the portlet specification.
- ❖ Open the following three files from *docroot/WEB-INF* folder of your project.
  - *liferay-portlet.xml* allows us to tell Liferay some extra things about the portlet, such as the location of CSS and JavaScript files, as well as the icon for the portlet.
  - *liferay-display.xml* allows us to place the portlet in a category of our choosing in the *Add Application* window.
  - *liferay-plugin-package.properties* allows us to define metadata about the portlet for Liferay's Software Catalog.

---

## DEPLOY THE PLUGIN

- ❖ Select the Servers tab from the bottom of the Eclipse Liferay Perspective and if the server is not running, start it now.
- ❖ Click and drag the project name from the Package Explorer to the server name in the Servers view.
- ❖ Watch the Console view to ensure the portlet deploys successfully. The portlet will be compiled and deployed.

---

WWW.LIFERAY.COM



---

## TEST THE PLUGIN

- ❖ Log into Liferay with the administrative credentials.
- ❖ From the Dock, click *Add Application*.
- ❖ From the *IPC* category, drag the Pitcher and Catcher portlets to the page.
- ❖ Click the *Pitch* link in the Pitcher portlet.
- ❖ The pitch event will be received (i.e., “caught”) by the Catcher Portlet.



---

WWW.LIFERAY.COM



---

## QUESTIONS?

- Bonus: Define another event to be thrown by the catcher when processing the event thrown by the pitcher and make the pitcher process this second event and display a different message.

Notes: \_\_\_\_\_



# PORTLETS AND WEB APP FRAMEWORKS

Copyright © 2000-2011 Liferay, Inc

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

## STANDARD PORTLETS

- The previous examples showed us how to use the standard Portlet API to create very simple portlets.
- As portlets get more complex, it is important to follow best practices for web application development, such as the separation of concerns.
- For proper separation of function from display, the developer must enforce good practices upon him or herself.
- Java portlets can be difficult to maintain once they get large.
- The Portlet 2.0 specification's use of annotations has made it easier to use the portlet as the controller for your application.
- Though many developers have been successful using this method, many others welcome the structure of a particular framework.
- Developers use frameworks for regular web applications; why not portlets?

---

## POPULAR FRAMEWORKS

- ❖ Liferay has sample portlets implemented using the following frameworks:
  - Struts
  - JSF
  - Spring Portlet MVC
  - Tapestry
- ❖ More are appearing all the time!

---

[WWW.LIFERAY.COM](http://WWW.LIFERAY.COM)



---

## STRUTS PORTLETS

- ❖ The Liferay Portal core and many of the out-of-the-box Liferay portlets are built with Struts, so even if you don't use Struts to develop new portlets, it is a good idea to become familiar with how it's used in Liferay.
- ❖ Struts implements MVC -- although there are other webapp frameworks that implement MVC, Struts is the most widely used and mature technology.
- ❖ What is MVC? MVC stands for Model-View-Controller: it is a design pattern that separates the presentation code from the business logic code.
- ❖ Struts provides centralized page-flow management in the form of struts-config.xml. This makes it highly scalable and allows you to modularize the coding process.

---

[WWW.LIFERAY.COM](http://WWW.LIFERAY.COM)



---

## STRUTS PORTLETS

- What are the main differences between a regular portlet and a Struts Portlet?
  - `struts-config.xml`
  - `tiles-defs.xml`
  
- Instead of forwarding directly to a JSP
  - `struts-config.xml` – define the page flow
  - `tiles-defs.xml` – define the page layout

---

## JSF PORTLETS

- JavaServer Faces (JSF) was designed in part by Craig McClannahan, the inventor of the Struts project.
- JSF is the standard MVC web application framework for Java.
- The JSR-127 (JSF 1.1) specification was designed with JSR-168 (Portlet 1.0) in mind.
- Often times JSF web applications can run as portlets with little to no modification.
- Liferay was one of the first portal vendors to provide support for JSF portlets back in May, 2005.

---

## SPRING PORTLET MVC

- ❖ Spring provides a portlet specific version of their popular Spring Web MVC application framework.
- ❖ The APIs and capabilities between the two frameworks are very similar, but unlike some other web application frameworks, the portlet MVC framework preserves the separation of Render and Action phases of the portlet life cycle.
- ❖ Spring Portlet MVC has been part of the core Spring Framework since Spring 2.0.
- ❖ Liferay v6.0 supports the use of Spring Portlet MVC and Service Builder in the same plugin project.

---

WWW.LIFERAY.COM



---

## LIFERAY MVC PORTLET

- ❖ With all of these established web application frameworks, why would Liferay try to develop another framework?
- ❖ Unlike many of the existing frameworks, the Liferay MVC Portlet framework is very light weight with a very minimal learning curve.
- ❖ There are no complicated XML configuration files that wire your portlet together.
- ❖ The Liferay MVCPortlet (the controller in our framework) extends GenericPortlet which you're already familiar with and allows you to access all of the underlying portlet functionality directly if you desire.

---

WWW.LIFERAY.COM



---

## LIFERAY MVC PORTLET

- ❖ We'll be using the `MVCPortlet` for the rest of our training examples, so let's start with a simple example to get our feet wet.
- ❖ Go to File > New Project... > Liferay > Liferay Project
- ❖ Project Name: `liferay-mvc-example`
- ❖ Display name: Liferay MVC Example
- ❖ Select the SDK and Liferay runtime and then select the plug-in type (portlet is default)
- ❖ This time, we'll leave the *Create custom portlet class* option unchecked.
- ❖ Leaving this unchecked creates a simple project that forwards to a JSP without the need for a portlet class.
- ❖ Click *Finish*.

---

## CHECKPOINT

- ❖ Let's take a quick inventory of what was created and configured.
- ❖ First, open the `docroot/WEB-INF/portlet.xml` and note the following have been defined:
  - `<portlet-name>`
  - `<display-name>`
  - `<portlet-class>`
- ❖ Review the `liferay-portlet.xml` and `liferay-display.xml`
- ❖ Review the `docroot/view.jsp`
- ❖ Deploy the portlet and add it to a page
- ❖ So far, things are pretty similar to the generic portlet example that we started with in the previous examples.
- ❖ Let's explore what makes the Liferay MVC Portlet different.

---

## LIFERAY MVC PORTLET

- ❖ Liferay's MVC Portlet is designed to remove the repetitive boiler plate code that we saw when extending *GenericPortlet*
- ❖ One of the ways it does this is by leveraging predefined parameters.
- ❖ During the initialization phase of the portlet life cycle, the Liferay MVC Portlet checks for the following initialization parameters
  - jsp-path
  - view-jsp
  - edit-jsp
  - help-jsp
  - about-jsp
  - config-jsp
  - edit-defaults-jsp
  - edit-guest-jsp
  - preview-jsp
  - print-jsp

---

## LIFERAY MVC PORTLET

- ❖ During the *render* phase of the portlet lifecycle, the Liferay MVC Portlet also looks for a specific render parameter named *jspPage*.
- ❖ The *jspPage* value should be a valid jsp page, including the path.
  - ex. *jspPage = "/html/showEditPublisher.jsp"*
- ❖ If *jspPage* is defined within the *renderRequest*, then the portlet will dispatch to the *jspPage* directly.
- ❖ This means that if you're creating a portlet whose sole purpose is to display data, you don't necessarily need to create a portlet class.
- ❖ The included *com.liferay.util.bridges.mvc.MVCPortlet* class provides all the portlet functionality you need and you're free to spend your time developing the *view* layer of your application.

---

## LIFERAY MVC PORTLET - EXERCISE

- ❖ Let's start with a simple example to show how to control page flow with the Liferay MVC Portlet.

- ❖ Open the *docroot/view.jsp* file

- ❖ Add the following (*01-view*) to the bottom of the file.

```
<portlet:renderURL var="editUser">
 <portlet:param name="jspPage" value="/edit_user.jsp" />
</portlet:renderURL>
<p><a href="<%= editUser %>">Click here to edit user record</p>
```

- ❖ **Create** a new file in the *docroot* directory named *edit\_user.jsp*.

- ❖ **Insert** the contents of *02-edit\_user.jsp* into the file you just created and deploy the portlet.

- ❖ **Click** on the new link you created to see the edit user form displayed.

---

## LIFERAY MVC PORTLET

- ❖ If your portlet requires the *action* phase of the portlet lifecycle (and most portlets do), the Liferay MVC Portlet provides two options.

- ❖ Option 1, which is used for most use cases, requires you create own class that extends *MVCPortlet* and add your action methods to that class.

- ❖ Option 2, which is used for very large and complex portlet applications allows you to break out your *actions* into their own classes. By following a standard naming convention, the *MVCPortlet* handles invoking the appropriate action without the use of complex xml based configuration files.

- ❖ For this training, we will be concentrating on the first option.

---

## LIFERAY MVC PORTLET - EXERCISE

- ❖ Update the action URL on the edit\_user.jsp to have a *name* parameter.
- ❖ Create the MyMVCPortlet class that extends MVCPortlet
- ❖ Implement the action method
- ❖ Modify the view.jsp to display the data that was submitted

---

## LIFERAY MVC PORTLET - EXERCISE

- ❖ First we'll need to create our portlet class, which extends the MVCPortlet.
  - Right Click on the *docroot/WEB-INF/src* folder in Package Explorer and select *New → Class*
  - Enter *com.liferay.training.mvc* for the Package.
  - Enter *MyMVCPortlet* for the Name.
  - Click the Browse button next to Superclass and start typing MVCPortlet. Select the *com.liferay.util.bridges.mvc.MVCPortlet* from the list.
  - Click Finish.

---

## LIFERAY MVC PORTLET - EXERCISE

- ❖ Second, we'll need to add a processAction method that will contain our business logic for processing the submitted form.
  - Open MyMVCPortlet.java
  - Drag the *o3-processAction* snippets into the MyMVCPortlet class.
  - Save and close the file.
- ❖ Next, we need to modify the deployment descriptor to tell the portlet container we're using a custom portlet class.
  - Open *docroot/WEB-INF/portlet.xml*
  - Modify the <portlet-class> to be:  
com.liferay.training.mvc.MyMVCPortlet
  - Save and close the portlet.xml file.

---

## LIFERAY MVC PORTLET - EXERCISE

- ❖ Last, give the Action URL in *edit\_user.jsp* a name parameter.
- ❖ The name doesn't actually matter, since we only have one action in this portlet.
- ❖ It should look something like this:  
`<portlet:actionURL name="myname" />`

---

## LIFERAY MVC PORTLET - EXERCISE

- ❖ After saving all of your changes, wait for the portlet to finish deploying and then test out the form by entering and submitting some data.
- ❖ You should see a message in the system log to indicate a new employee has been added.
- ❖ This is a very simple example, but the MVCPortlet framework can easily handle more complex portlets with many different actions by using the name attribute in the Action URL.
- ❖ Annotations are not necessary with MVCPortlet. When using the *name* attribute in your Action URL, just be sure that the value for the *name* attribute matches the name of your action method.
- ❖ Note: We're using Liferay's MVC specific features now. This means that these portlets can only be deployed on Liferay.

WWW.LIFERAY.COM



---

## FINAL THOUGHTS

- ❖ As you see from this simple portlet, the Liferay MVC Portlet provides:
  - A simple *Controller* that can be easily extended for more complex applications
  - A *View* layer that leverages your existing knowledge of Java Server Pages
  - Complete freedom to implement your *Model* layer with any technology you choose, but as we'll see in our next example the framework works very nicely with a Model layer that's been generated with the Liferay Service Builder.
- ❖ However, the Liferay MVC Portlet might not be the right choice for every situation so it's important to evaluate your needs and the benefits provided by all of the application frameworks before making a choice.

WWW.LIFERAY.COM



**Notes:** \_\_\_\_\_



# LIBRARY APPLICATION INTRODUCTION

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

## OVERVIEW

- Library Project Overview
- Design Approach
- Architectural Approach
- MVC Frameworks
- Getting Started

---

## INTRODUCING THE LIBRARY PROJECT

- During the next several presentations, we will create a Library application to illustrate several key Liferay development concepts.
- As the application evolves through several iterations, we will explore the following topics:
  - Advanced JSP Based Portlets
  - Liferay's Service Builder
  - User Feedback
  - Localization
  - Validation
  - Liferay's Search Container
  - Permissions
  - Control Panel

---

WWW.LIFERAY.COM



---

## DESIGN APPROACH

- We have been asked to create an application to keep track of books and publishers for an organization's small library.
- This application will eventually be accessed by members of our Library community.
- The completed application will consist of two portlets: a Publisher portlet that will be used to add, edit, delete, and view publishers and a Books portlet that will be used to add, edit, delete, and view books.
- These portlets will be non-instanceable. They can only be added once to a page and they will share data within a community or organization.

---

WWW.LIFERAY.COM



## PUBLISHER PORTLET - FORM VIEW

The screenshot shows the 'Liferay Publisher' portlet in 'Form View'. The title bar says 'Liferay Publisher'. The main area is titled 'New Publisher'. It contains four input fields: 'Name' (Liferay Press), 'Email Address' (press@liferay.com), 'Website' (http://www.liferay.com), and 'Phone Number' ((123)123-1234). Below the fields are 'Save' and 'Cancel' buttons.

WWW.LIFERAY.COM



## PUBLISHER PORTLET - LIST VIEW

The screenshot shows the 'Liferay Publisher' portlet in 'List View'. The title bar says 'Liferay Publisher'. There are two tabs: 'Add Publisher' (selected) and 'Permissions'. A table lists two publishers:

Name	Email Address	Phone Number	Website	Actions
Liferay Press	press@liferay.com	123-123-1234	http://www.liferay.com	
Manning Publications Company	sales@manning.com	(123)123-1234	http://www.manning.com	

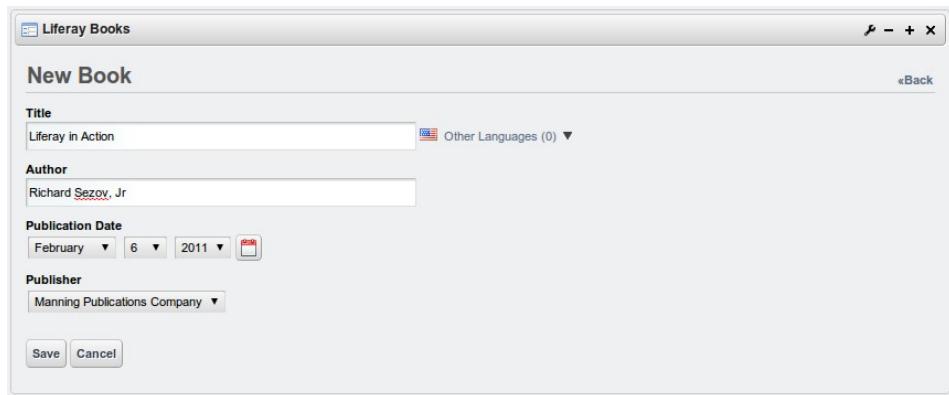
Showing 2 results.

WWW.LIFERAY.COM



---

## BOOKS PORTLET - FORM VIEW



Liferay Books

New Book

Title: Liferay in Action Other Languages (0)

Author: Richard Sezov, Jr.

Publication Date: February 6, 2011

Publisher: Manning Publications Company

Save Cancel

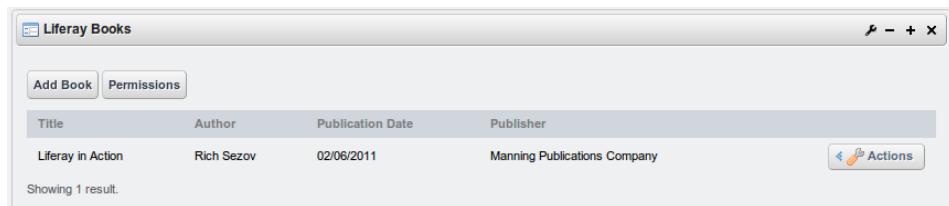
---

WWW.LIFERAY.COM



---

## BOOKS PORTLET - LIST VIEW



Liferay Books

Add Book Permissions

Title	Author	Publication Date	Publisher	Actions
Liferay in Action	Rich Sezov	02/06/2011	Manning Publications Company	

Showing 1 result.

---

WWW.LIFERAY.COM



---

## ARCHITECTURAL APPROACH

- It is a good practice to separate the various layers of the application: UI, model, persistence, etc. This is sometimes called Separation of Concerns.
- By keeping the layers as separate as possible, you gain the ability to change the implementation of any one layer more easily—if for some reason you find a better way to do it later.
- Model-View-Controller (MVC) is an architectural design pattern that separates the business logic concerns from the presentation concerns.
- There are many web frameworks that implement the MVC design pattern and many of these frameworks can be used for portlet based applications.
- For the Library application, we are leveraging the Liferay MVC Portlet to implement our own version of the MVC pattern using elements of the Portlet API.

---

## THE MODEL

- The model layer of the application holds all the data and any business rules for manipulating the data.
- Our Publisher and Book objects and all of their corresponding attributes will be part of the model layer.
- Any methods or logic that is used to manipulate these attributes are also part of the model layer.

---

## THE VIEW

- The view layer contains all of the logic used for displaying data to the users.
- Our view layer will handle the form elements as well as displaying the returned data.
- The Library application will make use of JSP for the view layer.

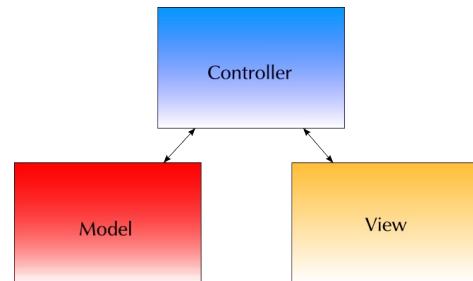
---

## THE CONTROLLER

- The controller layer acts as the traffic director, passing data back and forth to and from the model and view layers.
- Generally the model and view layers only talk to the controller layer.
- In some cases, the view layer may talk directly to the model layer for display purposes.
- In our Library application, the portlet classes act as the controller layers.

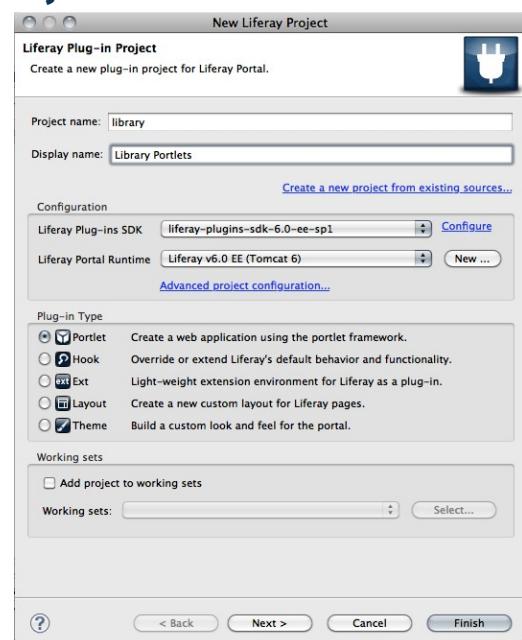
## MODEL-VIEW-CONTROLLER

- ❖ The View will be handled by JSPs with which users can interact. They will click buttons and submit forms.
- ❖ Our portlet classes are the Controllers. These pass data from the View layer to the Model layer. Based on feedback from the Model layer, the Controller determines which page in the View layer should be displayed next.
- ❖ The Model layer contains our business logic.



## CREATE LIBRARY PORTLETS PROJECT

- ❖ Go to File > New... > Liferay Project
- ❖ Project name:
  - library
- ❖ Display name:
  - Library Portlets
- ❖ Select the SDK and Liferay runtime you have configured
- ❖ Select the plug-in type (portlet is default)
- ❖ Click Next



## CHOOSE A FRAMEWORK

- Leave the portlet framework as the default.
- Check *Create custom portlet class*.

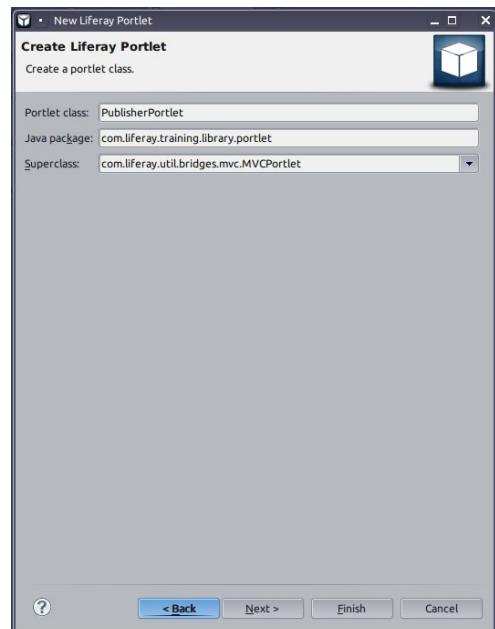


WWW.LIFERAY.COM

LIFERAY

## CREATE PUBLISHER PORTLET

- Enter the following
  - Portlet class: **PublisherPortlet**
  - Java package:  
**com.liferay.training.library.portlet**
  - SuperClass:  
**com.liferay.util.bridges.mvc.MVCPortlet**
- Click **Next**

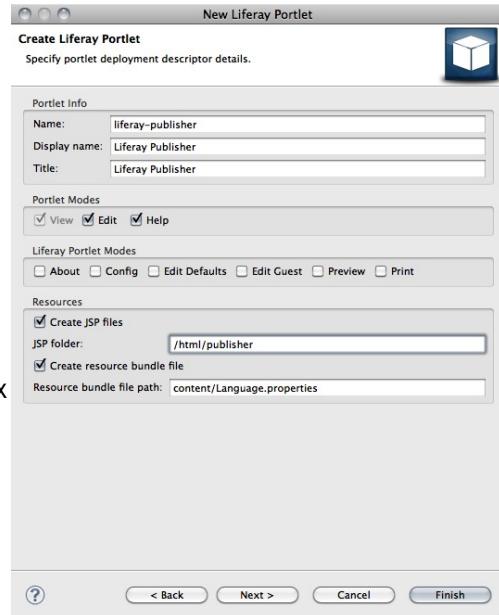


WWW.LIFERAY.COM

LIFERAY

## SPECIFY PORTLET DEPLOYMENT DESCRIPTOR

- ❖ Enter the following
  - Name: liferay-publisher
  - Display name: Liferay Publisher
  - Title: Liferay Publisher
- ❖ Check *Edit* and *Help* modes
- ❖ Change JSP folder to:
  - /html/publisher
- ❖ Check the *Create resource bundle file* box
- ❖ Accept the default *Resource bundle file path*
- ❖ Click *Next*

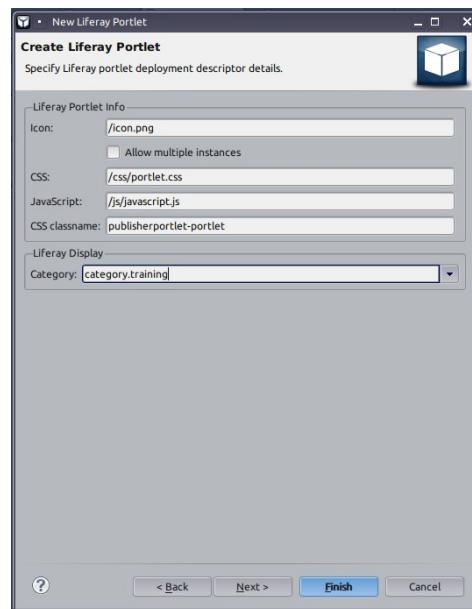


WWW.LIFERAY.COM

LIFERAY

## SPECIFY LIFERAY PORTLET DEPLOYMENT DESCRIPTOR

- ❖ Verify *Allow multiple instances* is unchecked.
- ❖ Change the Category to *category.training*
- ❖ Click *Next*

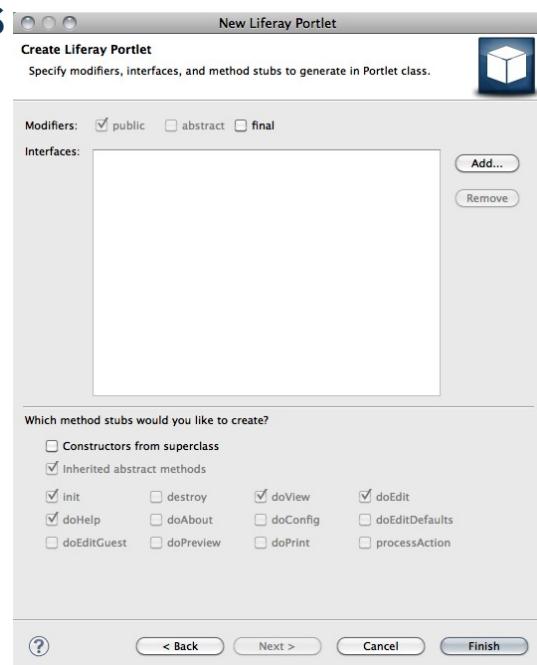


WWW.LIFERAY.COM

LIFERAY

## SPECIFY ADDITIONAL DETAILS

- ❖ Accept the Defaults and click *Finish*



WWW.LIFERAY.COM

LIFERAY

## CREATE BOOK PORTLET

- ❖ File → New → Liferay Portlet
- ❖ Enter the following
  - Portlet class: **BookPortlet**
  - Java package: **com.liferay.training.library.portlet**
  - SuperClass: **com.liferay.util.bridges.mvc.MVCPortlet**
- ❖ Click *Next*



WWW.LIFERAY.COM

LIFERAY

## SPECIFY PORTLET DEPLOYMENT DESCRIPTOR

- ❖ Enter the following
  - Name: liferay-book
  - Display name: Liferay Book
  - Title: Liferay Book
- ❖ Check *Edit* and *Help* modes
- ❖ Change JSP folder to:
  - /html/book
- ❖ Check the *Create resource bundle file* box
- ❖ Accept the default *Resource bundle file path*
- ❖ Click *Next*

The screenshot shows the 'Create Liferay Portlet' dialog box. It has tabs for 'Portlet Info', 'Portlet Modes', and 'Resources'. In 'Portlet Info', 'Name' is set to 'liferay-book', 'Display name' is 'Liferay Book', and 'Title' is also 'Liferay Book'. Under 'Portlet Modes', 'Edit' and 'Help' are checked. In 'Resources', 'Create JSP files' is checked, and 'JSP folder' is set to '/html/book'. 'Create resource bundle file' is also checked, and 'Resource bundle file path' is set to 'content/Language.properties'. At the bottom are buttons for '?', '< Back', 'Next >', 'Cancel', and 'Finish'.

WWW.LIFERAY.COM



## SPECIFY LIFERAY PORTLET DEPLOYMENT DESCRIPTOR

- ❖ Verify *Allow multiple instances* is unchecked.
- ❖ Change the Category to *category.training*
- ❖ Click *Next*

The screenshot shows the 'Create Liferay Portlet' dialog box. It has tabs for 'Liferay Portlet Info' and 'Liferay Display'. In 'Liferay Portlet Info', 'Name' is 'liferay-book', 'Icon' is '/icon.png', and 'CSS' is '/css/portlet.css'. 'JavaScript' is '/js/javascript.js' and 'CSS classname' is 'bookportlet-portlet'. Under 'Liferay Display', 'Category' is set to 'category.training'. At the bottom are buttons for '?', '< Back', 'Next >', 'Cancel', and 'Finish'.

WWW.LIFERAY.COM



## SPECIFY ADDITIONAL DETAILS

- ❖ Accept the Defaults and click *Finish*



WWW.LIFERAY.COM

LIFERAY

Notes:



# LIBRARY APPLICATION: BUILDING OUR SERVICE LAYER

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

## OBJECTIVE

- ❖ This exercise walks you through the steps necessary to use Liferay's Service Builder code generation tool in an MVC portlet.
- ❖ We will enhance the generated service layer with our own business logic and convenience methods.
- ❖ When done, the generated service layer is ready to hook up to the MVC UI.
- ❖ The snippets for the exercises in this presentation are in the *Training-05-Service Builder* category.

---

## LESSON OVERVIEW

- Liferay Portal Service Overview
- Setup WEB-INF/service.xml
- Run Service Builder
- Service and Persistence Design
- Modify Transfer Objects
- Enhance Service API

---

## WHAT IS A SERVICE?

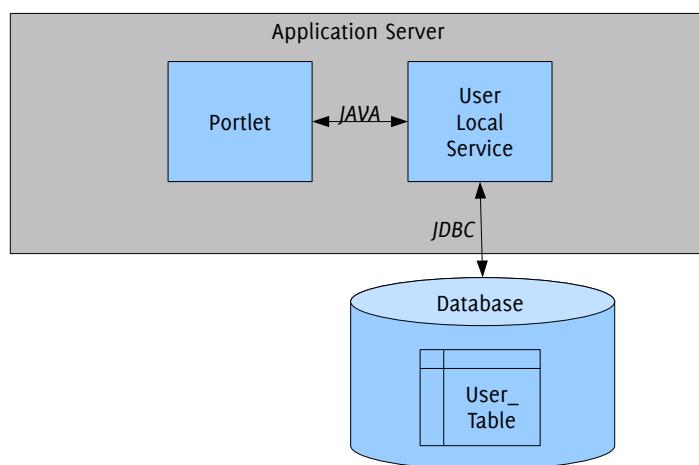
- Service is a term used in software engineering to define an object that performs one or more operations that can be invoked or consumed by an application.
- A service has an API that defines the contract to which the service and consuming application are bound.
- With respect to the Java programming language:
  - A service is typically defined as a Java “interface”
  - Service implementations are Java classes and are commonly referred to as “impl” classes
- With respect to Service Oriented Architecture (SOA):
  - Services are typically “Web Services”
  - Web service APIs are defined using the Web Services Description Language (WSDL)
  - Applications communicate with services via the Simple Object Access Protocol (SOAP)

## LIFERAY PORTAL SERVICES

- Liferay Portal ships with a robust set of services.
- Liferay portlets are “consumers” of Liferay services.
- Liferay services can be “local” or “remote.”
  - Local services are running on the same application server as the portal.
    - Example:
      - com.liferay.portal.service.UserLocalService (Java interface)
      - com.liferay.portal.service.impl.UserLocalServiceImpl (Java class)
  - Remote services are running on a different application server, and manifest themselves as **web services**.
    - Example:
      - com.liferay.portal.service.UserService (Java interface)
      - com.liferay.portal.service.impl.UserServiceImpl (Java class)

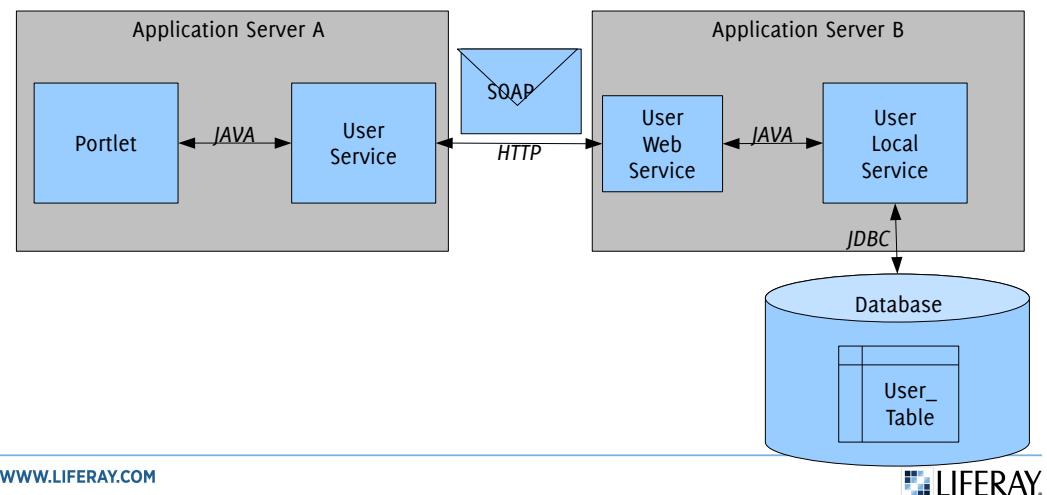
## LOCAL SERVICES

- The following diagram illustrates how a portlet makes direct Java calls to the *local* Liferay Portal **UserLocalService** to query data from the portal database.



## REMOTE SERVICES

- The following diagram illustrates how a portlet makes direct Java calls to the *remote Liferay Portal UserService*, which in turn communicates with a web service on a different application server via HTTP.



## LIFERAY SERVICE BUILDER

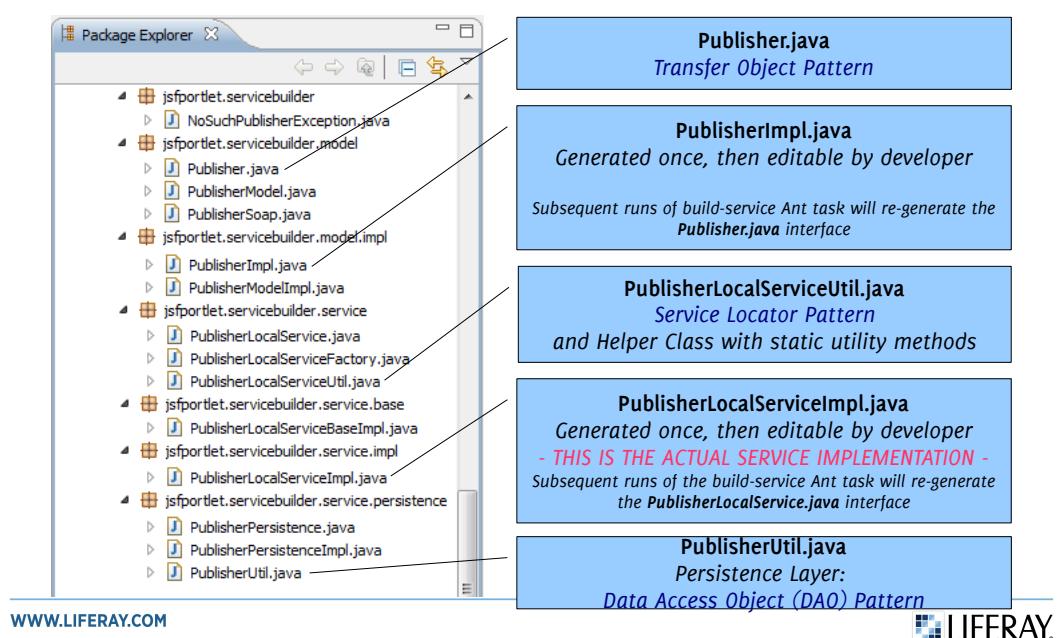
- Liferay Portal services are automatically built by a code generator called Service Builder.
- Services entities are *modeled* in a single XML file, named `WEB-INF/service.xml`
  - Example syntax:

```
<service-builder package-path="com.liferay.training">
 <namespace>Library</namespace>
 <entity name="Publisher" local-service="true" remote-service="false">
 <column name="publisherId" type="long" primary="true" />
 <column name="companyId" type="long" />
 <column name="groupId" type="long" />
 <column name="name" type="String" />
 <column name="emailAddress" type="String" />
 <column name="website" type="String" />
 <column name="phoneNumber" type="String" />
 <order by="asc">
 <order-column name="name" />
 </order>
 </entity>
</service-builder>
```

## USING SERVICE BUILDER

- Service Builder should be run in a Portlet or Hook Plugin. It can be run in an Ext Plugin but it is not recommended (it is deprecated).
- In the Plugins SDK, the `WEB-INF/service.xml` file must be configured.
- The code generator is invoked using the *build-service* Ant task.

## SERVICEBUILDER GENERATED CLASSES



---

## SERVICE LOCATORS: DEPENDENCY “PULL”

- When a consumer (like a portlet) needs a service, it can use a [service locator](#) to get an instance of the service.
- In the preceding example, `PublisherLocalServiceUtil.java` is the service locator, and also provides static utility methods for consuming the service.
- When a consumer goes through a service locator to acquire a service, *it is controlling its own dependencies*.
- Liferay services provide convenient service locators to acquire services, but this promotes a *tight coupling* between the consumer and the service.

---

## DEPENDENCY INJECTION

- **Dependency Injection (DI)** is a software engineering technique that *supplies* consumers with *services*, rather than the consumers themselves *requesting services* from a service locator.
- DI is a form of [Inversion of Control \(IOC\)](#).
- An [IOC Container](#) (also known as a *Bean Factory*) manages instances of services (beans).
- The most popular Java-based IOC container is the [Spring Framework](#), which manages Java class instances called “Spring Beans.”
- The Spring Framework also supplies [Aspect Oriented Programming \(AOP\)](#) features.
- Liferay Portal uses the Spring IOC container to manage services and Spring AOP to manage transactions.
- By using DI instead of acquiring a service with a service locator, *loose coupling* is achieved between the consumer and the service.

---

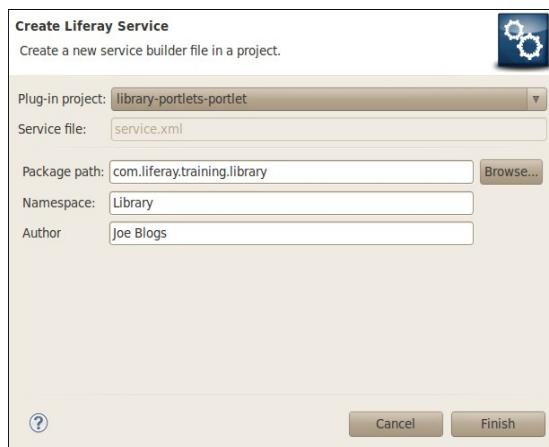
## DEPENDENCY INJECTION AND UNIT TESTING

- Besides *loose coupling* between consumers and services, another benefit of leveraging DI is to simplify [Unit Testing](#).
- Review:
  - Services APIs are defined by Java interfaces.
  - Service objects implement the Java interfaces, and are commonly referred to as “impl” classes.
- A common technique in unit testing is to create *mock* implementations of services.
- During automated execution of unit tests, the IOC container can inject mock implementations of services which are not dependent on runtime environments.

---

## CREATE LIFERAY SERVICE

- The Liferay IDE provides tools to help creating Liferay Services.
- Ensure the library-portlet project is selected in the Package Explorer and select File → New → Liferay Service
- Package Path:
  - com.liferay.training.library
- Namespace:
  - Library
- Author
  - Your name
- Click *Finish*



---

## WEB-INF/service.xml

- The Liferay IDE will create the WEB-INF/service.xml file and open it in the editor. Switch to the XML view and examine the file:
- Note the <author> and <namespace> tags at the beginning of the file.
- The <author> tag allows you to insert your name in the classes and interfaces that Service Builder will be creating. The tag is optional, but if you do not supply an author, a default value will be used.  

```
<author>Joe Bloggs</author>
```
- The <namespace> tag instructs Service Builder to create database tables that have their name prefixed with "Library\_"  

```
<namespace>Library</namespace>
```
- Although the *entities* will be named **Publisher** and **Book**, the database tables will be named **Library\_Publisher** and **Library\_Book** respectively
- This tag is mandatory.
- Additional information on required and optional tags can be found by reading the DTD file, found in the *definitions* folder of the portal source.

---

## WHAT CONSTITUTES A PUBLISHER?

Field Name	Field Type	Description
publisherId	long	Unique key for each entry (pk)
companyId	long	(fk)
groupId	long	(fk)
name	String	Name of publisher
emailAddress	String	Email address of publisher
website	String	Web Site of publisher
phoneNumber	String	Phone number of publisher

- Note that the Field Type refers to the Java type. Hibernate will map this value to the appropriate SQL type for us.
- The companyId corresponds to a Liferay Portal instance.
- The groupId corresponds to a Liferay community or organization.
- The companyId and groupId will be used to make our portlets non-instanceable.

---

## WEB-INF/service.xml (Library\_Publisher)

- Switch to the XML view of the service.xml file.
- Find the snippet called *o1-First Entity*.
- Replace the empty <entity> tag with the contents of that snippet in the WEB-INF/service.xml file.

---

## WEB-INF/service.xml (Local Services Only)

- Liferay services can be “local” or “remote”
  - For the sake of simplicity, this exercise will only ask Service Builder to generate *local* services, as indicated in the following XML fragment:
- ```
<entity name="Publisher" local-service="true" remote-service="false">
```

WHAT CONSTITUTES A BOOK?

Field Name	Field Type (Java Type)	Description
bookId	long	Unique key for each entry (pk)
company Id	long	(fk)
groupId	long	(fk)
publisherId	long	(fk)
title	String	Book title
authorName	String	Email address of publisher
publicationDate	Date	Web Site of publisher

WEB-INF/service.xml (Library_Book)

- Find the snippet called *o2-Second Entity*.
- Add the contents of this snippet after the closing `</entity>` tag for the Publisher entity and before the closing `</service-builder>` tag in the `WEB-INF/service.xml` file:

WEB-INF/service.xml (localized = true)

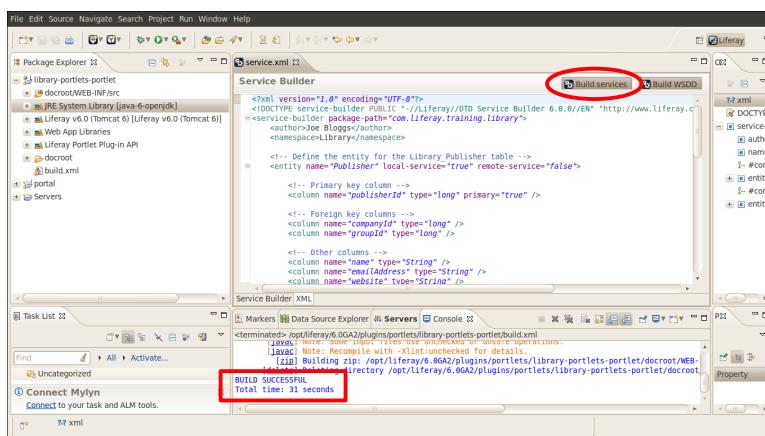
- The Book entity description, take a closer look at the line that defines the Book's title.
- The optional localized attribute is used to indicate whether or not the column can have different values for different locales.
- The default value is false, but we are overriding that and setting this to true.
- We will see the impact of this in later exercises.

WWW.LIFERAY.COM



RUN SERVICE BUILDER

- Run Liferay Service Builder by clicking on the *Build services* button.
- Watch the console for a message indicating *BUILD SUCCESSFUL*



WWW.LIFERAY.COM



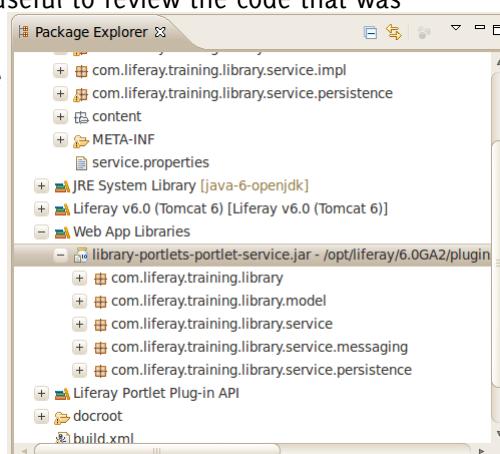
SERVICE BUILDER CONSOLE OUTPUT

- After running the build-service target, the Eclipse console log should contain the following:

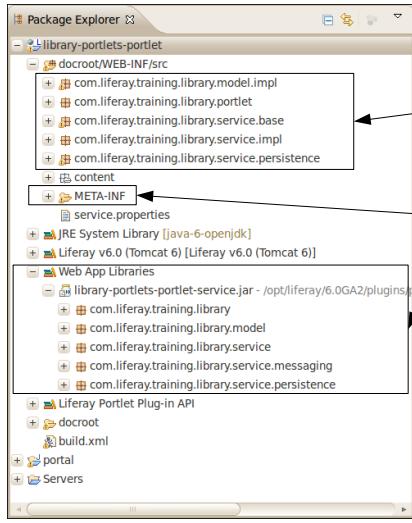
```
build-service:  
  [mkdir] Created dir: C:\liferay\plugins\portlets\library-part-a-portlet\docroot\WEB-INF\sql  
  ...  
  [java] Building Publisher  
  ...  
  [java] Writing docroot\WEB-INF\service\jsfportlet\servicebuilder\model\Publisher.java  
  ...  
  [java] Building Book  
  ...  
  [java] Writing docroot\WEB-INF\service\jsfportlet\servicebuilder\model\Book.java  
  ...  
BUILD SUCCESSFUL
```

ACCESSING LIFERAY SERVICE LAYER CODE

- The Liferay Service Builder has created new source code in the *docroot/WEB-INF/service* directory.
- The classes created in this directory are used by the Liferay Service Builder and should not be edited, but it may be useful to review the code that was generated.
- To prevent accidental editing of these files, the Liferay IDE makes them available as a jar file in the Web App Libraries.



SERVICE BUILDER GENERATED JAVA



Service Builder Generated Java Packages & Classes

Service Builder Spring & Hibernate Config

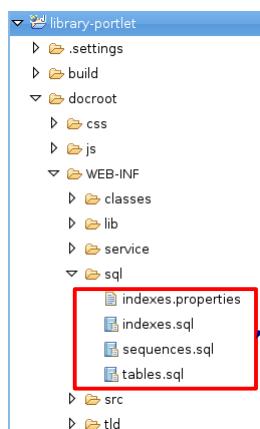
Did you know?

- On subsequent runs of the *build-service* task, Service Builder will generate interfaces by *reverse-engineering* implementations.
- When methods are added to implementation classes by the developer, the *build-service* task will regenerate the corresponding interface and add the new method signatures.

WWW.LIFERAY.COM

LIFERAY

SERVICE BUILDER GENERATED SQL



Service Builder generated SQL

Did you know?

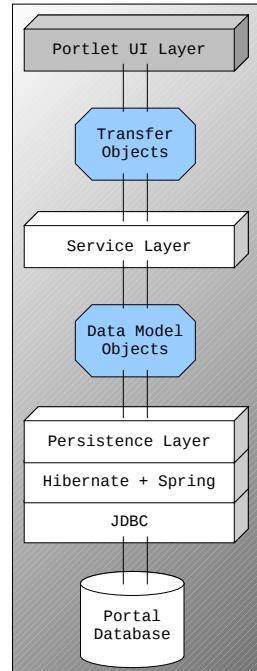
- At this point, Service Builder has generated the SQL that will be used to create the tables in the database.
- Once the Portlet has been deployed and placed on a page, the SQL statements will be executed in the database and the Library tables will be created.

WWW.LIFERAY.COM

LIFERAY

SERVICE AND PERSISTENCE DESIGN

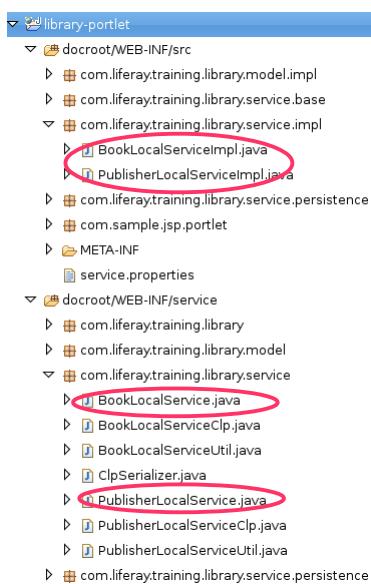
- ❖ The Service Builder generated persistence layer is currently built on top of **Hibernate**, and transactions are managed with **Spring AOP**
- ❖ When **Data Model Objects** are retrieved from the database by the persistence layer, the service layer *disconnects* them from the Hibernate session
- ❖ These disconnected objects follow the **Transfer Object** design pattern (sometimes referred to as **Value Objects**)
- ❖ Portlets manipulate the data in Transfer Objects, and then pass them back to the service layer
- ❖ The service layer then merges the Transfer Objects back into the hibernate session in order to persist them to the database



WWW.LIFERAY.COM

LIFERAY

IMPLEMENTING OUR BUSINESS LOGIC



- ❖ The generated service layer contains default methods for CRUD operations, but often times it is necessary to implement additional methods by hand.
- ❖ The **PublisherLocalServiceImpl.java** and **BookLocalServiceImpl.java** classes are *extension points* where you can add *custom methods*.
- ❖ Running the **build-service** task again will regenerate the **PublisherLocalService.java** and **BookLocalService.java** interfaces with the new method signatures.

WWW.LIFERAY.COM

LIFERAY

IMPLEMENTING THE PUBLISHER LOGIC

```
library-portlet
  docroot/WEB-INF/src
    com.liferay.training.library.model.impl
    com.liferay.training.library.service.base
      BookLocalServiceBaseImpl.java
      PublisherLocalServiceBaseImpl.java
    com.liferay.training.library.service.impl
      BookLocalServiceImpl.java
      PublisherLocalServiceImpl.java
    com.liferay.training.library.service.persistence
    com.sample.jsp.portlet
    META-INF
      service.properties
  docroot/WEB-INF/service
    com.liferay.training.library
    com.liferay.training.library.model
    com.liferay.training.library.service
      BookLocalService.java
      BookLocalServiceClp.java
      BookLocalServiceUtil.java
      ClpSerializer.java
      PublisherLocalService.java
      PublisherLocalServiceClp.java
      PublisherLocalServiceUtil.java
    com.liferay.training.library.service.persistence
```

- Type 'ctrl-shift-t' and open the `PublisherLocalServiceImpl.java` class.
- Notice that this class is currently empty, and that it extends the abstract class `PublisherLocalServiceBaseImpl.java`, which in turn implements the `PublisherLocalService.java` interface.
- We will add our business logic to the `PublisherLocalServiceImpl` class and then run service builder again to re-generate the interfaces.
- In our code, we will never call on the `PublisherLocalServiceImpl` class directly, but rather we will rely on the `PublisherLocalServiceUtil` object.

WWW.LIFERAY.COM



IMPLEMENTING THE PUBLISHER LOGIC

- Find the snippet called *o3-Publisher DAO*.
- Add the contents of this snippet into the `PublisherLocalServiceImpl.java` file.
- Add the following imports:
 - import java.util.List;
 - import com.liferay.counter.service.CounterLocalServiceUtil;
 - import com.liferay.portal.kernel.exception.SystemException;
 - import com.liferay.training.library.model.Publisher;

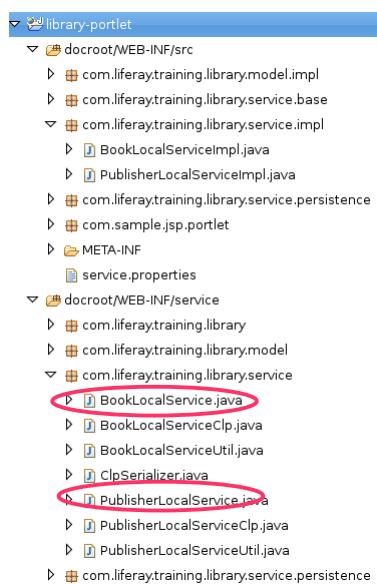
WWW.LIFERAY.COM



IMPLEMENTING THE BOOK LOGIC (I)

- We will add similar methods to the `BookLocalServiceImpl.java`.
- Find the exercise file called *04-Book DAO*.
- Add the contents of this snippet into the `BookLocalServiceImpl.java` file.
- Add the following imports:
 - `import java.util.List;`
 - `import com.liferay.counter.service.CounterLocalServiceUtil;`
 - `import com.liferay.portal.kernel.exception.SystemException;`
 - `import com.liferay.training.library.model.Book;`

EXAMINE THE SERVICE LAYER



- Open the `PublisherLocalService.java` and `BookLocalService.java` files
- Take note of the existing methods and their signatures included in these interfaces.
- Run the *build-service* task again
- Select the project root folder in the Eclipse package explorer and press the F5 key to refresh
- Notice that all of the methods we have implemented have been added to the interface: `addPublisher()`, `getPublishersByGroupId()`, `addBook()`, etc.

CHECKPOINT

- What happened? Our PublisherLocalService interface and BookLocalService interfaces have been regenerated based on the implementation of those interfaces.
- For developers who are used to programming to interfaces, this may seem a little backwards.
- Remember we are working with a code generator. Service Builder's job is to make things easier for us.
- We did initially define the interface in our service.xml. Service Builder generated the interfaces and its best guess at a default implementation.
- PublisherLocalServiceBaseImpl.java was that best guess.
- To add custom methods, you should not modify PublisherLocalServiceBaseImpl, because it's a generated class and your changes will be overwritten.
- Use PublisherLocalServiceImpl.java instead to add custom methods.

TROUBLESHOOTING

- If you get an error saying that a file can not be found in a temporary folder (e.g. service.properties), it is probably because the path of the temporary folder used by tomcat contains spaces or other non ascii characters. Please, make sure to define a temporary folder path containing only ascii characters by setting the property -Djava.io.tmpdir using one of these methods:
 - If you are launching tomcat from eclipse, add this parameter to the launch configuration of the server
 - If you are launching tomcat executing startup.bat, you need to add this parameter to setenv.bat
- For example:
-Djava.io.tmpdir=c:/temp

Notes:



LIBRARY APPLICATION: MVC IMPLEMENTATION

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated,
copied, sold, resold, or otherwise exploited for any commercial purpose
without express written consent of Liferay, Inc.

OBJECTIVES

- ❖ Housekeeping
 - Update WEB-INF/portlet.xml
 - Update WEB-INF/web.xml
- ❖ Understand the page flow and code organization for the MVC Portlet
- ❖ Implement the initial views for the portlet
- ❖ The snippets for the exercises in this presentation are in the category *Training-06-MVC*.

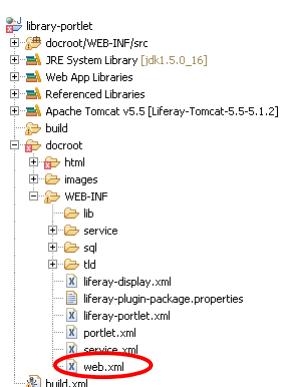
CHECKPOINT

- In the first part of this exercise, we used the Liferay IDE to create the Library project and to create the Publisher and Book portlets.
- The information we provided in the wizards was enough to create the initial Java classes, JSP pages, and configuration files.
- In the second part of this exercise, we used the Liferay IDE to define our service layer and then called the Liferay Service Builder to create the initial implementation.
- We then added our own business logic to the *LocalServiceImpl* classes that were generated and ran the Liferay Service Builder again to regenerate the interfaces.
- This service layer will be the Model layer in our MVC base architecture.
- Before we implement the initial View layer, we need to make a few updates to several of the configuration files.

PORTLET PREFERENCES

- As we learned earlier, the portlet specification defines portlet preferences as a way for portlets to store configuration information.
- These preferences can be defined in `WEB-INF/portlet.xml`.
- This step is optional, but it does allow you to specify default values.
- Portlet EDIT MODE is typically used to provide a way for users to configure portlet preferences.
- Add the contents of the snippet called *o1-Portlet Preferences* immediately following the `portlet-info` tag in the Publisher portlet.
- You can see we are defining a portlet preference for the number of rows per page and the phone number format.

WEB-INF/web.xml



- The **WEB-INF/web.xml** file is the standard Java web application **deployment descriptor** which is required by the **Java servlet specification**.
- This is where you configure the following types of settings:

```
<context-param />
<listener />
<servlet />
<servlet-mapping />
```
- Note that the elements listed above *must appear in the specified order*.
- When a portlet is deployed to the application server, Liferay Portal will add several entries to the deployed version of the web.xml file in order to ensure that the portlet will run properly.

WEB-INF/web.xml

- We will implement some of the logic in our JSPs using the JSTL tag library. We won't need this until the final part of the project when we implement Permission checking, but we'll configure WEB-INF/web.xml now.
- The Liferay IDE has already added one <taglib> element to the web.xml
- Add the contents of the snippet *o2-web.xml* just after the closing </taglib> element inside **WEB-INF/web.xml**.
- Now that we have enabled the JSTL tag libraries, we need to make the implementation of them available to our portlet.
- Since Liferay Portal uses many of the .jar libraries you are likely to use in your portlets, it is a best practice to use the same versions that Liferay uses.
- We have two ways of doing this:
 - Manually copy the jar files
 - Let the plugin automatically copy the jar files if they are already in the portal

ADDING JAR FILES: METHOD 1

- Method 1: Manually add the jar files
 - For JSTL, we have to place *jstl-api.jar* and *jstl-impl.jar* in */WEB-INF/lib* and *c.tld* in */WEB-INF/tld*
 - Copy *jstl-api.jar* & *jstl-impl.jar* from the *WEB-INF/lib* folder in your Liferay bundle to your project's *WEB-INF/lib* folder.
 - Copy *c.tld* from the *WEB-INF/tld* folder in your Liferay bundle to your project's *WEB-INF/tld* folder.
- Generally, this is not the “Liferay way” of doing things, but if you are using a non-Liferay-aware IDE, it is the only way to get it to recognize the .jars on the classpath.
- It additionally takes manual effort to keep the Liferay .jars and your project's .jars in sync when moving to future versions of Liferay.
- For this reason, we recommend using method 2 (next slide).

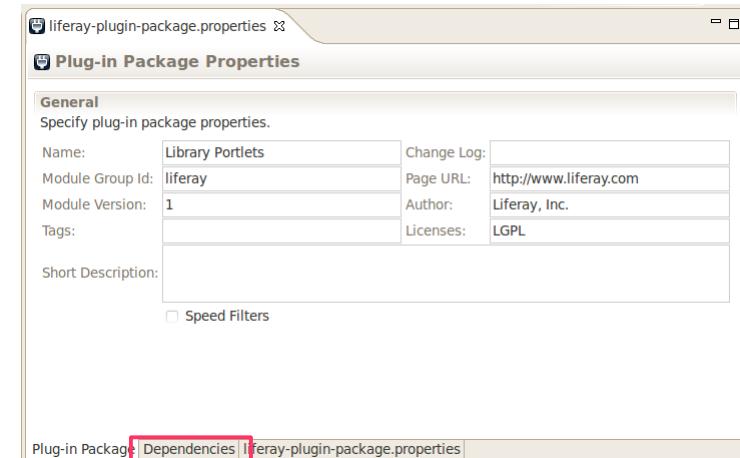
ADDING JAR FILES: METHOD 2

- Method 2: Let liferay add the jar files automatically (This is only possible if the portal is already using those jar files).
 - Portlet dependencies are declared in the *liferay-plugin-package.properties* file.
 - Opening the file in Liferay Developer Studio will give you a graphical interface to select the jar and tld files that are needed.
 - You can also manually add the following to the bottom of that file:

```
portal-dependency-jars=\n    jstl-api.jar,\n    jstl-impl.jar\n\nportal-dependency-tlds=c.tld
```

PORTLET DEPENDENCIES

- In this exercise, we'll be using the Liferay IDE to help manage our dependencies.
- Open the *docroot/WEB-INF/liferay-plugin-package.properties* file.
- Select the *Dependencies* tab from the bottom of the editor.

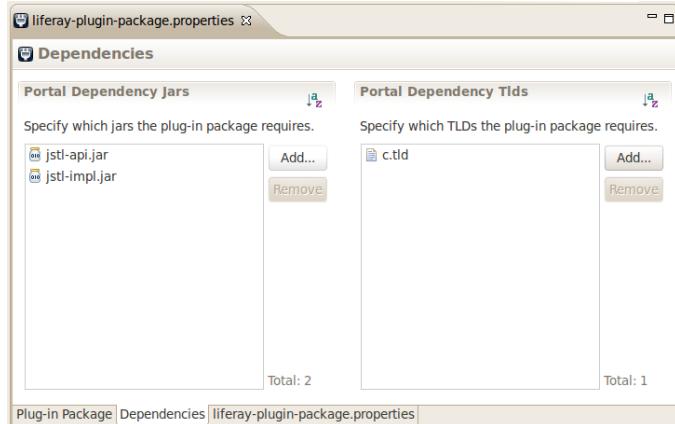


WWW.LIFERAY.COM

LIFERAY

PORTLET DEPENDENCIES

- Select the *Add...* button for the Portal Dependency Jars and select:
 - jstl-api.jar & jstl-impl.jar
- Select the *Add...* button for the Portal Dependency Tlds and select:
 - c.tld
- Save the file

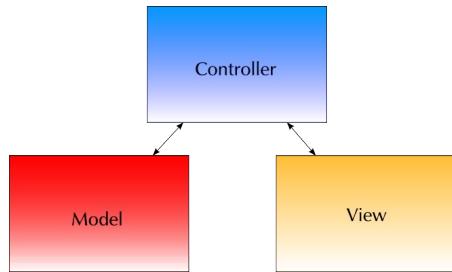


WWW.LIFERAY.COM

LIFERAY

MODEL-VIEW-CONTROLLER LAYERS

- Remember that in MVC, the Controller layer acts as a traffic director.
- It takes data from the View layer and passes it to the Model layer, where our business logic is.
- Once the Model layer has a result, the controller then determines based on that result what View should be displayed.
- It then sends the result to that view for display to the user.
- Our portlet classes will be our Controllers.



PORTLET CONTROLLER

- Because we chose to extend the Liferay MVCPortlet, the methods that correspond to the different portlet modes that can be rendered are hidden from us:
 - doView()
 - doEdit()
 - doHelp()
- If necessary, we can choose to implement them in our portlet classes, but for our purposes we can rely on the super class implementation of each method.
- Following the pattern established in the Liferay MVCPortlet framework, we will use a render parameter to decide which view the user wants to see
- The MVCPortlet—or rather the LiferayPortlet, which MVCPortlet extends—will use this render parameter to dispatch the user to the correct JSP.

VIEW MODE

- Our portlets' View Modes have two views:
 - The default view, which shows the list of Publishers or Books:

The screenshot shows a Liferay portlet titled "Liferay Publisher". It has a header bar with buttons for "Add Publisher", "Actions", and other portlet controls. Below the header is a table with four columns: "Name", "Email Address", "Web Site", and "Phone Number". A single row is displayed: "Liferay Press", "lp@liferay.com", "http://www.liferay.com", and "123-456-7890". At the bottom of the table, it says "Showing 1 result."

- The form, where users will add and edit Publishers or Books:

The screenshot shows the same "Liferay Publisher" portlet, but in Edit Mode. The title bar now says "Publisher Details". The form fields are identical to the View Mode table: Name (Liferay Press), Email Address (lp@liferay.com), Web Site (http://www.liferay.com), and Phone Number (123-456-7890). At the bottom are "Submit" and "Cancel" buttons.

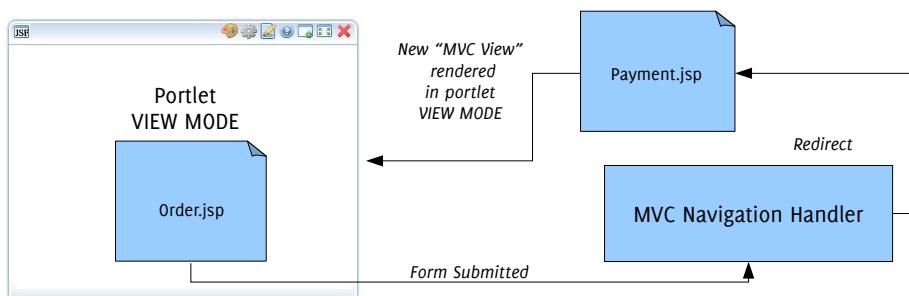
WWW.LIFERAY.COM

LIFERAY

SIDE ISSUE:

PORTLET VIEW MODE VS. MVC VIEW

- Because portlets have a VIEW MODE and our Publisher examples implement the Model / VIEW / Controller design pattern, it is easy to get the two *view* terms confused.
- It is important to understand that portlet VIEW MODE does not necessarily have a 1:1 relationship with an MVC view; rather it could be a 1:M relationship.
 - For example: MVC applications that follow a conversation (like booking a hotel) may have several MVC views rendered in portlet VIEW MODE

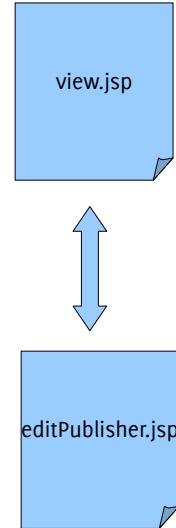


WWW.LIFERAY.COM

LIFERAY

PAGE FLOW

- The only Portlet Mode that will flow from one page to another in our portlets is *view mode*.
- Users will be able to click an *Add Publisher* or an *Edit Publisher* button and be brought to a form which allows them to do those things.
- When that form is submitted, users will be brought back to the original view.



PAGE FLOW IN THE CONTROLLER

- In MVC, the Controller layer is the traffic director, defining the page flow of the application.
- As we've seen, the *MVCPortlet* makes it very easy to control the page flow of your portlet without writing a single line of code.
- When invoking a Render URL, we simply have to pass a parameter called *jspPage* with the value set to the path of the page we want to invoke.

IMPLEMENTING THE CONTROLLER

- Our publisher portlet has two possible pages in view mode that may be presented.
- Initially, the first view will only contain an *Add Publisher* button. Under this button will be a list of items that have been added.
- The second view presents a form to end users so they can add a new publisher or edit an existing one.
- We switch between the two views by manipulating a the `jspPage` render parameter.
- This keeps our controller cleaner and easier to read and maintain.

IMPLEMENTING THE VIEW LAYER

- We will be implementing our view layer using JSP.
- The first view we will implement will be the default view for the portlet. It will contain a table with the Publishers and an *Add Publisher* button.
- The Liferay IDE already created a folder under docroot called `html`.
- Create a new file in the `docroot/html/` directory called `init.jsp`.
(This file will be used for the rest of the jsp files in order not to repeat the same includes or values)
- Add into the `init.jsp` file the contents of the snippet `o3-init.jsp`.

IMPLEMENTING THE DEFAULT VIEW

- The Liferay IDE also created a folder under html called *publisher* and a file in that folder named *view.jsp*
- Add the code from *04-Publisher view.jsp* into the *view.jsp* file.
- Notice that this file includes the *init.jsp* file we created in the last step, so it inherits all of the imports and tag library initialization.
- We then define a variable called *redirect*. This value will be added as a render parameter and will be read by the MVCPortlet. The redirect drops all the request parameters, and the URL will no longer tell the portal to perform the same action even if the user clicks refresh.

IMPLEMENTING THE DEFAULT VIEW

- Next, we create the Render URL. Note the two parameters that are added to the Render URL:

```
<portlet:param name="jspPage" value="/html/publisher/edit_publisher.jsp" />
<portlet:param name="redirect" value="<%=\ redirect %>" />
```
- The **jspPage** parameter is used by the MVC Portlet to define the jsp that should be rendered.
- Lastly, we use that URL in a button.

THE EDIT PUBLISHER VIEW

- Because we want to share the one view for both Editing and Adding, the Edit Publisher view is a little bit more complicated.
- Create a new file in the `docroot/html/publisher` directory called `edit_publisher.jsp`.
- Add the code from the file `05-Publisher edit_publisher.jsp` into the `edit_publisher.jsp` file.
- We will again include the common `init.jsp`
- Next, we check the `request` object to determine if a `publisherId` value was passed in.
- If it was, then this form will be used to edit an existing publisher. If it wasn't then the form will be used to create a new publisher.
- We again define the redirect, but this time we use the `<liferay-ui:header>` tag

THE EDIT PUBLISHER VIEW

- We then lay out our form using the Alloy UI form elements.
- The `<aui:model-context>` tag is used to indicate what type of object we're working with. This allows the `<aui:input>` tags to automatically read the corresponding attributes from the object identified and determine the correct inputs (e.g. which can be a text, localized text, calendar.... etc).
- Next, we create an action URL and use some logic to determine the value of the `name` attribute. The `name` attribute determines which `action` method will be called in our portlet class.
- Finally we create an Alloy UI based form to collect the user input.

EDIT PREFERENCES VIEW

- The Liferay IDE created our *edit.jsp* file in the *docroot/html/publisher/* directory.
 - Replace the contents of the *edit.jsp* file with the contents of *o6-Publisher edit.jsp*.
 - You will see that this is just a Alloy UI form, using a table to lay out the fields.
-
- Bonus: Create preferences for the book portlet too.

WWW.LIFERAY.COM



HELP VIEW

- The Liferay IDE created our *help.jsp* file in the *docroot/html/publisher/* directory.
 - Write any help message you want to display in this file.
-
- Bonus: Create a help page for the book portlet too.

WWW.LIFERAY.COM



VIEW FOR THE BOOK PORTLET

- ❖ Now it is time to implement the view mode for the Book portlet. It is virtually identical to the Publisher view.
- ❖ Add the code from *07-Book view.jsp* into the *docroot/html/book/view.jsp*.
- ❖ Create a new file in the *docroot/html/book* directory named *edit_book.jsp*
- ❖ Add the code from *08-Book edit_book.jsp* into the *docroot/html/book/edit_book.jsp*.

EDIT BOOK

- ❖ There are two differences in the *edit_book.jsp* that we should point out.
- ❖ The first difference is that we are dealing with a date value.
- ❖ In order to avoid confusion over a date entered in the incorrect format, it would be ideal to provide the user with some sort of a date picker widget.
- ❖ In fact, by using the following two lines of code, we've already done just that!

```
<aui:model-context bean="<% book %>" model="<% Book.class %>" />
```

And ...

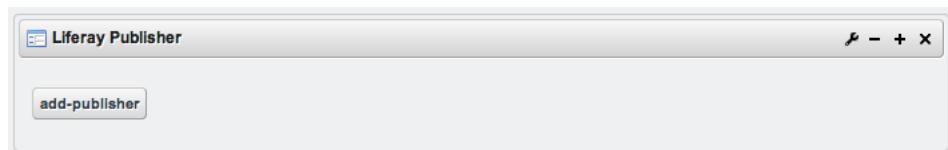
```
<aui:input name="publicationDate" />
```
- ❖ Behold the power of Alloy UI!

EDIT BOOK

- The second difference is that when creating or editing a book, the user is presented with a list of Publishers in a drop down list.
- First, we use the *PublisherLocalServiceUtil* to retrieve a list of publishers and store them as a *List*.
- Next we use the Alloy UI select tag and iterate through the *List*.

CHECKPOINT!

- You are now ready to deploy your portlet!
- Drag the project title to the Liferay server in the Server Tab. Watch the console and wait for the deployment to finish.
- Add the Publisher portlet to a page, and you should see something that looks like the portlet below:



Notes:



LIBRARY PORTLET: PORTLET ACTIONS

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

OBJECTIVES

- Understand how Action URLs are used to wire the View actions with Controller methods
- Modify Controller to process actions correctly
- The snippets for the exercises in this presentation are in the category *Training-07-MVC Actions*.

PORTLET ACTIONS

- In the previous exercise, we created our view layer and deployed the portlets to the portal.
- The portlets look fine, but as you have probably noticed, the Save buttons in our portlets don't work.
- That is because we haven't finished wiring the view layer to the controller layer.
- In this exercise, we will complete the wiring and implement the actions behind the buttons.

PORTLET ACTIONS

- There are three different actions that a user could trigger in our Publisher Portlet.
- The first is the *Add Publisher* action and is triggered by submitting the add/edit publisher form.
- The second is the *Update Publisher* action and is also triggered by submitting the add/edit publisher form.
- The third is the *Delete Publisher* action and is triggered by clicking the *Actions → Delete* button.
- We'll start first by implementing the *Add Publisher* action.
- **Insert** the contents of the snippet called *o1-Publisher Actions* into the PublisherPortlet.java file.
- Import the following classes:
`com.liferay.portal.kernel.log.Log`

HOW DOES addPublisher WORK?

- The first method that we added to the PublisherPortlet was the `addPublisher()` method which adds a new publisher to the database.
- This method makes use of a convenience method we'll discuss shortly, to retrieve the submitted publisher details from the ActionRequest.
- The submitted Publisher is then passed to the Model layer of the application.
- The Model layer adds the Publisher to the database using the `addPublisher` method provided by the `PublisherLocalServiceUtil` which you used Service Builder to create.
- The `sendRedirect` method is called to ensure that refreshing the page does not result in a duplicate record being inserted into the database.

HOW DOES publisherFromRequest WORK?

- The second method that we added to the PublisherPortlet was a convenience method called `publisherFromRequest()`.
- This method takes an ActionRequest as an argument and will return a Publisher object.
- From the ActionRequest, the ThemeDisplay can be obtained, which will be used to retrieve the Company ID and Group ID.
- The ActionRequest also provides access to the values the user submitted on the Add Publisher form.
- These values are added to a new PublisherImpl object.

THE BOOK PORTLET

- The Book portlet works in a similar way.
- **Add** the Action methods from the snippet *02-Book Actions* into the Book portlet.
- Import the following classes:

```
java.util.Date  
com.liferay.training.library.model.Book  
com.liferay.portal.kernel.log.Log  
com.liferay.portal.kernel.util.LocalizationUtil
```
- Again, there are couple of slight differences with the book portlet.
- First, because we're dealing with a date value, we retrieve the date components and use a Liferay convenience method to assemble the date value.
- Second, because we're working with a localized value for the book title, we need to retrieve a *Map* of localized values.

WWW.LIFERAY.COM



CHECKPOINT

- The Liferay IDE will automatically re-deploy your portlets. Wait for the deploy to finish and then click the *Add* buttons.
- Access the form and submit a new Publisher and Book.
- After submitting a new Publisher or Book, you will be redirected to the default view, which is still only showing the *Add* button.
- In the next part of this exercise, we'll implement the Liferay Search Container to display Publishers and Books already added to the database.

WWW.LIFERAY.COM



LIFERAY SEARCH CONTAINER

- A common requirement for data-driven Portlet applications is to work with a list of objects.
- In this example, we're working with lists of Publishers and Books, but we could just as easily be working with Users or Communities.
- We could create our own mechanism to display our list of objects in the view layer of our application, but Liferay has already solved this problem for us.
- The Liferay *SearchContainer* is a class that works in conjunction with Liferay's UI tag libraries to provide a user interface wrapper around lists of objects.
- Because SearchContainer is a standard Liferay UI component, using it makes our application look just like one of the core Portal applications.

WWW.LIFERAY.COM



LIFERAY SEARCH CONTAINER

- The Liferay tag `<liferay-ui:search-container>` encapsulates all the code.
- SearchContainer will handle all formatting and pagination for us automatically.
- We will provide details such as column header names, number of rows, etc. and SearchContainer will do the rest.

Name	Email Address	Phone Number	Website	
Julio Camarero	jcamarero@liferay.com	1111111111	www.liferay.com	Actions
Rich Sezov	rsezov@liferay.com	1111111111	www.liferay.com	Actions

Showing 2 results.

WWW.LIFERAY.COM



IMPLEMENTING THE SEARCH CONTAINER

- We'll start first by implementing the *SearchContainer* in the Publisher View.
- Add the contents of the snippet called *o3-Publisher Search Container* to the end of the *html/publisher/view.jsp* file.
- We will also need to add a new jsp to display the menu of actions that can be performed on each record in the SearchContainer.
- Create a new file in docroot/html/publisher called *publisher_actions.jsp* and add the contents of *o4-Publisher Actions* into the file.

THE SEARCH CONTAINER TAGLIB

```
<liferay-ui:search-container-row  
    className="com.liferay.training.library.model.Publisher"  
    keyProperty="publisherId"  
    modelVar="publisher"  
>  
    <!--  
        className - The type of Object in your List. In this case, we specify the class  
        name for Publisher.  
        keyProperty - The object field that acts as the Primary Key  
        modelVar - The object of the current iteration will be made available as a  
        variable with this name.  
    -->  
    <liferay-ui:search-container-column-text  
        name="name"  
        value="<%-- publisher.getName() %>-->" />  
    <liferay-ui:search-container-column-text  
        name="email-address"  
        property="emailAddress" />  
    <!--  
        Inserts a text column  
        -->  
        <!--  
            name - Name of the column  
            value - Value of the column  
            property - This will automatically look in the Publisher object for the  
            "emailAddress" property. It's the same thing as publisher.getEmailAddress()  
        -->
```

THE SEARCH CONTAINER TAGLIB

```
<liferay-ui:search-container-column-jsp  
    align="right"  
    path="/html/publisher/publisher_actions.jsp"  
/>
```

- Inserts a column with the content of a JSP file
 - align – alignment of the result
 - path – location of the JSP file

THE ACTIONS BUTTON

- The Actions button is created by the *publisher_actions.jsp* that is included on every row of the Search Container.
- The two buttons that are implemented need the primary key of the database record in order to perform their operations.
- As the Search Container is being created, it loops through the collection of objects (in our case, Publishers or Books) that were retrieved from the database.
- As it goes through, the primary keys of the objects can be pulled for each row and embedded in the Action button URLs.
- The Action button itself is generated through a `<liferay-ui:icon-menu>` tag.

THE BOOK PORTLET

- The SearchContainer for the Book portlet works exactly the same way.
- [Add](#) the contents of the snippet called *05-Book Search Container* to the end of the *html/book/view.jsp* file.
- [Create](#) a new file in docroot/html/book called *book_actions.jsp* and add the contents of *06-Book actions.jsp* into the file.

CHECKPOINT

- Once the Liferay IDE has finished deploying the portlets, refresh the page, and you should see any Publishers or Books added to the portlet displayed in the *SearchContainer*.
- If you click on the *Actions* button in the search container, you'll see that you have the option to edit that item or delete that item.
- Clicking on either of these options will result in an error message. This is because we have not wired these actions to methods in the controller yet.

PORLET ACTIONS (II)

- As mentioned on a previous slide, there were three different actions that a user could trigger.
- We've already addressed the *Add* action, so we will now turn our attention to the *Edit* and *Delete* actions.
- **Insert** the contents of the snippet called *07-Publisher Actions* into the PublisherPortlet.java file, below the publisherFromRequest() method, but before the *_log* variable declaration.
- Import the following classes:
`javax.portlet.PortletPreferences`

HOW DOES IT WORK

- The first method that we added to the PublisherPortlet was the *updatePublisher()* method which will update an existing publisher in the database.
- This method makes use of the same convenience method we used for adding new publishers.
- The *sendRedirect* method is then called to ensure that clicking refresh does not repeat the completed action.

HOW DOES IT WORK?

- ❖ The second method that we added to the PublisherPortlet was the *deletePublisher* method.
- ❖ This method retrieves the *publisherId* from the ActionRequest and then uses the *deletePublisher()* method provided by the PublisherLocalServiceUtil.
- ❖ This method is invoked with an Action URL that was also created in the *publisher_actions.jsp*.

HOW DOES IT WORK?

- ❖ The last method that we added to the PublisherPortlet was the *setPublisherPref* method.
- ❖ This method retrieves the values submitted from the *edit.jsp* form to set the preferences for the portlet.
- ❖ This method is invoked with an Action URL that was created in the *edit.jsp*.

THE BOOK PORTLET

- ❖ The update, delete, and portlet preference actions for the Book portlet work exactly the same way.
- ❖ Insert the contents of the snippet called *o8-Book Actions* into the BookPortlet.java file, below the bookFromRequest() method, but before the *_log* variable declaration.
- ❖ Import the following classes:

```
javax.portlet.PortletPreferences
```

CHECKPOINT

- ❖ After the Liferay IDE finished deploying the portlets, refresh the page, and you should see any Publishers or Books added to the portlet displayed in the *SearchContainer*.
- ❖ If you click on the *Actions* button in the search container, you'll see that you have the option to edit that item or delete that item.
- ❖ Clicking on either of these options should now work correctly.

Notes:



LIBRARY APPLICATION: USER FEEDBACK, VALIDATION, AND LOCALIZATION

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.



OBJECTIVE

- This exercise walks you through the steps necessary to refine our View and Controller Layers of our application
- We will be implementing the following features:
 - User Feedback
 - Localization of Messages
 - Validation

WHAT'S ALREADY BEEN DONE

- The Library Application was created using the Plugins SDK.
- The service layer was created by Liferay Service Builder.
- Custom business logic was implemented in the xxxLocalServiceImpl.java files.
- The service layer, plus the custom business logic constitutes the model layer of our MVC application.
- The PublisherPortlet has been updated to act as the controller layer of our MVC application.
- A basic view layer has been created using JSP.

WWW.LIFERAY.COM



WHAT STILL NEEDS TO BE DONE

- The portlet needs to provide user feedback after a successful save or an error has occurred.
- Feedback messages need to be localized.
- User input needs to be validated before writing to the database.

WWW.LIFERAY.COM



USER FEEDBACK AND VALIDATION

- When it comes to things like displaying user feedback or providing validation, developers tend to turn to frameworks, rather than write their own.
- Some developers choose to create their own utilities to provide these features that they can then reuse on subsequent projects.
- This code can be mundane and unfulfilling to write.
- There is nothing wrong with the existing frameworks, and if you are already familiar with them, you can use many of the leading frameworks in Liferay.
- The Liferay platform, however, also provides these features for developers to use.
- Portlets that utilize these features will look and act more like native Liferay portlets.

MESSAGES AND USER FEEDBACK

- In our existing view layer, you'll notice that by leveraging Alloy UI form tags, we have been able to minimize the number of hard-coded labels we needed to add use.
- This may be acceptable for applications whose audience is limited to one locality or language, but our requirement is to create an application that supports multiple languages through localization.
- Additionally, because Alloy UI is relying on how the model layer was created, we would have difficulty if we ever needed to override the current labels.
- Liferay provides some built-in tools that allow us to address both of these issues.
- To use these tools, we will need to add language keys in a separate file.
- This file is referred to as a resource bundle and is called *Language.properties*.

RESOURCE BUNDLES

- A resource bundle can provide language-specific versions of text-based elements of the portlet such as the category or user messages and feedback.
- The fully qualified name of the resource bundle should be set in the *portlet.xml* using the `<resource-bundle>` tag.
- Based on our input, Liferay Developer Studio has already created the *Language.properties* file and configured the *portlet.xml* file to be aware of this file.

RESOURCE BUNDLES

- **Add** the contents of the snippet *01-Language Properties* into the `Language.properties` file in the `docroot/WEB-INF/src/content` directory.

LOCALIZE EDIT PUBLISHER VIEW

- Now we will revisit our view layer and modify it to make use of the message in our resource bundle.
- Because we will be making use of some of Liferay's custom tags, we need to have a tag library declaration for them. Open *init.jsp* and [view](#) the following declaration below the existing Portlet API declaration that has been added for you:

```
<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui" %>
```
- **Add** the following just below the *<include>* tag in the *view.jsp* (*o2-Success Messages*):

```
<liferay-ui:success key="publisher-added" message="publisher-added-successfully" />
<liferay-ui:success key="publisher-updated" message="publisher-updated-successfully" />
<liferay-ui:success key="publisher-deleted" message="publisher-deleted-successfully" />
```
- These tags will be replaced with the messages we defined in our *Language.properties* file at runtime.

LOCALIZE EDIT PUBLISHER VIEW

- The messages will only be displayed if the corresponding key is found in the *SessionMessages* object.
- Add the following line into the placeholder in the *addPublisher* method in the *PublisherPortlet.java* file.

```
SessionMessages.add(request, "publisher-added");
```
- Add the following line into the placeholder in the *updatePublisher* method in the *PublisherPortlet.java* file.

```
SessionMessages.add(request, "publisher-updated");
```
- Add the following line into the placeholder in the *deletePublisher* method in the *PublisherPortlet.java* file.

```
SessionMessages.add(request, "publisher-deleted");
```
- Hit Ctrl-Shift-O to add the import statement.

CHECKPOINT

- After saving the file, watch the console to make sure the Liferay IDE hot-deploys the portlet properly.
- Click the *Add Publisher* button in the Publisher portlet.
- Complete the form and click the *Submit* button.
- You should see the following message displayed.



SUPPORTING MULTIPLE LANGUAGES

- We have replaced the hard coded labels in our form with messages stored in a *Language.properties* file, but we haven't implemented the mechanism for supporting multiple languages.
- We will do this by providing companion files to our original *Language.properties* file.
- These companion files will have a two-letter language code appended to them.
- For example, *Language_es.properties* or *Language_fr.properties* for Spanish or French respectively.
- Our application will use the values in these files if the user's locale is set to *es* or *fr*.
- Typically, the translation process is a lot of work, requiring someone who is knowledgeable in the language to translate the file for each language you want to support.
- Liferay provides a tool to help with this process by generating a translation automatically.

SUPPORTING MULTIPLE LANGUAGES

- The Plugins SDK contains Ant targets that use the Babelfish service (<http://babelfish.yahoo.com>) to translate all of the keys in our *Language.properties* file to multiple languages.
- Babelfish is an automated translation service and doesn't provide the same level of translation that a human translator would provide. It is, however, a starting point.
- If you are targeting a particular audience, you would want to have someone review the generated translation before moving into a Production environment.
- Another feature that Liferay provides portlet developers is a common language file that is inherited from the portal.
- This file is located in the source code at *portal-impl/src/content*.
- This file contains many common messages, such as *Save* or *Cancel*, that can be used in your portlet as-is.

SUPPORTING MULTIPLE LANGUAGES

- Right-click on *Language.properties* → Liferay → Build Languages
- Be sure you are connected to the Internet when you execute this.
- After the build is complete, look at the *Content* folder to see the generated translations.

VALIDATION

- Our current iteration of the portlet will accept any user input, or even no input at all. This is not desirable, so we will now implement validation to ensure that users are providing valid data.
- Rather than writing our own validation code or making use of one of the frameworks, we will make use of Liferay's validation utility.
- The Liferay validator utility is implemented in `com.liferay.kernel.util.Validator`, which will need to be imported into our controller class.
- For our project, we will implement a simple example of validation. We will be checking to ensure that users have provided some response. As long as there is a response, we will consider it valid.

VALIDATION

- [Create](#) a new Class called PublisherValidator.java in the src folder inside the package `com.liferay.training.library.portlet`.
- [Replace](#) the contents of the newly created PublisherValidator class with the snippet *o3-Publisher Validator*.

VALIDATION

- [Replace](#) the method `addPublisher` in `PublisherPortlet.java` so that it validates the introduced data, using the contents of the snippet `o4-addPublisher()`.

VALIDATION

- [Replace](#) the method `updatePublisher` in `PublisherPortlet.java` so that it validates the introduced data, using the contents of the snippet `o5-updatePublisher()`.

VALIDATION

- ❖ [Replace](#) the method `deletePublisher` in `PublisherPortlet.java` so that it validates the introduced data, using the contents of the snippet `o6-deletePublisher()`.

VALIDATION

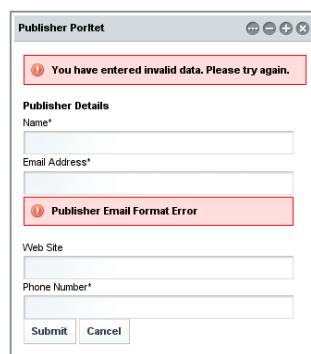
- ❖ Our validation uses the `isNull` method to ensure that none of the form values are set to null.
- ❖ The `isEmailAddress` method is used to ensure the submitted email address is in the proper format.
- ❖ The `isPhoneNumber` method is used to ensure the submitted phone number is in a valid format.
- ❖ If submitted values are valid, the data is written to the database.
- ❖ If there are any errors, we add a relevant message to the `SessionErrors` object which will be passed back to the view layer.
- ❖ These error messages also need to be added to the `Language.properties`

VALIDATION

- **Add** the contents of *07-More Properties* into the *content/Language.properties* file.
 - Now we will need to update the *edit_publisher.jsp* so that any errors passed back from the controller will be displayed in the proper place.
 - **Replace** the contents of the *<aui:fieldset>* tag in the *edit_publisher.jsp* with the contents of the *08-edit_publisher.jsp* snippet.
-
- Bonus: Improve the validation in order to avoid deleting publisher if there are books published by those publishers.
 - Bonus 2: Remove the option to delete a publisher from the UI when there are books published by this publisher.

CHECKPOINT

- Build your Languages again and wait until the server shows your portlet has been deployed.
- Click the *Add Publisher* button in the Publisher portlet.
- Leave empty any obligatory field in the form and click the *Submit* button.
- You should see at least one error message displayed.



A screenshot of a web application window titled "Publisher Portlet". The window contains a form for "Publisher Details" with fields for Name*, Email Address*, Web Site, and Phone Number*. There are two red validation messages: "You have entered invalid data. Please try again." above the Name field, and "Publisher Email Format Error" above the Email Address field. At the bottom are "Submit" and "Cancel" buttons.

MAKE THE CHANGES TO THE BOOK PORTLET

- **Add** to Language.properties the contents of 09-Even More Properties.
- **Replace** contents of the fieldset in edit_book.jsp with 10-edit_book.jsp.
- **Create** BookValidator.java in the com.liferay.training.library.portlet package and add the methods in 11-Book Validator to the file.
- **Replace** the addBook, updateBook, deleteBook, and setBookPref methods in BookPortlet with the versions in 12-Book Methods.
- **Deploy** the portlets again. Now the BookPortlet is localized and validates its fields.

Notes:

 LIFERAY.
**LIBRARY APPLICATION:
PERMISSIONS**

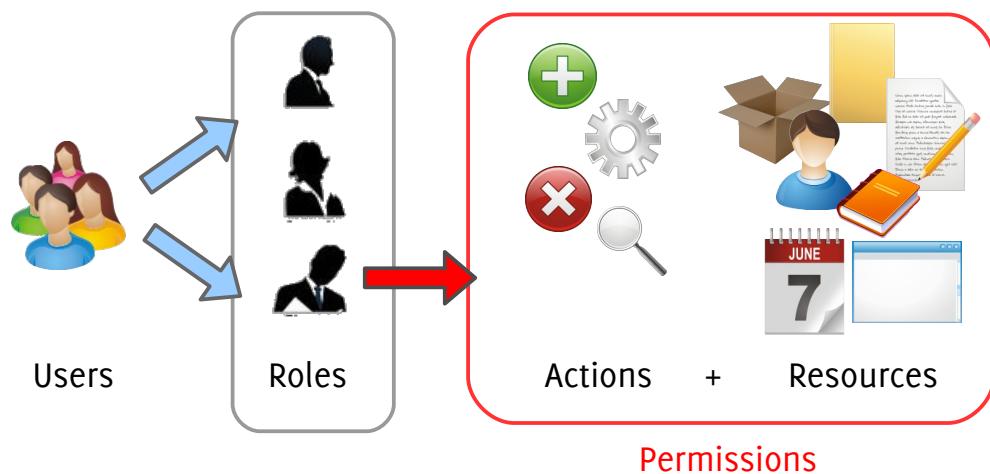
Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

OBJECTIVE

- >This exercise walks you through the steps necessary to connect your portlet to Liferay's permission system. Snippets are in 09-Permissions.



PORTLET RESOURCES AND MODEL RESOURCES

- Permissions are granted or denied by Liferay Portal based on a resource+action pair.
 - Resources can either be <portlet-resource> or <model-resource>.
 - The <portlet-resource> type is used to define actions that the end-user can perform with respect to each portlet window.
-
- The <model-resource> type is used to define actions that the end-user can perform with respect to the service/persistence layer. There are usually two types of actions:
 - Top level actions: are not applied to a specific resource. For example, to create a new publisher.
 - Resource actions: are applied to a specific resource. For example, editing or deleting a specific publisher.

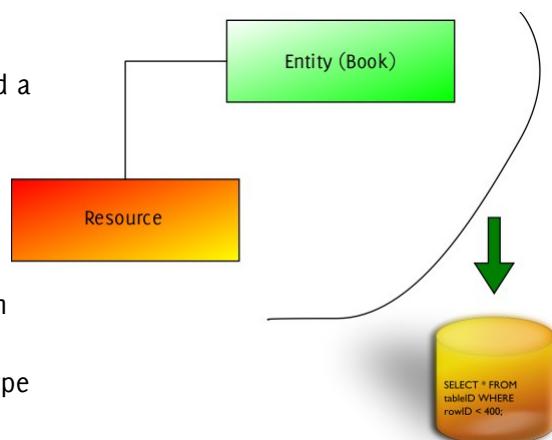


WWW.LIFERAY.COM

LIFERAY

CHANGES TO SERVICE LAYER

- In order to do permissions on a model resource, you need to add a *resource* to the database at the same time you add your entity.
- Liferay uses resources for permission checking.
- Resources are linked in the database to the entities to which they belong.
- Liferay adds an entry for each type of permission available on the resource.



WWW.LIFERAY.COM

LIFERAY

REPLACING ADD AND DELETE METHODS

- We need to replace the add and delete methods in both of our entities so that they add and delete resources at the same time.
- 1. [Open](#) `PublisherLocalServiceImpl.java`. Remove the old implementation of `addPublisher()`.
- 2. [Add](#) the contents of `o1-Publisher Service` to the file. Hit Ctrl-Shift-O to fix imports.
- 3. [Open](#) `BookLocalServiceImpl.java`. Remove the old implementation of `addBook()`.
- 4. [Add](#) the contents of `o2-Book Service` to the file.
- 5. Run *Build Services* again.
- Notice that we now make a call to `resourceLocalService` when we add our entity to the database.
- **Bonus:** How did an instance of `resourceLocalService` become available in our `-Impl` class?

MODIFY THE PORTLET CLASS

- Since we have modified the call to add entities so that it now requires a user ID, we need to refactor every call we've made to this method.
- Open your `BookPortlet.java` and `PublisherPortlet.java` classes, do a Find for the locations of calls to `addBook()` and `addPublisher()` and modify them so that they include the user ID.
- The solution to this is on the next slide.

SERVICE CALL SOLUTION

Add before call to addBook / addPublisher:

```
ThemeDisplay themeDisplay =
    (ThemeDisplay)request.getAttribute(WebKeys.THEME_DISPLAY);
long userId = themeDisplay.getUserId();
```

Publisher call:

```
PublisherLocalServiceUtil.addPublisher(publisher, userId);
```

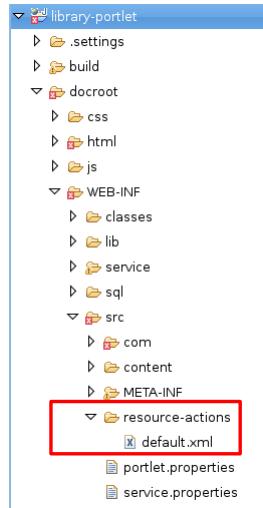
Book call:

```
BookLocalServiceUtil.addBook(book, userId);
```

CONFIGURING PERMISSIONS

- Permissions in Liferay are defined declaratively in an XML file.
- Once defined, no further configuration is necessary.
- All a developer needs to do after this is to wrap elements in permission checks.
- First, we'll define the permissions, and after this, we'll add permission checks to our code.

default.xml



- [Create](#) a new folder in src called resource-actions
- [Create](#) a new file in that folder called default.xml:
- [Enter](#) the following into the file (*o3-Publisher Template*):

```
<?xml version="1.0" encoding="UTF-8"?>
<resource-action-mapping>
```
- In the next slide, we will define a [`<portlet-resource>`](#) that contains the following portlet window actions:

```
ADD_TO_PAGE
VIEW
CONFIGURE
```

default.xml

- [Add](#) the contents of *o4-Publisher Portlet Permissions* in between the `<resource-action-mapping>...</resource-action-mapping>` tags.

```
<portlet-resource>
  <portlet-name>liferay-publisher</portlet-name>
  <supports>
    <action-key>ADD_TO_PAGE</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 />
</portlet-resource>
```



- These permissions are associated with the portlet-resource `liferay-publisher` (each portlet window of this type).

default.xml

- Add the contents of *05-Book Portlet Permissions* after the contents you previously pasted.

```
<portlet-resource>
    <portlet-name>liferay-book</portlet-name>

    <supports>
        <action-key>ADD_TO_PAGE</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 />

```

```
</portlet-resource>
```

- These permissions are associated with the portlet-resource liferay-book (each portlet window of this type).



default.xml

- Add underneath your previous paste the contents of *06-Model Permissions*. These permissions are associated with creating the content:

```
<model-resource>
    <model-name>com.liferay.training.library.model</model-name>
    <portlet-ref>
        <portlet-name>liferay-book</portlet-name>
    </portlet-ref>

    <supports>
        <action-key>ADD_PUBLISHER</action-key>
        <action-key>ADD_BOOK</action-key>
    </supports>

    <community-defaults/>
    <guest-defaults/>
    <guest-unsupported>
        <action-key>ADD_PUBLISHER</action-key>
        <action-key>ADD_BOOK</action-key>
    </guest-unsupported>

```

```
</model-resource>
```

- These permissions are associated with the models, specifically its top level actions.



default.xml

- Add below the previous snippet the contents of *07-Publisher Permissions*. These permissions are associated with the publishers:

```
<model-resource>
    <model-name>com.liferay.training.library.model.Publisher</model-name>
    <portlet-ref>
        <portlet-name>liferay-publisher</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>
```

- These permissions are associated with the models. In particular, they are actions for publisher resources.



default.xml

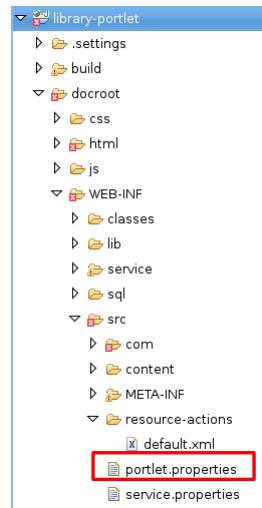
- Add below the previous snippet the contents of *08-Book Permissions*. These permissions are associated with the books:

```
<model-resource>
    <model-name>com.liferay.training.library.model.Book</model-name>
    <portlet-ref>
        <portlet-name>liferay-book</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>
```



- These permissions are associated with the models. In particular, they are actions for specific book resources.

portlet.properties



- Liferay Portal's hot-deploy process looks in the *portlet.properties* file in order to determine where to look for portlet permissions.
- This file must be at the root of the classpath
- **Create** this file in:
WEB-INF/src/
- And **enter** this content:
resource.actions.configs=resource-actions/default.xml
- This is a pointer to the *default.xml* file.

PERMISSION CHECKING

- **Insert** the contents of *09-Publisher Permission Check* right after the redirect variable is declared in *publisher/view.jsp*, inside the script tags.

```
<%  
boolean hasAddPermission = permissionChecker.hasPermission(  
    scopeGroupId, "com.liferay.training.library.model",  
    scopeGroupId, "ADD_PUBLISHER");  
boolean hasConfigurePermission = permissionChecker.hasPermission(  
    scopeGroupId, Group.class.getName(), scopeGroupId,  
    ActionKeys.PERMISSIONS);  
String redirect = PortalUtil.getCurrentURL(renderRequest);  
%>
```

PERMISSION CHECKING

- ❖ Replace the contents of the <aui:button-row> tag for the *Add Publisher* button with the following in the Publisher portlet's view.jsp (*10-Add Publisher Button*):

```
<c:if test='<%= hasAddPermission %>'>
    <portlet:renderURL var="addPublisherURL">
        <portlet:param name="jspPage" value="/html/publisher/edit_publisher.jsp" />
        <portlet:param name="redirect" value="<%= redirect %>" />
    </portlet:renderURL>

    <aui:button value="add-publisher" onClick="<%= addPublisherURL.toString() %>"/>
</c:if>
```

PERMISSION CHECKING

- ❖ The `permissionChecker` object was made available to us by the `<liferay-theme:defineObjects />` declaration in the *init.jsp*.
- ❖ The method `hasPermission`:

```
public boolean hasPermission(long groupId, String name, long primKey, String actionId)
```

 - `groupId`: the group to which the resource belongs
 - `name`: the model name of the resource (usually the class name)
 - `primKey`: the primary key of the resource. (For top level actions the primary key will be the `groupId`)
 - `actionId`: the id of the action from default.xml

WRAP THE THREE ACTIONS IN PERMISSIONS

- ❖ In *publisher/publisher_actions.jsp*, each of the three actions should be checked for permissions before being displayed.
- ❖ Replace the current Edit Action and Delete Actions with the contents of the snippet *11-Edit and Delete Buttons*.
- ❖ Edit Action:

```
<c:if test="<% permissionChecker.hasPermission(groupId, name, publisherId, ActionKeys.UPDATE) %>">
<portlet:renderURL var="editURL">
<portlet:param name="jspPage" value="/html/publisher/edit_publisher.jsp" />
<portlet:param name="publisherId" value="<% String.valueOf(publisherId) %>" />
<portlet:param name="redirect" value="<% redirect %>" />
</portlet:renderURL>
<liferay-ui:icon image="edit" url="<% editURL.toString() %>" />
</c:if>
```

WRAP THE THREE ACTIONS IN PERMISSIONS (2)

- ❖ Delete Action:

```
<c:if test="<% permissionChecker.hasPermission(groupId, name, publisherId, ActionKeys.DELETE) %>">
<portlet:actionURL name="deletePublisher" var="deleteURL">
<portlet:param name="publisherId" value="<% String.valueOf(publisherId) %>" />
<portlet:param name="redirect" value="<% redirect %>" />
</portlet:actionURL>
<liferay-ui:icon image="delete" url="<% deleteURL.toString() %>" />
</c:if>
```

WRAP THE THREE ACTIONS IN PERMISSIONS (3)

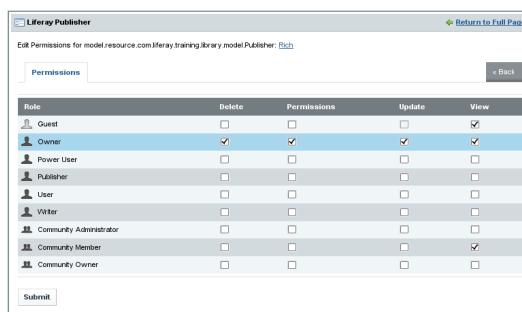
- **Add** the following Permissions Action after the Delete Action but before the `</liferay-ui:icon-menu>`. You will find it in *12-Permissions Action Button*.

```
<c:if test="<% permissionChecker.hasPermission(groupId, name, publisherId, ActionKeys.PERMISSIONS) %>">
    <liferay-security:permissionsURL
        modelResource="<% Publisher.class.getName() %>"
        modelResourceDescription="<% publisher.getName() %>"
        resourcePrimKey="<% String.valueOf(publisherId) %>"
        var="permissionsURL"
    />

    <liferay-ui:icon image="permissions" url="<% permissionsURL %>" />
</c:if>
```

WRAP THE THREE ACTIONS IN PERMISSIONS (4)

- `<liferay-security:permissionsURL>` generates a URL to a permissions view for a resource:
- ModelResource: the model name of the resource
 - ModelResourceDescription: a description for the resource
 - ResourcePrimKey: the primary key of the resource



Role	Delete	Permissions	Update	View
Guest	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Owner	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Power User	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Publisher	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
User	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Writer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Community Administrator	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Community Member	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Community Owner	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

ADD A LINK TO THE TOP LEVEL ACTIONS PERMISSIONS

- **Add** the following code in view.jsp after the add Publisher button. You will find it in *13-Portlet Permissions Button*:

```
<c:if test='<%= hasConfigurePermission %>'>
<liferay-security:permissionsURL
    modelResource="com.liferay.training.library.model"
    modelResourceDescription="library-top-level-actions"
    resourcePrimKey="<% String.valueOf(scopeGroupId) %>"
    var="permissionsURL"
/>
```

PERMISSIONS

- Portlet Resource



- Model Resource (top level actions)



- Model Resource (Resource actions)



IMPLEMENTING THE BOOK PORTLET

➢ The controller for the BookPortlet is virtually identical to the PublisherPortlet.

➢ We will copy the completed Book examples and examine any differences.

➢ **Copy** the *BookPortlet.java* and *BookValidator.java* from:

```
C:\liferay\liferay-developer-studio\solutions-sdk\portlets\library-
portlet\docroot\WEB-INF\src\com\liferay\training\library\portlet\
To
C:\liferay\liferay-developer-studio\plugins-sdk\portlets\library-portlet\docroot\WEB-
INF\src\com\liferay\training\library\portlet\
```

➢ (See next slide for further instructions)

IMPLEMENTING THE BOOK PORTLET 2

➢ **Copy** the docroot\html\book*.jsp files from:

```
C:\liferay\liferay-developer-studio\solutions-sdk\portlets\library-
portlet\docroot\html\book
To
C:\liferay\liferay-developer-studio\plugins-sdk\portlets\library-
portlet\docroot\html\book
```

➢ **Copy** the complete Language.properties file from:

```
C:\liferay\liferay-developer-studio\solutions-sdk\portlets\library-
portlet\docroot\WEB-INF\src\content\
To
C:\liferay\liferay-developer-studio\plugins-sdk\portlets\library-
portlet\docroot\WEB-INF\src\content\
```

LOCALIZATION IN THE BOOK PORTLET

- ▶ Notice the difference in the book_actions.jsp from the publisher portlet:

```
<c:if test="<% permissionChecker.hasPermission
    (groupId, name, bookId, ActionKeys.PERMISSIONS) %>">
    <liferay-security:permissionsURL
        modelResource="<% Book.class.getName() %>"
        modelResourceDescription="<% book.getTitle(locale) %>"
        resourcePrimKey="<% String.valueOf(bookId) %>"
        var="permissionsURL"
    />
    <liferay-ui:icon image="permissions" url="<% permissionsURL %>" />
</c:if>
```

- ▶ We pass in the locale because we localized the title field on the Book entity.
- ▶ This ensures that the title is returned in the locale of the user or the default locale.

TEST THE PORTLET

- ▶ Right-click in the Language.properties file → Liferay → Build Languages
- ▶ Add the Book Portlet to the page if it isn't already there.
- ▶ See that the “Add Publisher” and “Add Book” button is shown when you are logged in but it is hidden when you are a guest.
- ▶ Bonus: Create a community Role that can Delete Books from the Control Panel and assign it to a new user.

Notes:



ADDING ADMINISTRATION PORTLETS TO THE CONTROL PANEL

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

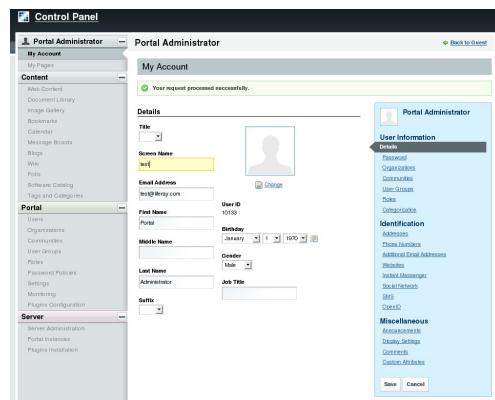
No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

OBJECTIVE

- ❖ This exercise walks you through the steps necessary to add an administration portlet to the Control Panel

INTRODUCTION TO THE CONTROL PANEL

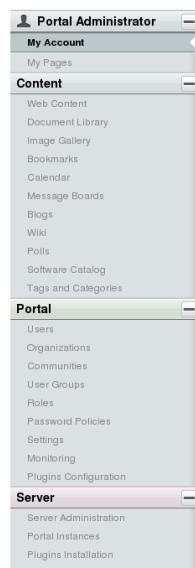
- Unified zone for administering the whole portal
- Accessible by all portal users
 - The tools shown depend on the permissions of the user
 - At the very least users will be able to manage their personal account
- Fully customizable
 - Allows hiding any of the default portlets
 - It's possible to add custom administration portlets



WWW.LIFERAY.COM

LIFERAY

INTRODUCTION TO THE CONTROL PANEL (Cont.)



Divided into 4 areas

- **Personal:** administration of elements specific to the current user
- **Content:** administration of any type of content.
 - Content always belongs to an organization or community
 - A navigation menu allows going to all the organizations or communities that the user can administer
- **Portal:** administration of elements global to the whole portal
- **Server:** administration of elements related to the server or installation and global to all portal instances

WWW.LIFERAY.COM

LIFERAY

liferay-portlet.xml

Adding the library-publisher portlet to the Control Panel

- **Remove** the `<instanceable>...</instanceable>` tag
- **Type** the following right after the `<icon>...</icon>` tag.

```
<control-panel-entry-category>content</control-panel-entry-category>
<control-panel-entry-weight>1.5</control-panel-entry-weight>
```

- Explanation:

- control-panel-entry-category: must be one of personal, content, portal or server
- control-panel-entry-weight: determines the position of the portlet in the menu.
 - Higher weight means lower in the menu.
 - The default items have weights from 1.0 to 11.0
 - Decimals can be used to have full control of the position

liferay-display.xml

- **Modify** the *liferay-display.xml* file and move the entry for liferay-publisher to a new category called `category.hidden`:

```
<category name="category.hidden">
    <portlet id="liferay-publisher" />
</category>
```

- Explanation:

- The category named 'category.hidden' is special. Any portlets added to it will not be available in the "Add Application" menu
- Usually portlets added to the Control Panel are not accessible through the "Add Application Menu" but you don't have to hide a portlet if you want users to be able to access it outside of the Control Panel.

CHECKPOINT

- [Save](#) the portlet
- Enter the portal as the administrator user
- Go to the Control Panel and verify that the Publisher portlet is accessible and fully usable
- Go back to a regular page and try to add the Publisher portlet. It should not be possible
- Log out and enter as a user that does not have the Administrator role
- Go to the Control Panel and verify that the Publisher portlet is not visible

CONTROL PANEL PERMISSIONS (I)

- Portlets added to the Control Panel are only accessible to users with enough privileges.
- By default, the permission check depends on the category where the portlet is placed:
 - Personal: every user will see the portlets in this section
 - Content: The portlet will be visible if the user is an administrator or has either the “Community Owner” role or the “Community Administrator” role
 - Portal or Server: The portlet will be visible if the user has the Administrator role.

CONTROL PANEL PERMISSIONS (II)

- The Permission “ACCESS IN CONTROL PANEL” can be granted to any role for any portlet in the Control Panel.
- It is also possible to implement custom permission checks by:
 - Implementing a class that implements the ControlPanelEntry interface
 - Registering such class through the `<control-panel-entry-class>` element in the liferay-portlet.xml

Notes:



THEMES OVERVIEW

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

OVERVIEW

- ❖ Liferay introduced the concept of “themes” in version 3.5 to give users the ability to customize the look and feel of their portal.
- ❖ The creation of themes has evolved over several major revisions and is now a completely CSS-based design environment.
- ❖ ALL themes in Liferay are based off of one “master” theme that provides a simple but elegant design. By simple modifications in a few CSS files, developers can produce a multitude of themes.

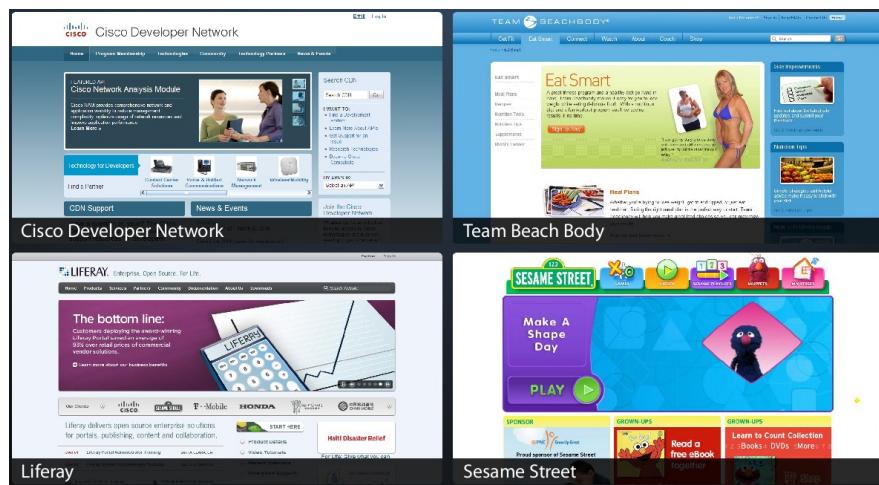
FULL FLEXIBILITY

- When more flexibility is needed Liferay's themes provide control over:
 - All the images used outside of the portlets
 - Images used inside Liferay's portlets (including icons)
 - HTML code of the page (outside of the portlets)
 - HTML code of the portlet box
 - The position and behavior of the top navigation elements
- By combining these possibilities, it's possible to adapt Liferay to very creative designs such as...

WWW.LIFERAY.COM



THEME SAMPLES



WWW.LIFERAY.COM



COLOR SCHEMES

- ❖ Color Schemes are variants of a single theme.
- ❖ Their purpose is to allow the end user to select their preferred variant of the theme.
- ❖ Color Schemes are implemented by providing variations on the CSS files and images of the base theme.

THEMES AVAILABLE

- ❖ Liferay Portal comes bundled with 1 theme.
 - Classic: it's the default theme and is mainly designed for internal pages
- ❖ Liferay's website contains many additional themes:
 - 5+ official themes: those used by Liferay's website and samples for accessibility, WAP, etc:
http://www.liferay.com/web/guest/downloads/official_plugins
 - 60+ community themes: most of them (if not all) available under an OpenSource license:
http://www.liferay.com/web/guest/downloads/community_plugins
- ❖ Portal administrators can browse and install themes in seconds using the Plugin Installer in the Control Panel...

THEMES AVAILABLE

- ❖ More themes can be installed adding these urls to the trusted and untrusted repositories:
 - Trusted Plugin Repository
 - <http://plugins.liferay.com/official>
 - Untrusted Plugin Repository
 - <http://plugins.liferay.com/community>

INSTALLING THEMES

- ❖ To view or change your available repositories:
 - Go to *Control Panel* → *Server* → *Plugins Installation* → *Theme Plugins*
 - Click *Install More Themes*
 - Go to *Configuration* (top tab)
 - Enter one url per line
 - Click *Save* and then click *Browse Repository*

PLUGIN INSTALLER - BROWSING

The screenshot shows the Liferay Control Panel with the 'Server > Plugins Installation' path selected. The main title is 'Plugin Installer'. Below it, there are tabs: 'Portlet Plugins' (selected), 'Theme Plugins', 'Layout Template Plugins', 'Hook Plugins', and 'Web Plugins'. A search bar and filters for 'Keywords', 'Tag', 'Repository', and 'Install Status' are present. The results table has columns: 'Theme Plugin', 'Trusted', 'Tags', 'Installed Version', 'Available Version', and 'Modified Date'. One result is listed: 'Bitter Sweet 0.0.3.1' by 'Arcsin'. The status indicates it's 'Not Installed or Out of Date'. A note at the bottom says 'Showing 1 result.' and 'List of plugins was last refreshed on 8/4/10 7:01 PM.'

WWW.LIFERAY.COM



PLUGIN INSTALLER - INSTALLING

The screenshot shows the Liferay Control Panel with the 'Server > Plugins Installation' path selected. The main title is 'Plugin Installer'. A green success message says 'Your request processed successfully.' Below it, another message says 'The plugin was downloaded successfully and is now being installed.' The plugin details for 'Bitter Sweet' are shown: Name: (v6.0.3.1), Author: Arcsin, Types: Theme, Tags: 2-column, Licenses: (Open Source), Liferay Versions: 6.0.3+, Repository: http://plugins.liferay.com/community (Untrusted). A note says 'Short Description: Line flavoured template with two columns.' and 'Change Log: Adapted to the latest version of Liferay.' An image of the plugin theme is displayed, and an 'Install' button is visible.

WWW.LIFERAY.COM



THE THEME IS NOW AVAILABLE

The screenshot shows the Liferay Control Panel's "Look and Feel" section under "Themes". It displays the "Current Theme" as "Classic" with a description: "Portlets, themes, and layout templates included with Liferay Portal." It also shows three color schemes: Blue, Green, and Orange. Below this, there is a section for "Available Themes" which lists "7Cogs" and "Bitter Sweet".

WWW.LIFERAY.COM

LIFERAY.

KEEP UPDATED

- ❖ The update manager helps keep themes (and other plugins) up to date

The screenshot shows the Liferay Control Panel's "Update Manager" interface. It lists 12 results, all of which are "Up to Date" and have an "Uninstalled" status. The plugins listed include various portlets and themes such as "7Cogs Mobile Theme", "7Cogs Theme", "Chat Portlet", "Geolocation Portlet", "Mail Portlet", "OpenSocial Portlet", "WDRP Portlet", "Web Form Portlet", and "World of Liferay Portlet".

WWW.LIFERAY.COM

LIFERAY.

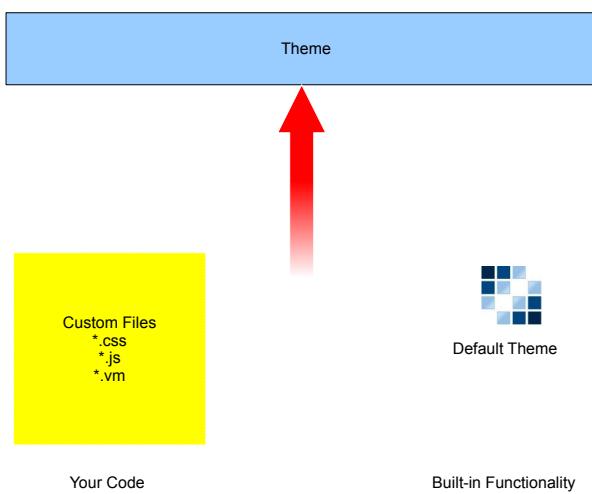
HOW SIMPLE IS IT REALLY?

- ❖ All themes are now based on differences from the default theme.
- ❖ This reduces the number of files and the complexity of custom themes.
- ❖ It also simplifies automatic updating of themes to new versions of the product.
- ❖ Theme projects contain only the css files, images, javascript files or templates that are different from the ones in the default theme.
- ❖ Themes can now be as simple as a few custom .css and image files.
- ❖ The source of most (or all) of the available themes is also available and can be used under Open Source licenses.
- ❖ Liferay provides the Plugins SDK to help manage the creation of themes.

WWW.LIFERAY.COM

 LIFERAY.

THEME ARCHITECTURE



WWW.LIFERAY.COM

 LIFERAY.

DEVELOPMENT APPROACH

- ❖ The recommended method for developing themes is to use the Plugins SDK
 - It manages the differences between existing themes
 - Creates a plugin WAR file with the resulting theme
 - Is able to install the theme in a local installation
- ❖ The Liferay IDE also provides support for Theme Development if you are using Eclipse

THEME DEVELOPMENT TECHNOLOGIES

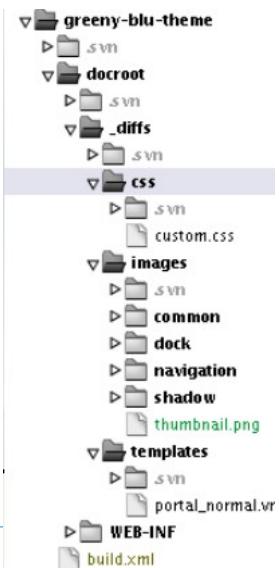
- ❖ The main technologies used in theme development are:
 - CSS: Provide a great degree of control over the look and feel of the page
 - Velocity (or optionally JSPs): A series of templates offer control over the HTML produced
 - (X)HTML: The most common output of a theme template

BASE THEMES

- ❖ Liferay includes two template themes that are the basis for every other theme:
 - _unstyled: contains the default templates and images. Its CSS files only contain placeholders to add formatting rules
 - _styled: adds CSS files with formatting rules
- ❖ The classic theme is based on _unstyled and _styled and adds the different color schemes
- ❖ The base themes can be found in Liferay's source code in:
`LIFERAY_SOURCE\portal-web\docroot\html\themes`

STRUCTURE OF A THEME IN THE PLUGINS SDK

- ❖ All custom layout code goes in the _diffs folder
- ❖ When the theme is deployed, three things happen:
 1. The “_unstyled” base theme is copied first
 2. The “_styled” base theme is copied next
 3. Any files in the _diffs folder from your theme are copied on top of those files



OVERVIEW OF FILES - CSS

- ❖ application.css - styling related elements used in applications (tabs, pickers, ...)
- ❖ base.css - has the basic styling that applies to the entire portal
- ❖ custom.css - is where your overriding and custom css should go
- ❖ dockbar - has styling related to the dock bar
- ❖ forms - has styling related to forms
- ❖ layout.css - has the styling related to layouts
- ❖ main.css - includes all the other css
- ❖ navigation.css - styles the navigation
- ❖ portlet.css - styles the portlet

OVERVIEW OF FILES - IMAGES

- ❖ common/ - this folder contains all general images
- ❖ dockbar/ - this folder contains images for styling the dockbar
- ❖ messages/ - this folder contains the images for the different portal messages
- ❖ navigation/ - this folder contains the images for the navigation styling
- ❖ portlet/ - this folder contains images related to the portlet decoration
- ❖ ... other portlet specific image folders

OVERVIEW OF FILES - TEMPLATES

- ❖ init.vm - this file loads the variables used by the other templates
- ❖ init_custom.vm - this file is used to override or store new variables
- ❖ navigation.vm - this file contains the navigation HTML
- ❖ portal_normal.vm - this file is the main "index" file that contains the base HTML
- ❖ portal_popup.vm - this file is similar to portal_normal.vm except that it is shown in popups
- ❖ portlet.vm - this file contains the box around all portlets
- ❖ The recommended format for templates is velocity. FreeMarker and JSP can also be used, if that's the case change the 'vm' extension in the files above to 'ftl' or 'jsp'.

OVERVIEW OF FILES - OTHER

- ❖ javascript/
 - javascript.js - this file can be used to place javascript code that will be executed after the page is loaded or after each portlet is loaded
- ❖ WEB-INF/
 - liferay-look-and-feel.properties - this is the file that contains the properties and meta data of the theme

RECOMMENDED TOOLS

- Firebug is a Firefox extension that allows you view the CSS that is being applied and inherited, edit the page's CSS live, run Javascript from a command line, debug Javascript with breakpoints, as well as profiling tools and tracking of AJAX requests.
- Firebug is freely available, and can be downloaded at:
<http://getfirebug.com>

WWW.LIFERAY.COM



RECOMMENDED TOOLS

- The IE Developer Toolbar is the closest equivalent to Firebug for Internet Explorer. While not anywhere near as powerful, it still is great for viewing what CSS is being applied to an object, and editing that CSS live.
- The IE Developer Toolbar is available from Microsoft's website:

<http://www.microsoft.com>

WWW.LIFERAY.COM



RECOMMENDED TOOLS

- ❖ DebugBar is a bit like the IE Developer Toolbar and Firebug for Internet Explorer. It is quite powerful as it allows you to also run Javascript from a command line and edit CSS live
- ❖ Debug Bar is available at:

<http://debugbar.com>

RECOMMENDED TOOLS

- ❖ Google Chrome has tools similar to Firebug and DebugBar built into the browser.
- ❖ Right-clicking on an element will give you an “Inspect Element” option which gives you an advanced view of the source code for a page or an element on the page.

Notes: _____



THEME DEVELOPMENT EXERCISE

Copyright © 2000 – 2011 Liferay, Inc.

All Rights Reserved.

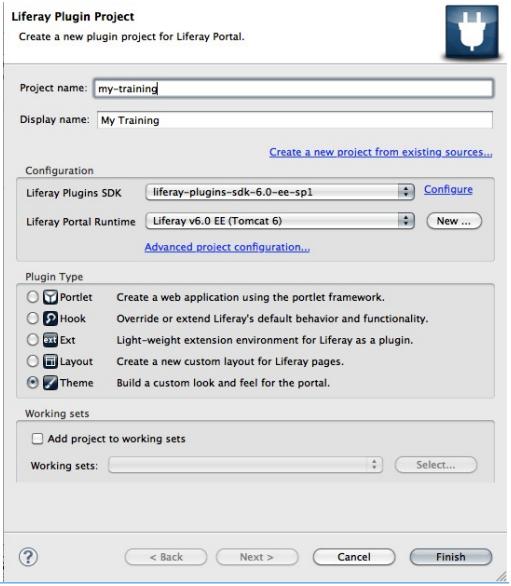
No material may be reproduced electronically or in print, duplicated,
copied, sold, resold, or otherwise exploited for any commercial purpose
without express written consent of Liferay, Inc.

PURPOSE

- ❖ Create a new theme using the Liferay IDE
- ❖ Deploy it to an existing Liferay Portal installation
- ❖ Customize several aspects of the base themes

CREATE MY TRAINING THEME

1. Go to File > New Project... > Liferay > Liferay Project
2. Project name:
 - my-training
3. Display name:
 - My Training
4. Select the SDK and Liferay runtime you have configured
5. Select the *Theme* plug-in type



WWW.LIFERAY.COM

LIFERAY.

CUSTOMIZING YOUR THEME

- What has happened?
 - The plugin SDK has created a theme using the *_unstyled* and *_styled* base themes
 - Since you didn't have any customization (yet) your theme is quite barebones
- The Liferay IDE is configured to continuously build your Theme project any time it detects a change.

WWW.LIFERAY.COM

LIFERAY.

THEME CONSTRUCTION

1. To see this in action, let's create a new folder in `_diffs` directory called `css`.
2. Watch the console and the project structure in the Package Explorer to see the changes.
 - ✓ Files have been copied from the `_unstyled` and `_styled` base themes to your new project.

CHECKPOINT 1

- ❖ Let's deploy the theme to see what we have as a starting point.
 - ❖ The process to deploy a theme is the same as deploying a portlet.
 - ❖ Click and drag the project name from the Package Explorer to the server name in the Servers view.
 - ❖ *Ensure your tomcat log shows the theme is deployed and registered.*
1. Login to `http://localhost:8080` and select `Manage → Sitemap` from the Liferay Dockbar.
 2. Click on `Look and Feel` and you should see your new theme in the list.
 3. Select `My Training` and your theme should immediately change, but it will be quite empty.

THE _diffs DIRECTORY

- The bulk of the theme is in the following directory
my-training-theme\docroot\
- The Liferay IDE auto-build and deploy mechanism has done two things:
 - The base themes are copied to that directory
 - Any files in the _diffs directory are copied on top of those files



WWW.LIFERAY.COM

LIFERAY.

CUSTOMIZING YOUR THEME

- It is also possible to develop our theme from an existing one.
- For training purposes, we will be extending the Liferay classic theme.
 1. Modify the property "theme.parent" to "classic" in build.xml
 2. To trigger the auto build process, we'll create a new folder in the _diffs directory called *templates*
 3. Wait for the Liferay IDE to auto-build and deploy your theme and then refresh your browser.
 - ✓ Your theme will be the very similar to the classic theme
- In practice, you should only extend Themes that were created with extension in mind. The Liferay Classic theme was **not**, so using it as a starting point could cause problems when upgrading Liferay.

WWW.LIFERAY.COM

LIFERAY.

VELOCITY

- Velocity is a templating language that Liferay uses to allow developers to use variables, conditional statements and loops alongside the HTML.
- Velocity variables look like the following: \$company_url, \$has_navigation, etc.
- In Velocity, all variables are preceded by a \$ sign, and all directives, statements and macros begin with a # sign, like so:

```
#if ($has_navigation)
#parse ("$full_templates_path/navigation.vm")
#end
```
- A complete guide and a full reference of the velocity templating language can be found in:
 - <http://velocity.apache.org/engine/devel/user-guide.html>
 - <http://velocity.apache.org/engine/devel/vtl-reference-guide.html>

WWW.LIFERAY.COM



EXERCISE 2a: UPDATE THE DOCK

1. Copy the file `portal_normal.vm`

```
from:
my-training-theme\docroot\templates\
to:
my-training-theme\docroot\_diffs\templates
```

2. Replace the link:

```
#if(!$is_signed_in)
<a href="$sign_in_url" id="sign-in" rel="nofollow"> $sign_in_text </a>
#end
```

with a button (`o1-portal_normal.vm`):

```
#if(!$is_signed_in)
<form action="$sign_in_url" method="get">
  <input id="sign-in" type="submit" value="Log In" />
</form>
#end
```

WWW.LIFERAY.COM



EXERCISE 2b: UPDATE THE NAVIGATION

1. Copy the file **custom.css** into `my-training-theme_diffs\css` from: `my-training-theme\docroot\css` and enter the following code:

```
.portlet {  
    background: #ffff;  
    border: 1px solid #9AC1E7;  
    margin: 0 0 10px;  
    padding: 3px;  
}  
.portlet-topper {  
    border-bottom: 1px solid #AEBBBC;  
    padding-right: 40px;  
}
```

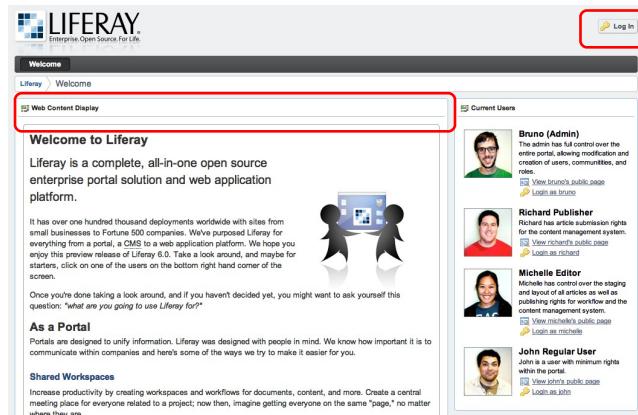
- Though you can modify copies of `navigation.css` and `base.css` accordingly, we recommend changes be added to `custom.css` (read last by Liferay). Because of the nature of CSS, the styles loaded last (i.e., `custom.css`) will overwrite the specific styles read in earlier files.

WWW.LIFERAY.COM

LIFERAY

CHECKPOINT 2

- When you save, the theme should autodeploy.
- Ensure your tomcat log shows the theme is deployed and registered.
- Refresh your browser and you should see the changes reflected.



WWW.LIFERAY.COM

LIFERAY

SUMMARY

- ❖ Changes should **always** be made in the _diffs directory. Otherwise, your files will be deleted in your next ant deploy.
- ❖ Since styles can be overwritten in CSS, it is advised to make CSS changes in the custom.css file only.

Notes: _____



LAYOUT TEMPLATES

Copyright © 2000 – 2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

WHAT IS A LAYOUT TEMPLATE?

- Layout templates are similar to themes in that they're used to control the way pages look in Liferay.
- Layout templates are used to control how portlets can be arranged on the page, and are usually a grid-like structure, and most often are created with a combination of Velocity, HTML markup, and CSS.
- Layout templates, like themes, are hot deployable as plugins.

CREATING A LAYOUT TEMPLATE

- Layout templates are also created with the Plugins SDK.
- They are stored inside the layouttpl subdirectory.
- The structure of a Layout Template is:
 - my-template.tpl: Velocity template that generates the HTML struture of the template
 - my-template.wap.tpl: Variant template for mobile devices
 - my-template.png: Representation of the template. It will be shown to the user as a preview.
 - WEB-INF/liferay-layout-templates.xml: Definition file. Used to set the name of the layout template and the location of the three files above. It is possible to define several layout templates in one file.

PURPOSE

- Create a new layout template using the Liferay IDE
- Deploy it to an existing Liferay Portal installation
- Customize several aspects of layout template

CREATE TRAINING-3-COLUMN LAYOUT

1. Go to File > New Project... > Liferay > Liferay Project
2. Project name:
 - training-3-column
3. Display name:
 - Training 3 Column
4. Select the SDK and Liferay runtime you have configured
5. Select the *Layout* plug-in type

✓ Click *Finish*

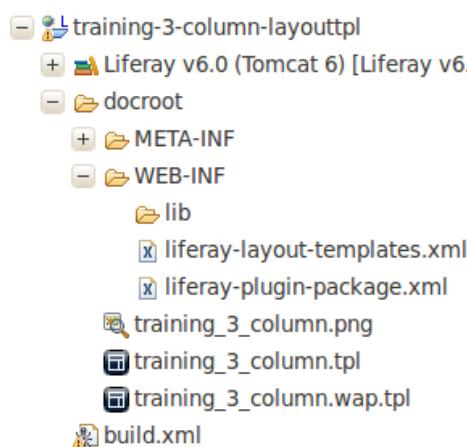


WWW.LIFERAY.COM

LIFERAY

CHECKPOINT

- After the Liferay IDE has created the template, expand the project in the package explorer and examine the files and folders created.
- The .tpl files are the layout templates themselves.
 - They contain html, css, and velocity.
 - They can be edited with any text editor.
 - By default you get one template for traditional browsers and one template for WAP based browsers.
 - The .png file is the thumbnail that you see when selecting the *Page Layout* from the Dock Bar.
 - You are responsible for creating the actual thumbnail. You can use the .png as a starting point.

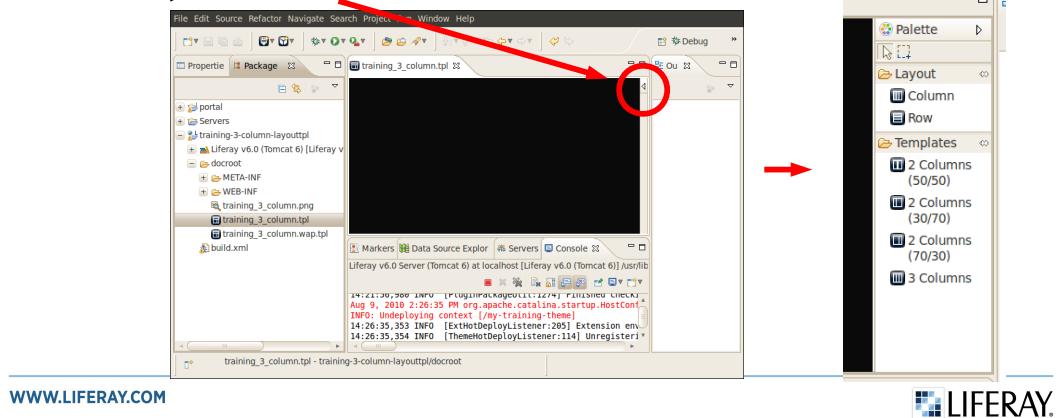


WWW.LIFERAY.COM

LIFERAY

EDIT THE LAYOUT TEMPLATE

1. To edit a template, double click on the .tpl you wish to work with. For training, we'll edit the *training_3_column.tpl* file.
2. The Liferay IDE provides a palette to graphically manipulate the template. To expand the palette, click on the arrow in the upper right hand corner of the layout editor.



CREATE LAYOUT

- With the Palette open, you can now drag and drop columns and rows to graphically build your desired Layout Template.
- 1. Drag 3 columns on the editor pane.
- 2. Click and drag to resize the columns so the first column is 25%, the second column is 50%, and the third column is 25%.
- Sometimes it can be difficult to manipulate the template using the graphical editor and you may need to work with the text editor.
- To change to the text editor, just click on the "Source" tab in the view pane.

EDIT LAYOUT

- Examining the template, you can see a combination of HTML, CSS, and Velocity.
- The place holders that set where portlets can be dropped are Velocity commands and they look like this:

```
$processor.processColumn("column-1", "portlet-column-content portlet-column-content-first")
```

- All layouts must have at least this line:

```
$processor.processColumn("column-1")
```

- After your layout is finished and saved, you can deploy the template the same way that we deployed the portlets and the theme.
- Once the layout is registered, you can select the layout by going to the Dock Bar, and choosing *Manage – Page Layouts*. Your layout should be in the list, and once you choose it, it should become the grid that you've selected.

ADVANCED LAYOUT TEMPLATE EXERCISE

- In many Liferay installations, it's a common requirement to have certain portlets always appear on every page.
- For example, you might want a Liferay Alert portlet to appear on every page, if there is ever a reason to display an emergency alert message. Or, perhaps every page should have a common navigation mechanism that runs along the left hand column of the page.
- Layout Templates give you a way to do this by embedding portlets into the template.
- Core Liferay portlets and plugin portlets that you create yourself can be embedded in Layout Templates.

ADVANCED LAYOUT TEMPLATE EXERCISE

For this exercise, we will embed the Navigation portlet in the top of the left most column and we'll embed the Search portlet in the top of the right most column.

The Liferay IDE does not currently support this through the graphical editor so we will have to open our *training-3-column.tpl* file with the text editor.

1. *Insert* the following line (*o1-Embed Navigation Portlet*) right above line that contains the \$processor.processColumn directive for the first column.

```
$processor.processPortlet("71_INSTANCE_abc1")
```

2. *Insert* the following line (*o2-Embed Search Portlet*) right above line that contains the \$processor.processColumn directive for the third column.

```
$processor.processPortlet("3")
```

Why the difference between the two commands? The Navigation portlet is an instanceable portlet. When adding an instanceable portlet, you need to provide a unique 4 alphanumeric instance value.

ADVANCED LAYOUT TEMPLATE EXERCISE

Just like templates, the Liferay IDE continuously builds and deploys layout templates.

Once you have saved your file, watch the console to ensure it registers successfully and then refresh your page.

Notice the portlets that have been embedded can not be deleted or moved, they can only be minimized.

Additionally, if you try to move a portlet above one of these portlets you'll see that you can not.

If you are attempting to embed a non-core portlet (i.e. a portlet developed with the Plugins SDK) you'll need to use the *Fully Qualified Portlet ID (FQPI)* which takes the form of

```
portletId_WAR_webapplicationcontext
```

For example, to embed the Pitcher Portlet from the IPC Baseball example, you'd use the following:

```
$processor.processPortlet("pitcherportlet_WAR_ipcbaseballportlet_INSTANCE_abc2")
```

Notes:



ADVANCED THEMING

Copyright © 2000 – 2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

CONTENTS

- ❖ Color schemes
- ❖ Settings
- ❖ Handling differences among browsers
- ❖ Predefined velocity variables and macros
- ❖ Embedding portlets into a theme
- ❖ Styling layouts and portlets from a theme

COLOR SCHEMES

- Color schemes are variations of a given theme.
 - They may have specific images and CSS that override the ones defined in the theme.
 - To define color scheme it's necessary to create a new file within WEB-INF.
 - liferay-look-and-feel.xml: contains information necessary for Liferay's plugin management system
 - Tip: get an initial version of this file from the deployment folder of the application server to obtain it from within the WEB-INF directory. Liferay created it automatically when the them was first deployed.
 - Using Color Schemes creatively allows offering more options to the end users with very little work.
- Example:
<http://www.liferay.com/web/guest/community/wiki/-/wiki/Main/Theme+variations+using+color+schemes>

WWW.LIFERAY.COM



COLOR SCHEMES - liferay-look-and-feel.xml

```
<look-and-feel>
<theme id="my-training" name="My Training">
    <color-scheme id="01" name="Blue">
        <css-class>blue</css-class>
        <color-scheme-images-path>${images-path}/color_schemes/${css-
class}</color-scheme-images-path>
    </color-scheme>
    <color-scheme id="02" name="Red">
        <css-class>red</css-class>
        <color-scheme-images-path>${images-path}/color_schemes/${css-
class}</color-scheme-images-path>
    </color-scheme>
</theme>
</look-and-feel>
```

WWW.LIFERAY.COM



SETTINGS

- Each theme can define a set of settings to make it configurable.
- The settings are defined in the liferay-look-and-feel.xml using the following syntax:

```
<settings>
  <setting key="my-setting" value="my-value" />
  ...
</settings>
```
- These settings can be accessed in the theme templates using the following code:

```
$theme.getSetting("my-setting")
```
- There are two predefined settings that are read by core portlets:
 - portlet-setup-show-borders-default: if set to false the portal will turn off borders by default for all the portlets. The default is true.
 - bullet-style-options: The value must be a comma separated list of valid bullet styles to be used by the navigation portlet. The default is an empty list, so no option will be shown in the portlet configuration screen.

HANDLING DIFFERENCES AMONG BROWSERS

- Different browsers render CSS formatting rules in slightly different ways.
- This can be very frustrating and time consuming for website designers.
- A typical solution has been to use CSS-hacks.
 - But often have to be redone for new versions of each browser
- Liferay Themes offer an easy to use and maintainable solution: custom classes for each main browser that can be prepended to existing selectors:
 - .ie, .ie6, .ie7, .safari, .gecko, .konqueror, etc
- It can also detect the operating system or if JavaScript is enabled or not:
 - .js: only present if it's enabled
 - .win, .mac, .linux
- The selectors can be combined
 - .win.gecko

PREDEFINED VELOCITY VARIABLES

- Liferay provides several useful variables that can be used from the velocity templates:
- Context objects:
 - \$theme_display: contains information about the portal context of the current request.
 - \$portlet_display: contains information about the current portlet and is meant to be used within the portlet.vm template.
 - \$page: the current page. The current community can be obtained using \$page.getGroup()
 - \$theme: utility methods
 - \$serviceLocator: allows invoking Liferay's service layer with findService(SERVICE_NAME)

MORE PREDEFINED VELOCITY VARIABLES

- Navigation
 - \$nav_items: contains all the pages and can be iterated with #foreach. Each object has the following methods:
 - isSelected(), getURL(), getTarget(), getName(), hasChildren(), getChildren()
 - \$sign_in_url, \$sign_out_url
- User information
 - \$user_greeting, \$user_id, \$user_email_address, \$user_name, \$language_id, etc.
- Paths and URLs:
 - \$css_folder, \$images_folder, \$javascript_folder, \$company_logo, \$company_url, etc.

PREDEFINED VELOCITY VARIABLES (Cont.)

➤ Macros:

- #css(FILE_NAME): Creates an HTML link element to a CSS
- #js(FILE_NAME): Creates an HTML script element
- #language(LANGUAGE_KEY): writes a localized message with the given key

➤ More information:

- <http://content.liferay.com/4.3/misc/theme-api-4.3.0.html>
- It is also possible to see the definitions of several variables in Liferay's base themes:
`/html/themes/_unstyled/templates/init.vm`

EMBEDDING PORTLETS INTO A THEME

- Sometimes, you'll wish to embed a portlet into a theme. For instance, this may be if you wish to always have a Journal Content article in the footer of your page.
- This is done with the `$theme.runtime()` call.
- The important part to remember is the difference between instanceable portlets and regular portlets.
- An instanceable portlet is one that can appear multiple times on the same page. If your portlet is instanceable, (such as the Journal Content portlet), you will need to pass in not only the portlet id, but a string at the end of it.

EMBEDDING PORTLETS INTO A THEME (Cont.)

- To embed a non-instanceable portlet such as the Image Gallery, you would use:

```
$theme.runtime("31")
```

- To embed an instanceable portlet, you would need to have the portlet id, followed by the word INSTANCE, and a random 4 letter code, eg:

```
$theme.runtime("56_INSTANCE_abcd")
```

STYLING LAYOUTS AND PORTLETS FROM A THEME

- Sometimes you'll need to do something custom, such as styling a certain portlet on a page, or all portlets within a certain column.
- In your layout, you can give a layout any unique id, such as 'id="blogColumn"', and in your CSS, you can say that any portlet inside of this column will have a blue background with code like:

```
#blogColumn .portlet {  
background: #06f;  
}
```

STYLING LAYOUTS AND PORTLETS FROM A THEME (Cont.)

- If you need to stylize a certain type of portlet, you would do something like:

```
.portlet-breadcrumb {  
    border: 2px solid #fc0;  
}
```

or

```
.portlet-tagged-content {  
    background: #ccc;  
}
```



BEST PRACTICES

Copyright © 2000 – 2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

THEMES

- Try to use semantic HTML whenever possible. Avoid using divs when a p tag will do.
- Use divs and spans appropriately. Divs can contain other block level elements, such as p's and other divs, but spans can only contain inline elements, such as images, anchors, etc.
- An ID can only appear once on a page. If you need multiple identifying elements, use a class.
- Avoid "class-itis", which is placing a class on every child of an element, and use elements and class names sparingly.

CSS

- Avoid using CSS hacks to target different browsers. Liferay provides browser selectors that allow you to target both a specific browser, as well as a specific version of that browser, using a simple selector namespace, such as `.ie`, `.ie6`, `.firefox`, `.safari`, etc.
- Use dashes in your CSS class names instead of underscores or camelCasing.
- Use camelCasing for your ID's

JAVASCRIPT

- Avoid using global variables in your javascript files. This can break other scripts inside of the portal, as well as cause endless amounts of conflicts, and make troubleshooting difficult.
- To do this, make sure you only define variables inside of a function, and always place "var" in front of that variable when defining it.
- For your theme, it is recommended to use JSON style to contain all of your functions, properties, values, etc.
- For example, if your theme is named "My Training Theme", in your javascript, you would place all of your methods, properties, etc inside of one object like so:

```
var MyTraining = {  
    method1: function(){  
        this.method2();  
    },  
    method2: function(){}
};
```

Notes:



CUSTOMIZING LIFERAY WITH HOOKS

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

HOOKS OVERVIEW

- ❖ Hooks were introduced in Liferay 5.1 as an alternate development approach to the Liferay Extension Environment.
- ❖ Hooks are a type of Liferay Plugin and were designed to be smaller in size than the Extension Environment and more modular.
- ❖ Hooks were also designed to be hot deployable.
- ❖ Over the last several versions of Liferay, the capabilities of Hooks have been expanded and are generally preferred over working in the Extension Environment or the EXT Plugin whenever possible.

HOOKS ADVANTAGES

- ❖ Hot deployable
 - Changes are made available as soon as they are deployed
 - Changes are removed as soon as they are undeployed
- ❖ Useful as a light weight customization archive. Examples:
 - Seven Cogs
 - Social Office
 - TweetRay

WWW.LIFERAY.COM



CUSTOMIZABLE ELEMENTS

- ❖ The following elements can be customized from a hook:
 - Change the configuration
 - Customize language keys
 - Add or overwrite JSP files
 - Provide custom services implementations

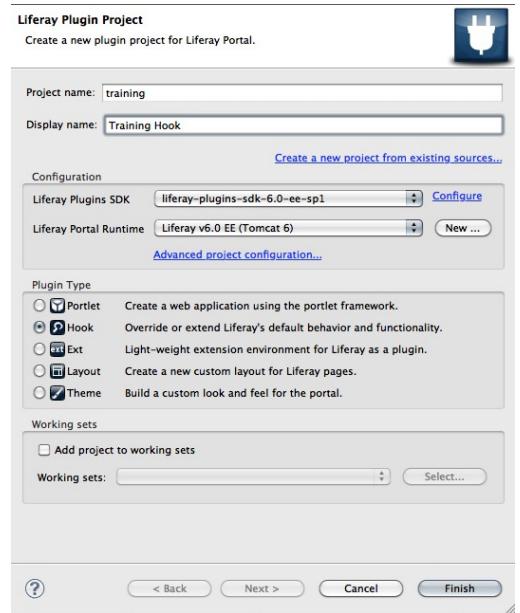
WWW.LIFERAY.COM



CREATE TRAINING HOOK

1. Go to File > New Project... > Liferay >Liferay Plug-in Project
2. Project name:
 - » training
3. Display name:
 - » Training Hook
4. Select the SDK and Liferay runtime you have configured
5. Select the Hook plug-in type

✓ Click *Finish*



WWW.LIFERAY.COM

LIFERAY

LIFERAY-HOOK.XML

- » At this point, we have created a Liferay Hook Plugin Project, but we haven't actually created a Hook yet.
- » Have a look at *liferay-hook.xml*, the descriptor of the hook.
- » The *liferay-hook.xml* file is used to identify the types of hooks that this hook plugin will contain.
- » To see a listing of the different types of hooks allowed, refer to the http://www.liferay.com/dtd/liferay-hook_6_0_0.dtd
- » You can create different Hook Plugins for different purposes, or you can combine several different Hooks into a single project
- » In this exercise we will be adding several Hooks to a single Hook Plugin project.

WWW.LIFERAY.COM

LIFERAY

CUSTOMIZING PROPERTIES

- The first Hook we will implement will be to customize the *portal.properties* and to implement a custom post-login action.
- Not all the properties can be modified yet by a hook—there is a limited set defined in DTD that are most popular and applicable.
 - Default landing page, enable/disable public and private user pages, ...
 - Events (Startup, service, login, logout...etc), model listeners...
- Simple values are overwritten.
- List values are appended.
- Properties can be overwritten in the *portal.properties* file defined in *liferay-hook.xml*

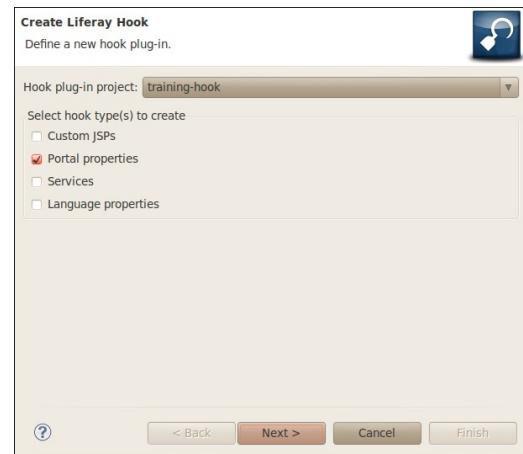
CUSTOMIZING PROPERTIES

- Liferay Portal provides a pluggable mechanism whereby you can register classes that get fired on pre-login and post-login.
- Event classes are specified in *portal.properties* by overriding the following key values:
 - `login.events.pre`
 - `login.events.post`
- To create a cookie with login info, we'll create an event class and append the class name to `login.events.post`.

CREATE CUSTOM POST LOGIN ACTION

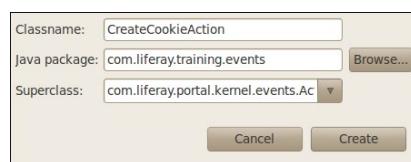
1. Go to File > New... > Liferay Hook
2. Select the *training-hook* as the *Hook plug-in project*.
3. Check the *Portal properties* check box

✓ Click *Next*



CREATE CUSTOM POST LOGIN ACTION

1. Accept the default path for the *Portal properties file*
2. Click the *Add...* button to *Define actions to be executed on portal events*
3. For *Event*, click *Select...* and choose *login.events.post*
4. For *Class*, click *New* and enter
 - Classname: *CreateCookieAction*
 - Java Package: *com.liferay.training.events*
 - Superclass: *com.liferay.portal.kernel.events.Action*



✓ Click *Create*, then *Ok*, then *Finish*

CREATE CUSTOM POST LOGIN ACTION

1. **Insert** the contents of the snippet *01-Create Cookie Action* into the *CreateCookieAction.java* file, replacing the auto generated public constructor and the empty *run* method.
2. Add the following imports:

```
> import com.liferay.portal.kernel.util.StringPool;
> import com.liferay.portal.util.PortalUtil;
> import com.liferay.portal.kernel.util.PropsUtil;
> import com.liferay.portal.kernel.util.Validator;
> import javax.servlet.http.Cookie;
> import javax.servlet.http.HttpServletRequest;
> import javax.servlet.http.HttpServletResponse;
> import org.apache.commons.logging.Log;
> import org.apache.commons.logging.LogFactoryUtil;
```

CREATE CUSTOM POST LOGIN ACTION

1. Save and deploy your hook
- ✓ After login, you should see a line similar to the following in the console:

21:55:18,328 Added TRAINING_COOKIE value to response:
companyId=1,userId=2

CUSTOMIZING LANGUAGE KEYS

- Liferay Hooks DTD allows for multiple entries so that you can modify language bundles per country and per key.
- Specify a bundle with a language code
- Overlays existing language bundles

CUSTOMIZING LANGUAGE KEYS

1. Go to File > New... > Liferay Hook
2. Select the *training-hook* as the *Hook plug-in project*.
3. Check the *Language properties* check box
- ✓ Click *Next*



CUSTOMIZING LANGUAGE KEYS

1. Accept the default path for the *Content folder*
2. Click the *Add...* button to add *Language property files*:
3. Add the following files:

Language.properties
Language_en.properties
Language_es.properties

- ✓ Click *Finish*
- The Liferay IDE has created the three files that will contain our customized language keys,
- All we need to do is populate the language keys.

CUSTOMIZING LANGUAGE KEYS

1. Insert the contents of *o2-Language Properties* to the *docroot/WEB-INF/src/content/Language.properties* file.
 2. Insert the contents of *o3-English Language Properties* file to the *docroot/WEB-INF/src/content/Language_en.properties* file.
 3. Insert the contents of *o4-Spanish Language Properties* to *docroot/WEB-INF/src/content/Language_es.properties* file.
- ✓ The Liferay IDE will pick up the changes as you save your files and will automatically deploy the hook. You may need to refresh your browser to see the changes.
 - To see the Spanish translation, add the Language portlet to your layout and click on the flag of Spain.

CUSTOMIZING JSP FILES

- Override JSPs in the portal
 - Changes are reversed when undeployed
 - Copies existing xxx.jsp to xxx.portal.jsp
- Use liferay-util:buffer to minimize upgrade headaches when customizing a JSP
- Certain portlets have *-ext.jsp patterns already

WWW.LIFERAY.COM



OVERRIDE TERMS OF USE

- When a user logs in to Liferay Portal for the first time, a *Terms of Use* page is displayed.
- The user may not login until the *I Agree* button is clicked.

A screenshot of a web browser displaying the Liferay Terms of Use page. The page has a blue header with the Liferay logo and the text "Welcome, Doris Griffin". Below the header is a search bar. The main content area is titled "Terms of Use". It contains several paragraphs of text about the terms of use, including sections for "Acceptance of Agreement", "Copyright", "Service Marks", and "Limited Right to Use". At the bottom of the page are two buttons: "I Agree" and "I Disagree". A red arrow points from the text in the "Limited Right to Use" section to the "I Agree" button.

Welcome to our site. We maintain this web site as a service to our members. By using our site, you are agreeing to comply with and be bound by the following terms of use. Please review the following terms carefully. If you do not agree to these terms, you should not use this site.

Acceptance of Agreement

You agree that the conditions outlined in this Terms of Use Agreement ("Agreement") with respect to our site (the "Site"). This Agreement constitutes the entire and only agreement between us and you, and supersedes all prior or contemporaneous agreements, representations, warranties and understandings whether written or oral, that may exist between us and you concerning the products or services provided by or through the Site, and the subject matter of this Agreement. This Agreement may be amended at any time by us. Please check this page from time to time without specific notice to you. The most recent version of this Agreement will be posted on the Site, and you should review this Agreement prior to using the Site.

Copyright

The trademarks, organization, graphics, design, compilation, magnetic translation, digital conversion and other materials related to the Site are protected under applicable copyrights, trademarks and other proprietary (including but not limited to intellectual property) rights. The copying, redistribution, use or publication by you of any material in any part of the Site, except as allowed by Section 4, is strictly prohibited. You do not acquire any ownership interest in any material or document you may obtain through the Site. The passing of information or materials on the Site does not constitute a waiver of any right in such information and materials.

Service Marks

Products and names mentioned on the Site may be trademarks of their respective owners.

Limited Right to Use

The viewing, printing or downloading of any content, graphic, form or document from the Site grants you only a limited, nonexclusive license for your own personal use, and not for redistribution, distribution, exhibition, sale or generation of derivative works.

I Agree I Disagree

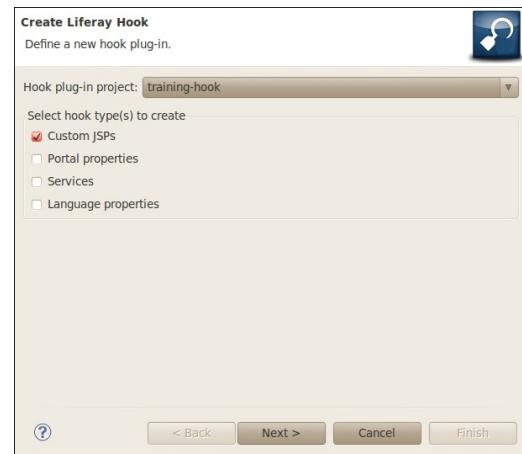
WWW.LIFERAY.COM



CUSTOMIZING LANGUAGE KEYS

1. Go to File > New... > Liferay Hook
2. Select the *training-hook* as the *Hook plug-in project*.
3. Check the *Custom JSPs* check box

✓ Click *Next*



OVERRIDE TERMS OF USE

1. Accept the default path for the *Custom JSP folder*
 2. Click the *Add from Liferay...* button to override an existing JSP page
 3. In the search box, begin typing the word *terms* and select *terms_of_use.jsp* when it appears in the results window. You can also browse to *html → portal → terms_of_use.jsp*
- ✓ Click *Ok, then Finish*
- The Liferay IDE has created a copy of the *terms_of_use.jsp* in the *docroot/custom_jsp/html/portal* directory.
 - Make a change to this file and save the changes. The Liferay IDE will automatically deploy your hook.
 - You will need to create a new user and log in with that user to be presented with the Terms of Use page again.

OVERRIDE TERMS OF USE

- Browse to your tomcat bundle too see what has happened with the original file:
 - C:\liferay\bundles\liferay-portal-[version]\tomcat-6.0.18\webapps\ROOT\html\portal
- Now, terms_of_use.jsp has been overriden with your file
- The old file has been renamed as terms_of_use.portal.jsp
- Why?
 - When you undeploy your hook, the original file will be restored
 - You can include the original file from yours

OVERRIDE TERMS OF USE

1. Add the following to your terms_of_use.jsp:

```
<liferay-util:include page="/html/portal/terms_of_use.portal.jsp" />
```

- This will include the original file in yours, which is really good to minimize upgrade problems when customizing a JSP

LIMITATIONS

- Limits
 - Order is not guaranteed
 - Use multiple hooks with caution
- Road map
 - More portal.properties keys
 - Struts actions

CLASS LOADING

- EXT Plugin allows native access to portal class loader
- Other Plugins can also access the portal class loader via PortalClassInvoker
- Plugins can access other plugins via PortletClassInvoker
- ServiceBuilder simplifies class loading issues
- MessageBus bypasses class loading issues

Notes:



EXT PLUGIN OVERVIEW

Copyright © 2000 – 2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

OVERVIEW

- ❖ EXT plugins were added to Liferay in version 6.
- ❖ Use EXT plugins to modify core functionality.

- ❖ EXT plugins (<10MB) are a light weight version of the EXT Environment (>100MB).
- ❖ EXT plugins support the same functionality as the EXT Environment.

- ❖ Do *NOT* use EXT plugins to create *new* portlets or services. Create new portlets and services in portlet plugins.
- ❖ If you are migrating from an old version of Liferay, use this release to migrate your portlets to plugins, as they will no longer be supported in Ext in future releases.

WHEN TO USE EXT PLUGINS (1)

In some circumstances where advanced customizations are desired, EXT plugins provide some additional capabilities:

- ❖ Overriding portal.properties not supported by Hook plugins, such as adding or removing sections from the User profile

```
#  
# Input a list of sections that will be included as part of the user form  
# when updating a user.  
#  
users.form.update.main=details,password,organizations,communities,user-  
groups,roles,pages,categorization  
users.form.update.identification=addresses,phone-numbers,additional-email-  
addresses,websites,instant-messenger,social-network,sms,open-id  
users.form.update.miscellaneous=announcements,display-  
settings,comments,custom-fields
```

WHEN TO USE EXT PLUGINS (2)

- ❖ Changing specific characteristics of Liferay's core portlets, such as whether it appears in the control panel and in what order
liferay-portlet-ext.xml
- ❖ Changing the init-params or default portlet preferences of core portlets.
 - Example: changing the default feeds for the RSS portlet
portlet-ext.xml
- ❖ Heavy customizations of core portlets, such as changing the mappings of paths to custom classes, adding new mappings or changing forwards to go to custom JSPs
struts-config-ext.xml, *tiles-defs-ext.xml*
- ❖ Custom servlets or filters: *web.xml*

WHEN TO USE EXT PLUGINS (3)

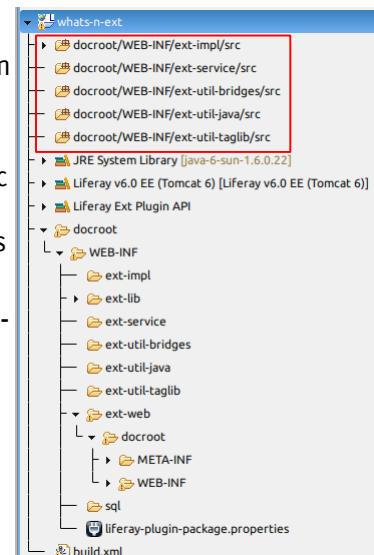
- Providing custom implementations for beans declared in Liferay's spring XML files (better: use Service Wrappers from a Hook plugin to override Liferay services)
- Adding JSPs referenced in portal.properties (for example, if you override the following property in portal-ext.properties, add the custom JSP in the EXT plugin for maintainability)

```
#  
# Settings for panel layouts.  
#  
layout.edit.page[panel]=/portal/layout/edit/panel.jsp
```

- Overwriting Liferay classes (not recommended except for advanced cases)
 - Limitations: Must take into account the class loading order of the operating system, JVM system and application server

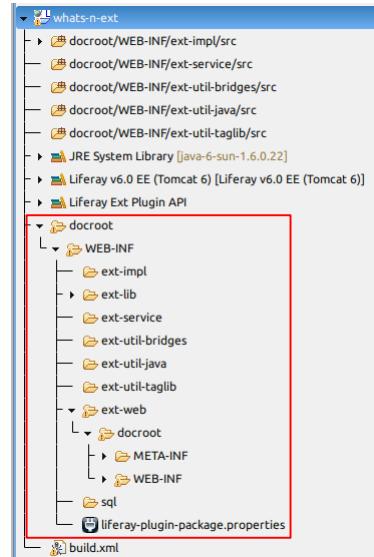
STRUCTURE OF EXT PLUGINS (1)

- **ext-impl/src:** mirrors portal-impl/src from Liferay's source tree. Anything you'd want to override from there goes here.
- **ext-service/src:** mirrors portal-service/src from Liferay's source tree. Place classes that should be available to other plugins here.
- **ext-util-bridges, ext-util-java and ext-util-taglib** mirror the corresponding folders from Liferay's source tree. Use them to customize util-bridges.jar, util-java.jar and util-taglib.jar respectively. In most scenarios you can ignore these directories.



STRUCTURE OF EXT PLUGINS (2)

- ❖ **ext-web/docroot:** contains descriptors that extend Liferay's:
 - struts-config-ext.xml and tiles-defs-ext.xml (for customizing Struts actions and tiles)
 - portlet-ext.xml and liferay-portlet-ext.xml (for modifying portlet parameters)
 - You can also place any JSPs needed by your customizations here.



WWW.LIFERAY.COM

LIFERAY.

STRUCTURE OF EXT PLUGINS (3)

- ❖ **ext-lib/global:** .jars placed here get copied to the global classpath.
- ❖ **ext-lib/portal:** .jars placed here get copied to the Liferay application folder, for use within the portal classloader.
- ❖ By default, several files are added to the plugin. Here are the most significant ones:
 - docroot/WEB-INF/ext-impl/src/**portal-ext.properties**
 - Inside docroot/WEB-INF/ext-web/docroot/WEB-INF
 - portlet-ext.xml
 - liferay-portlet-ext.xml
 - struts-config-ext.xml
 - tiles-defs-ext.xml

WWW.LIFERAY.COM

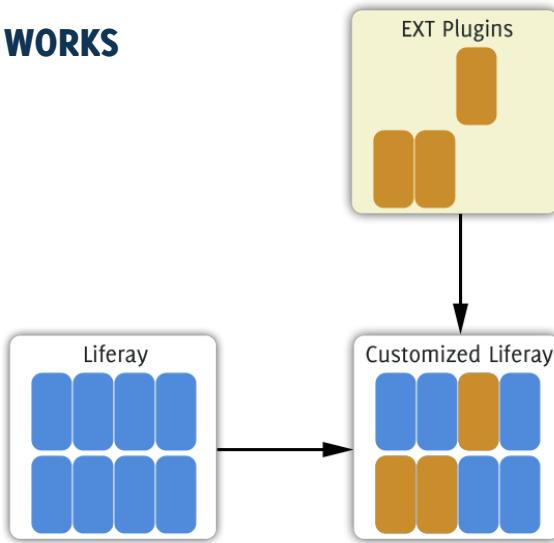
LIFERAY.

STRUCTURE OF EXT PLUGINS (4)

- ❖ Best Practice: if you're not customizing a file, remove it from the generated Ext plugin.
 - Do not delete the folders because they're required by the ant build scripts.
 - Liferay keeps track of the files deployed by each EXT plugin, and it won't allow deploying two EXT plugins if they override the same file to avoid collisions.
 - If necessary, the original files can be copied again from the ext template directory:

liferay-plugins-sdk-6.0-ee-sp1\tools\ext_tmpl

HOW IT WORKS



- ❖ Note: Files such as portal-ext.properties, struts-config-ext.xml, custom JSPs and custom libraries are deployed alongside the default Liferay files.

EXT PLUGIN LIMITATIONS

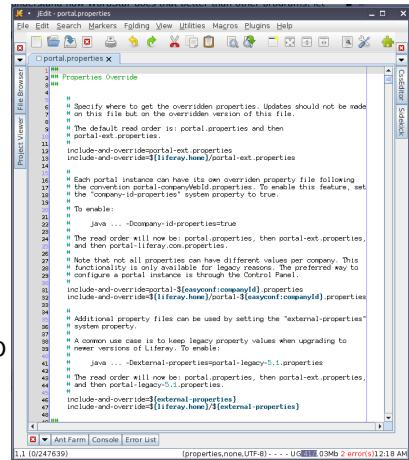
- ❖ Unlike other plugins, EXT plugins are not hot deployable and require a server restart.
- ❖ Redeploying an EXT plugin is more complicated than redeploying other types of plugins.

PORTAL.PROPERTIES

- ❖ Liferay Portal is highly configurable and extensible.
- ❖ Default configuration settings are defined in the source tree in this file:
`portal-impl/src/portal.properties`
 - ❖ Many of the most common configuration changes involve this file.
 - ❖ This file is read-only and must not be edited.
 - ❖ This file eventually finds its way into the `portal-impl.jar` archive when the portal is packaged for deployment to an application server.

EXERCISE: PORTAL.PROPERTIES

- ▶ Open the *portal.properties* file which has been provided to you in your course materials. This file was extracted from the Liferay EE source.
 - ▶ The file is well documented and in addition to the comments on the available properties, the file lists common alternate settings.
 - ▶ These alternate settings are commented out, but can be copied to the *portal-ext.properties* file.
 - ▶ Take a moment to examine some of the default configuration settings in the file.



WWW.LIFERAY.COM

LIFERAY

PORTAL-EXT.PROPERTIES

- The EXT plugin provides the ability to *override* values in the portal.properties file.
 - Unlike a Hook plugin, an EXT Plugin can override any property in this file.
 - Custom values should be placed in this file:
`ext-impl/src/portal-ext.properties`
 - This file eventually finds its way into the portal's WEB-INF\classes folder when the portal is deployed to an application server.

WWW.LIFERAY.COM

LIFERAY

PORTAL.PROPERTIES RUNTIME VALUES

- The final values of portal.properties and portal-ext.properties can be viewed in the *Server Administration* portlet in the Control Panel.

The screenshot shows the Liferay Server Administration interface. The top navigation bar includes links for Resources, Log Levels, Properties (which is the active tab), Captcha, Data Migration, File Uploads, Mail, OpenOffice, and Script. Below the navigation is a status bar indicating the version (Liferay Portal Enterprise Edition 6.0 EE SP1) and uptime (01:38:05). The main content area is titled "Properties" and displays a table of system properties. The table has two columns: "Property" and "Value". The visible rows are:

Property	Value
admin.default.group.names	
admin.default.role.names	Power User User
admin.default.user.group.names	
admin.email.from.address	test@liferay.com
admin.email.from.name	Joe Bloos

At the bottom of the page, there are links for WWW.LIFERAY.COM and the LIFERAY logo.

Notes:



EXTENDING USER MANAGEMENT

Copyright © 2000 - 2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

GOALS

- ❖ Show how Liferay's user management UI can be customized.
- ❖ Learn how to:
 - Show or hide sections of the user profile
 - Add custom fields through the UI
 - No database modification necessary!
 - Create a custom user profile section

WHAT IS A USER PROFILE?

- ❖ Go to the Control Panel and click the *Users* link on the left.
- ❖ Create a user.

The screenshot shows the Liferay Control Panel's User Information screen. The user profile for "Joe Schmoe" is displayed. The "Details" section includes fields for Title (Mr.), Screen Name (joschmoe), Email Address (joschmoe@place.com), First Name (Joe), Middle Name (Schmoe), Last Name (Schmoe), Suffix (Dr.), User ID (10482), Birthday (January 1, 1970), Gender (Male), and Job Title (empty). On the right, a sidebar lists sections like User Information, Identification, and Miscellaneous. The "User Information" section is currently selected. At the bottom right of the form are "Save" and "Cancel" buttons.

WWW.LIFERAY.COM

LIFERAY.

EXERCISE: CREATE EXT PLUGIN PROJECT

1. Go to File > New > Liferay Project
2. Project name: *extending-user-management*
3. Display name: *Extending User Management*
4. Liferay Plug-ins SDK: *liferay-plugins-sdk-6.0-ee-sp1*
5. Liferay Portal Runtime: *Liferay v6.0 EE (Tomcat 6)*
6. Plug-in Type: *Ext*
- ✓ Click Finish

The screenshot shows the "Liferay Plug-in Project" configuration step of the wizard. It includes fields for "Project name" (extending-user-management), "Display name" (Extending User Management), and "Configuration" settings for "Liferay Plug-ins SDK" (liferay-plugins-sdk-6.0-ee-sp1) and "Liferay Portal Runtime" (Liferay v6.0 EE (Tomcat 6)). Other options like "Advanced project configuration..." and "Create a new project from existing sources..." are also visible. The "Plug-in Type" section shows "Ext" selected. The "Working sets" section has an unchecked checkbox for "Add project to working sets". At the bottom are "Finish" and "Cancel" buttons.

WWW.LIFERAY.COM

EXERCISE: REMOVE FILES FROM EXT PLUGIN

- ✓ Best Practice: Remove files from the EXT plugin that we won't use (this will help us avoid conflicts with other EXT plugins)

Remove all of the files in the following directory:

docroot/WEB-INF/ext-web/docroot/WEB-INF

- portlet-ext.xml
- liferay-portlet-ext.xml
- struts-config-ext.xml
- tiles-defs-ext.xml
- web.xml

CUSTOMIZE USER FORM SECTIONS

- ❖ portal.properties has a set of properties that determine the sections of the user profile
- ❖ Sections that are included in the user form when adding a user:

```
users.form.add.main=details,organizations  
users.form.add.identification=  
users.form.add.miscellaneous=
```

- ❖ Sections that are included in the user form when updating a user:

```
users.form.update.main=details,password,organizations,communities,user-groups,roles,categorization  
users.form.update.identification=addresses,phone-numbers,additional-email-addresses,websites,instant-messenger,social-network,sms,open-id  
users.form.update.miscellaneous=announcements,display-settings,comments,custom-fields
```

EXERCISE: CUSTOMIZE USER FORM SECTIONS

1. **Add** the contents of *01-portal-ext.properties* from the snippets to the bottom of portal-ext.properties.

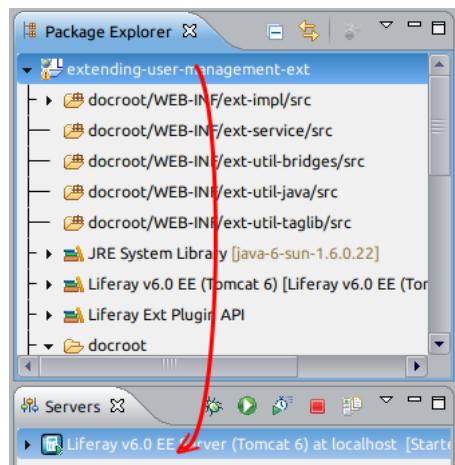
```
users.form.update.main=details,password,organizations,communities,user-groups,roles,categorization  
users.form.update.identification=addresses,phone-numbers,additional-email-addresses,websites,instant-messenger,social-network,sms,open-id  
users.form.update.miscellaneous=announcements,display-settings,comments,custom-fields
```

2. All of the sections from *users.form.update.identification* should be empty:

```
users.form.update.main=details,password,organizations,communities,user-groups,roles,categorization  
users.form.update.identification=  
users.form.update.miscellaneous=announcements,display-settings,comments,custom-fields
```

EXERCISE: DEPLOY (PUBLISH) THE EXT PLUGIN

1. **Drag** the *extending-user-management-ext* project onto the "Liferay v6.0 EE Server (Tomcat 6) at localhost" Server
2. If the server is running, Ext will be deployed and the server will restart.
3. If the server is not running, start it.



USER FORM SECTION UPDATED

- ❖ Go to the Control Panel → Users and click on a user.
- ❖ The Addresses, Phone Numbers, Additional Email Addresses, Websites, Instant Messenger, Social Network, SMS and Open ID sections should no longer display.
- ❖ Liferay also removed the *Identification* sub-heading because all of the sections were removed.



WWW.LIFERAY.COM

LIFERAY.

EXT PLUGIN DEPLOYMENT

- ❖ Ext plugins get deployed directly to the running Liferay instance instead of as their own applications.
- ❖ This means a server restart is required every time you deploy an Ext plugin, but Liferay Developer Studio automates this for you.
- ❖ Publishing an Ext plugin the first time is as easy as drag/drop.
- ❖ To re-publish, right-click on the server runtime and select *Publish*.
- ❖ The rest of your plugins should auto-deploy normally.

WWW.LIFERAY.COM

LIFERAY.

ADDING CUSTOM FIELDS

- ❖ Next, we'll add custom fields to the User profile.
- ❖ Since we already have a Library portlet, let's let users store their favorite books on their profiles.

Joe Bloggs

Library Id
I don't have any favorite books

Favorite Books

User Information

- Details
- Password
- Organizations
- Communities
- User Groups
- Roles
- Pages
- Categorization

Miscellaneous

- Announcements
- Display Settings
- Comments

Save Cancel

WWW.LIFERAY.COM

LIFERAY.

EXERCISE: ADD CUSTOM FIELDS (1)

- ❖ Go to Control Panel → Portal → Custom Fields and Click User

Control Panel

Control Panel > Portal > Custom Fields

Bruno Admin

liferay.com

Portal

- Users
- Organizations
- Communities
- User Groups
- Roles
- Password Policies
- Settings
- Custom Fields

Custom Fields

- Monitoring
- Plugins Configuration
- Page Templates
- Site Templates
- Workflow
- OpenSocial
- WSRP

Server

- Server Administration
- Portal Instances
- Plugins Installation
- Update Manager

Custom Fields

Resource	Custom Fields
Wiki Page	Edit
Message Boards Category	Edit
Message Boards Message	Edit
Calendar Event	Edit
Page	Edit
Organization	Edit
User	Edit
Web Content	Edit
Document Library Document	Edit
Document Library Folder	Edit
Bookmarks Entry	Edit
Bookmarks Folder	Edit
Image Gallery Image	Edit
Image Gallery Folder	Edit
Blogs Entry	Edit

WWW.LIFERAY.COM

LIFERAY.

EXERCISE: ADD CUSTOM FIELDS (2)

1. **Click** the “Add Custom Field” button to create two new attributes with the following properties:

- Key: library-id
- Type: Text field – indexed

- Key: favorite-books
- Type: Text box – indexed

2. **Edit** favorite-books and **Enter** in the default value field: “I don't have any favorite books.”

WWW.LIFERAY.COM

 LIFERAY.

EXERCISE: ADD CUSTOM FIELDS (3)

1. **Go** to *Portal* → *Users* to see a list of users in the system.
2. **Edit** one of the existing users
3. On the right-hand side navigation, **Click** *Custom Fields*
✓ Checkpoint:
 - You should see one text field labeled Library ID and one text box labeled Favorite Books

Custom Fields

Favorite Books

I don't have any favorite books.

Library Id

WWW.LIFERAY.COM

 LIFERAY.

EXERCISE: ADD A SECTION TO THE USER PROFILE

1. Go back to the *portal-ext.properties* file that you have open in Developer Studio.

2. Add a new value called *library* to *users.form.update.main*

```
users.form.update.main=details,password,organizations,communities,user-groups,roles,categorization,library
```

3. Remove the value *custom-fields* from *users.form.update.miscellaneous*

```
users.form.update.miscellaneous=announcements,display-settings,comments
```

EXERCISE: CREATE THE JSP FOR THE LIBRARY SECTION

1. Navigate to *ext-web\docroot* in Developer Studio, right click on the docroot folder and click *New → Folder*.
2. Enter “*html/portlet/enterprise_admin/user/*” as the folder name.
3. Create the file *library.jsp* in the path you just created
4. Add the contents of *o2-library.jsp* from the snippets in the *library.jsp* file you just created.

EXERCISE: ADD LANGUAGE PROPERTIES

1. **Create** a new file called *Language-ext.properties* in the following path

ext-impl\src\content\Language-ext.properties

2. **Add** the contents of o3-Language-ext.properties from the snippets in Language-ext.properties

```
library=Library  
library-id=Library Identifier
```

ADD A NEW PROFILE SECTION (1)

- All JSPs inside the folder enterprise_admin/user are prepared to be overridden or replaced
- The following request attributes are available to them:
 - **user.selUser**: The user being edited (User)
 - **user.selContact**: The contact details of the user.
 - **user.passwordPolicy**: The password policy of the user (PasswordPolicy)
 - **user.groups**: The communities of the user (List<Group>)
 - **user.organizations**: The organizations of the user (List<Organization>)
 - **user.roles**: The roles of the user (List<Role>)
 - **user.communityRoles**: The community roles of the user (List<UserGroupRole>)
 - **user.organizationRoles**: The organization roles of the user (List<UserGroupRole>)
 - **user.userGroups**: The user groups of the user (List<UserGroup>)

ADD A NEW PROFILE SECTION (2)

- The liferay-ui:custom-attribute JSP tag shows a form field for a custom attribute.
- The main attributes of this tag are:
 - **className**: identifies the type of object which will hold the custom attribute.
For user management the value will always be "com.liferay.portal.model.User"
 - **classPK**: the identifier of the user or 0 (zero) if the user has not been created yet
 - **editable**: if true, an HTML form field will be rendered, otherwise the field value will be shown.
 - **label**: if true, a form field label will be generated using Liferay Portal's convention. Otherwise only the field itself will be shown.
 - **name**: The name of the custom attribute provided when it was created

WWW.LIFERAY.COM

LIFERAY.

EXERCISE: USER FORM UPDATED WITH LIBRARY SECTION

- Deploy / Publish the EXT plugin
- *Library* is now available under the *User Information* section, and *Custom Fields* was removed from the *Miscellaneous* section
- Clicking on the Library section displays the content of library.jsp

Joe Bloggs

Library Id
I don't have any favorite books

Favorite Books



WWW.LIFERAY.COM

LIFERAY.

COMBINING EXT WITH HOOKS

- ❖ A best practice is to keep Ext as simple as possible.
- ❖ Only use Ext when no other plugin type will do the job.
- ❖ You can even combine hooks and Ext to together provide a feature you need.
- ❖ This will be the subject of the next exercise.

ADDING CUSTOM FIELDS PROGRAMMATICALLY

- ❖ As we've seen, Custom Fields can be added through the user interface which can be useful for initial testing and set up.
- ❖ Custom Fields can also be added programmatically using the Expando API (useful when migrating Custom Fields from one environment to another).
- ❖ The easiest way to do this is through a custom StartupAction implemented with a Hook Plugin.
- ❖ Note: Custom Fields are called Expandos in the back end API.

GET/ADD DEFAULT EXPANDO TABLE

- ❖ First, we get the default Expando table for the User class if it exists, and add the default Expando table if it does not exist
- ❖ Use `getDefaultTable` and `addDefaultTable` when working with objects that have Custom Fields in the UI, such as the User and Organization object (use `getTable` and `addTable` for custom Expando tables)

```
ExpandoTable table = null;
try {
    table = ExpandoTableLocalServiceUtil.getDefaultTable(
        companyId, User.class.getName());
}
catch(NoSuchTableException nste) {
    table = ExpandoTableLocalServiceUtil.addDefaultTable(
        companyId, User.class.getName());
}
```

WWW.LIFERAY.COM



ADD EXPANDO COLUMN

- ❖ Since we want to programmatically add Expando columns to our environment on startup, we'll add the *library-id* column first
- ❖ If the column already exists, we'll get the existing column instead.
- ❖ How would we add an *employeeld* column instead?

```
ExpandoColumn column = null;
try {
    column = ExpandoColumnLocalServiceUtil.addColumn(
        tableId, "library-id", ExpandoColumnConstants.STRING);
    ...
}
catch(NoSuchColumnException nsce) {
    column = ExpandoColumnLocalServiceUtil.getColumn(
        tableId, "library-id");
```

WWW.LIFERAY.COM



ADD EXPANDO COLUMN PROPERTIES

- ❖ Here, we make our *library-id* column searchable by adding the INDEXABLE property

```
column = ExpandoColumnLocalServiceUtil.addColumn(
    tableId, "library-id", ExpandoColumnConstants.STRING);

UnicodeProperties properties = new UnicodeProperties();

properties.setProperty(
    ExpandoBridgeIndexer.INDEXABLE, Boolean.TRUE.toString());

column.setTypeSettingsProperties(properties);

ExpandoColumnLocalServiceUtil.updateExpandoColumn(column);
```

WWW.LIFERAY.COM



ADD/UPDATE EXPANDO VALUES (1)

- ❖ What if we wanted to programmatically update the data in the *library-id* column for a specific user?
- ❖ Use *ExpandoValueLocalServiceUtil.addValue* to add or update the value of the *library-id* Expando Column for the specific user.

```
String emailAddress = "test@liferay.com";
User user = UserLocalServiceUtil.getUserByEmailAddress(
    companyId, emailAddress);
long classNameId = table.getClassNameId();
long columnId = column.getColumnIndex();
long classPK = user.getUserId();
String data = "777";
ExpandoValue value = ExpandoValueLocalServiceUtil.addValue(
    classNameId, tableId, columnId, classPK, data);
```

WWW.LIFERAY.COM



ADD/UPDATE EXPANDO VALUES (2)

```
ExpandoValue value = ExpandoValueLocalServiceUtil.addValue(  
    classNameId, tableId, columnId, classPK, data);
```

- **classNameId:** Map the value of User.class.getName() to an ID (the class name is mapped to an ID for performance, query based on an ID vs. a String)
- **tableId:** tableId of the default Expando Table for the User class
- **columnId:** the columnId of the "library-id" Expando Column
- **classPK:** Primary key of the class we want to perform an update on (update the data for the user test@liferay.com, classPK = user.getUserId())
- **data:** Data / value to store in the *library-id* column

EXERCISE: CREATE HOOK PROJECT (1)

1. [Go](#) to File > New > Liferay Project
 2. Project name:
expando-startup-action
 3. Display name:
Expando Startup Action
 4. Liferay Plug-ins SDK:
liferay-plugins-sdk-6.0-ee-sp1
 5. Liferay Portal Runtime:
Liferay v6.0 EE (Tomcat 6)
 6. Plug-in Type:
Hook
- ✓ [Click Finish](#)

EXERCISE: CREATE HOOK PROJECT (2)

1. [Go](#) to File > New > Liferay Hook
 2. [Check](#) Portal properties
 3. [Click](#) Next
 4. [Click](#) the Add... button next to the *Define actions to be executed on portal events* text box
 5. [Select](#) application.startup.events
- ✓ [Click](#) OK

EXERCISE: CREATE HOOK PROJECT (3)

1. [Click](#) New... next to Class:
 2. Classname:
ExpandoStartupAction
 3. Java Package:
com.liferay.training.action
 4. Superclass:
com.liferay.portal.kernel.events.SimpleAction
 5. [Click](#) Create
 6. [Click](#) Ok
- ✓ [Click](#) Finish

EXERCISE: CREATE EXPANDO STARTUP ACTION

1. Open *ExpandoStartupAction.java*

expando-startup-action-hook\docroot\WEB-INF\src\com\liferay\training\action\ExpandoStartupAction.java

2. Delete the contents of the file
3. Add the contents of *05-Expando-Startup-Action* from the snippets in *ExpandoStartupAction.java*
4. Drag the *expando-startup-action-hook* project onto the "Liferay v6.0 EE Server (Tomcat 6) at localhost" Server

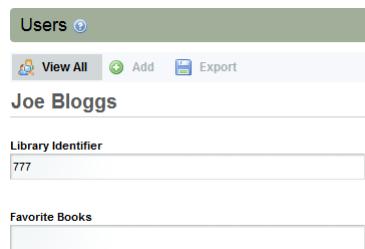
LIBRARY ID UPDATED PROGRAMMATICALLY

- After the Hook is deployed, the value of the *library-id* Expando Column is displayed in the console

```
10:05:11,253 INFO [HookHotDeployListener:406] Registering hook for expando-startup-action-hook
Loading file:/C:/liferay-developer-studio/liferay-portal-6.0-ee-spl/tomcat7/lib/ext/library_id_777.jar
Library ID: 777
10:05:11,347 INFO [HookHotDeployListener:662] Hook for expando-startup-action-hook registered
Jan 8, 2011 10:05:11 AM org.apache.catalina.core.StandardContext reload
INFO: Reloading this Context has started
10:05:11,353 INFO [PluginPackageUtil:1080] Reading plugin package for com.liferay.training.expando.startup
10:05:11,354 INFO [HookHotDeployListener:758] Hook for expando-startup-action-hook registered
10:05:11,396 INFO [PluginPackageUtil:1080] Reading plugin package for com.liferay.training.expando.startup
10:05:11,445 INFO [HookHotDeployListener:406] Registering hook for expando-startup-action-hook
Loading file:/C:/liferay-developer-studio/liferay-portal-6.0-ee-spl/tomcat7/lib/ext/library_id_777.jar
Library ID: 777
10:05:11,508 INFO [HookHotDeployListener:662] Hook for expando-startup-action-hook registered
```

LIBRARY ID UPDATED PROGRAMMATICALLY (2)

1. [Go](#) to Control Panel > Portal > Users
2. [Search](#) for *Joe Bloggs*
(*test@liferay.com*)
3. [Edit](#) *Joe Bloggs*
(*Actions button > Edit*)
4. [Click](#) the *Library* section on the right-hand navigation menu
 - ✓ The *Library Identifier* text box should now display 777





EXTENDING SELF-REGISTRATION

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

GOALS

- Our goal is to trigger a new action on the Create Account page.
- To do this, we will learn how EXT Plugins can be used to override Struts Actions.
- To start, we need to learn a little bit more about the portlet we are trying to modify.

WHICH PORTLET TO MODIFY?

1. Start Liferay
2. Click on the *Create Account* link in the *Sign In* portlet (You must be signed out to see the link).
3. Examine the URL

```
http://127.0.0.1:8080/web/guest/home?  
p_auth=P84GQZXy&p_p_id=58&p_p.lifecycle=1  
&p_p_state=maximized&p_p_mode=view  
&p_p_col_id=column-2&p_p_col_count=2  
&saveLastPath=0&_58_struts_action=/login/create_account
```

- `p_p_id=58` is the portlet name
- `p_p.lifecycle=1` indicates a struts action
- `&_58_struts_action=%2Flogin%2Fcreate_account` is the struts mapping `/login/create_account` defined in `struts-config.xml`

FIND THE PORTLET

- Liferay core portlets are defined in a portlet deployment descriptor called `portlet-custom.xml`, which can be found in the `liferay-developer-studio\liferay-portal-6.0-ee-spi\tomcat-6.0.29\webapps\ROOT\WEB-INF` directory

```
<portlet>  
  <portlet-name>58</portlet-name>  
  <display-name>Login</display-name>  
  <portlet-class>com.liferay.portlet.StrutsPortlet</portlet-class>  
  <init-param>  
    <name>view-action</name>  
    <value>/login/view</value>  
  </init-param>  
  <expiration-cache>0</expiration-cache>  
  <supports>  
    <mime-type>text/html</mime-type>  
  </supports>  
  ...  
</portlet>
```

FIND THE STRUTS ACTION

- The Liferay core portlets' Struts Actions are defined in a file called *struts-config.xml* which can also be found in the *liferay-developer-studio\liferay-portal-6.0-ee-spi\tomcat-6.0.29\webapps\ROOT\WEB-INF* directory

```
<action path="/login/create_account"
       type="com.liferay.portlet.login.action.CreateAccountAction">
    ...
</action>
```

- We will override this value to point to our custom *AddUserAction* class.

```
<action path="/login/create_account"
       type="com.ext.portlet.login.action.AddUserAction">
    ...
</action>
```

EXERCISE: CREATE EXT PLUGIN PROJECT

- [Go](#) to File > New > Liferay Project
- Project name:
add-user-action
- Display name:
Add User Action
- Liferay Plug-ins SDK:
liferay-plugins-sdk-6.0-ee-spi
- Liferay Portal Runtime:
Liferay v6.0 EE (Tomcat 6)
- Plug-in Type:
Ext
- [Click](#) Finish

EXERCISE: REMOVE FILES FROM EXT PLUGIN

1. Best Practice: Remove files from the EXT plugin that we won't use (this will help us avoid conflicts with other EXT plugins)

[Remove](#) the files in the following directory:

docroot/WEB-INF/ext-web/docroot/WEB-INF

- » portlet-ext.xml
- » liferay-portlet-ext.xml
- » tiles-defs-ext.xml
- » web.xml

EXERCISE: CREATE CUSTOM STRUTS ACTION CLASS

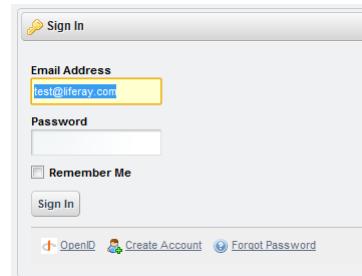
1. In Liferay Developer Studio, [Right Click](#) on the docroot/WEB-INF/ext-impl/src source folder and [Select](#) New → Class
 - » Package: *com.ext.portlet.login.action*
 - » Name: *AddUserAction*
2. [Replace](#) the contents of AddUserAction with the contents of 06-AddUserAction from the snippets

EXERCISE: OVERRIDE STRUTS ACTION MAPPING

1. Open *struts-config-ext.xml* in the *add-user-action-ext* project
add-user-action-ext\docroot\WEB-INF\ext-web\docroot\WEB-INF\struts-config-ext.xml
2. Replace the contents of *struts-config-ext.xml* with the contents of *07-struts-config-ext.xml* from the snippets

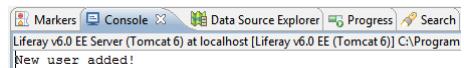
EXERCISE: DEPLOY (PUBLISH) THE EXT PLUGIN

1. Drag the *add-user-action-ext* project onto the "Liferay v6.0 EE Server (Tomcat 6) at localhost" Server
2. Right click the "Liferay v6.0 EE Server (Tomcat 6) at localhost" server, and click *Publish*
3. Open your browser and go to <http://localhost:8080>
4. Sign Out if you're signed in
5. Sign In (Log In) at the upper right-hand corner of your browser
6. Click on *Create Account*



EXERCISE: DEPLOY (PUBLISH) THE EXT PLUGIN

1. [Create](#) a new user account
- ✓ The message *New User added!* is displayed in the Console.



Notes:



EXT BEST PRACTICES

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

EXTEND, DON'T MODIFY

- As shown in the previous example, it is far better to extend a Liferay class with your functionality than it is to modify the class.
- It may be tempting to paste in Liferay's code and then make your modifications.
- This will break on upgrades—sometimes even point release upgrades.
- Extend Liferay's class, add your functionality, and then call super().
- This creates much more maintainable code.

PORLETS ARE DEPRECATED IN 6.0

- In the past, you could create portlets in the Extension Environment.
- If you are upgrading from an older release of Liferay, they will still work in Ext plugins, but they are deprecated.
- Take the time during the release cycle of 6.0 to migrate your portlets out of Ext and into plugins.

WWW.LIFERAY.COM



SERVICE BUILDER IS DEPRECATED IN 6.0

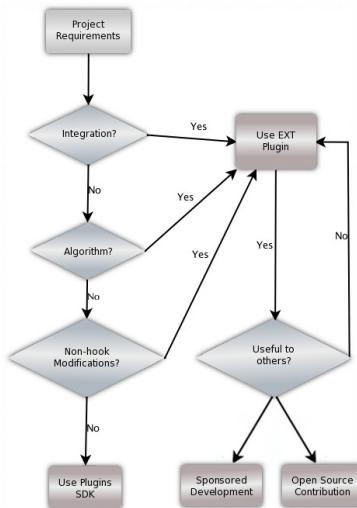
- In the past, you could create services in the Extension Environment.
- If you are upgrading from an older release of Liferay, this will still work in the Ext plugin, but it is deprecated.
- Take the time during the release cycle of 6.0 to migrate your services to plugins.
- Services created in plugins are packaged in a .jar file which is copied to the plugin's WEB-INF/lib folder.
- If you move this .jar out to the server's global classpath, that service will be available to all plugins, including Ext plugins.

WWW.LIFERAY.COM



DO I REALLY NEED EXT?

- Does my project require integration with other products that Liferay does not yet support?
- Is Liferay's algorithm for some functionality inappropriate for my use case?
- Do I need to change a bundled portlet in a way not supported by hooks?
- Would my changes be a useful addition to the product?
- **YES to any of these questions means you have a good reason to use Ext!**



MULTIPLE EXT PLUGINS

- As we have demonstrated, you can have multiple Ext plugins deployed.
- Remember: Ext plugins are deployed into the installed instance of Liferay.
- You will therefore need to keep track of what you have customized in order to maintain this code.
- Though it's possible to deploy multiple Ext plugins, the more you have, the less easy it is to know which plugin has customized what functionality.
- For this reason, try to keep the number of Ext plugins deployed to a minimum, or divide customizations into easily defined categories.
- Example: shopping-customization-ext, wiki-customization-ext, login-customization-ext.

MANUALLY DEPLOYING EXT PLUGINS

- Unlike other plugins, EXT plugins are not truly hot deployable.
- During development, you can manually execute the `deploy` target in the Ant window or, if you're using Liferay Developer Studio, you can take advantage of its deployment process, as we have.
- To deploy to production, you'll want to use the `war` target of the Ant script, and then deploy the resulting .war file to your production server.
- Watch the console; you should see a message that confirms the plugin is deployed and prompting you to restart the server.

Extension environment for extending-user-management-ext has been applied. You must reboot the server and redeploy all other plugins.

- Redeploying all other plugins is not strictly mandatory, but you should do it if some changes applied through the Ext plugin may affect the deployment process itself.

DEPLOY VS DIRECT-DEPLOY

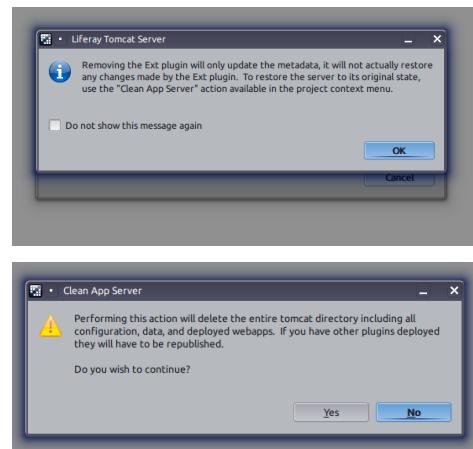
- The `deploy` target is the standard deployment where the Plugins SDK packages a .war file and deploys it to the server's hot deploy directory.
- Liferay Developer Studio uses the `direct-deploy` target which deploys the files directly to the Liferay instance in the server instead of packaging it as a .war file.
- Note that `direct-deploy` does not work for servers like WebSphere and WebLogic and is not an option for a producti. n environment

REDEPLOYING EXT PLUGINS

- Redeploying an EXT Plugin is a simple matter of dragging and dropping the project on to your runtime again.
- In production, as long as you don't need to *remove* something from your customized Liferay installation, you can redeploy an Ext plugin.
- If, however, something goes wrong with your customization, or you need to remove a part of it, you'll need to undeploy Ext.
- This cannot be done, except by reverting back to a clean Liferay installation.
- If you are using a Liferay bundle, this can be done by unzipping the bundle and pointing it to your database.
- If you are using a Liferay .war file, you will need to redeploy the original Liferay .war file.
- In development, with Liferay Developer Studio, this process is automated (see next slide).

EXT PLUGIN UNDEPLOYMENT

- To remove an Ext plugin, right-click it in the runtime and click *Remove*.
- After the message (see top right), right-click the Ext plugin project and select Liferay → Clean App Server.
- Clicking *Yes* to the next message removes your Liferay runtime and installs a clean one in its place.



DEPLOYING IN PRODUCTION METHOD 1

- ❖ This method can be used in any application server that supports auto deploy, such as Tomcat or JBoss. Its main benefit is that the only artifact that needs to be transferred to the production system is the .war file which the Ext plugin produced using the ant war target, which is usually a small file. Here are the steps that need to be executed on the server:
 - Redeploy Liferay. To do this, follow the same steps you used when first deploying Liferay on the app server. If you are using a bundle, you can just unzip the bundle again. If you've installed Liferay manually on an existing application server, you'll need to redeploy the .war file and copy the global libraries to the appropriate directory within the application server. If this is the first time the Ext plugin is deployed, you can skip this step.
 - Copy the Ext plugin .war into the auto deploy directory. In a bundle, this directory is in the root of the unzipped bundle called deploy.
 - Once the Ext plugin is detected and deployed by Liferay, restart the Liferay server.
<http://www.liferay.com/documentation/liferay-portal/6.0/development/-/ai/deploying-in-production>

DEPLOYING IN PRODUCTION METHOD 2

- ❖ Method 2: Generate an aggregated WAR file
- ❖ This method can be used for application servers that do not support auto deploy, such as WebSphere or Weblogic. Its main benefit is that all Ext plugins are merged before deployment to production, so a single .war file will contain Liferay plus the changes from one or more Ext plugins. Before deploying the .war file, you'll need to copy the dependency .jars for both Liferay and the Ext plugin to the global application server class loader in the production server. This location varies from server to server; please see the Liferay Portal Administrator's Guide for further details for your application server.
- ❖ To create the aggregated .war file, deploy the Ext plugin first to the Liferay bundle you're using in your development environment. Once it's deployed, create a .war file by zipping the webapps/ROOT folder of Tomcat. Also, copy all the libraries from the lib/ext directory of Tomcat that are associated to all the Ext plugins to your application server's global classpath, as noted above. These steps will be automated with Ant targets in the next version of Liferay, but for now, they need to be done manually.
- ❖ Once you have the aggregated .war file follow these steps on the server:
 - Redeploy Liferay using the aggregated WAR file.
 - Stop the server and copy the new version of the global libraries to the appropriate directory in the application server.

MIGRATING FROM THE EXT ENVIRONMENT (1)

- Run the following target from the *ext* directory in the Plugins SDK to upgrade the EXT Environment:

```
ant upgrade-ext -Dext.dir=C:/projects/liferay/ext -Dext.name=my-ext -Dext.display.name="My Ext"
```

- *ext.dir* is a command line argument that points to the location of the old Extension Environment.
- *ext.name* is the name of the Ext plugin that you want to create.
- *ext.display.name* is the display name

MIGRATING FROM THE EXT ENVIRONMENT (2)

- Review and update any Liferay's APIs that you used in your code.
- Review any Liferay core JSPs that you modified and merge in your changes with the new version of the core JSPs.
- Run *ant build-service* for services you've created with Service Builder.
- Recommendation: Move custom services to portlet plugins because the use of Service Builder in EXT plugins is deprecated and is not guaranteed to be in future releases.
- Recommendation: Move custom portlets to portlet plugins as support for custom portlets in EXT plugins may not be supported in future releases (portlet plugins are easier to maintain and can be deployed without a server restart)

Notes:



LIFERAY WEB SERVICES

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

WEB SERVICES

- ❖ Web Services are resources which may be called over the HTTP protocol to return data.
- ❖ They are platform-independent, allowing communication between applications on different operating systems and application servers.

LIFERAY'S WEB SERVICES

- ❖ When the database classes are generated by the Service Builder, web services can be generated too.
- ❖ As a result, nearly all of Liferay's back-end API calls can be made using web services.
- ❖ Liferay's web services support several protocols including SOAP, Burlap, Hessian and JSON over HTTP.
- ❖ Java clients may be generated from Liferay's WSDL using any number of tools (Axis, Xfire, JAX-RPC, etc.).

WWW.LIFERAY.COM



ACCESSING LIFERAY'S WSDL

- ❖ From the machine on which Liferay is running, use this URL:
`http://localhost:<port>/tunnel-web/axis`
- ❖ A list of all the available WSDL documents will be displayed.
- ❖ It is an extensive list.

WWW.LIFERAY.COM



LIFERAY'S WSDL

And now... Some Services

- Portal_AccountService ([wsdl](#))
 - deleteImage
 - getImage
- Portlet_SC_SCFrameworkVersionService ([wsdl](#))
 - addFrameworkVersion
 - addFrameworkVersion
 - deleteFrameworkVersion
 - getFrameworkVersion
 - getFrameworkVersions
 - getFrameworkVersions
 - updateFrameworkVersion
- WSRPBaseService ([wsdl](#))
 - getMarkup
 - performBlockingInteraction
 - releaseSessions
 - initCookie
- Portlet_Shopping_ShoppingItemService ([wsdl](#))
 - getItem
 - addBookItems
 - deleteItem
- Portlet_Doc_FolderService ([wsdl](#))
 - getFolder
 - updateFolder
 - addFolder
 - addFolder
 - deleteFolder
 - reindexSearch
- Portlet_Blogs_BlogsCategoryService ([wsdl](#))
 - getCategory
 - addCategory
 - addCategory
 - deleteCategory

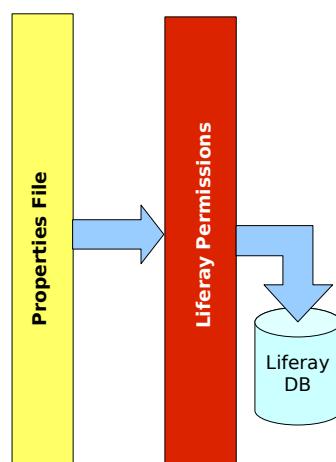
- ❖ If you click on the WSDL link beside any of the descriptions, the document will be displayed.
- ❖ You can save the document and use it to generate a Java client.

WWW.LIFERAY.COM

 LIFERAY.

WEB SERVICES AND SECURITY

- ❖ To access a service remotely, the host must be allowed via the portal-ext.properties file.
- ❖ After that, the user must have permission to access the portal resources.



WWW.LIFERAY.COM

 LIFERAY.

portal-ext.properties

```
##  
## Axis Servlet  
##  
  
#  
# Servlets can be protected by com.liferay.filters.secure.SecureFilter.  
#  
# Input a list of comma delimited IPs that can access this servlet. Input  
# a  
# blank list to allow any IP to access this servlet. SERVER_IP will be  
# replaced with the IP of the host server.  
axis.servlet.hosts.allowed=127.0.0.1,SERVER_IP  
axis.servlet.https.required=false
```

LIFERAY PERMISSIONS

- ❖ The user must already have permission (using the GUI) to access whatever resources will be accessed via the web service.
- ❖ For example, if uploading via a web service to a Document Library folder, the user should already have permission to upload documents to that folder using a browser.

CREDENTIALS

- ❖ Your credentials need to be passed on the URL

`http://<user ID>:<password>@<server name>:<port number>/tunnel-web/axis`

- ❖ For example, to get Organization data:

`http://2:test@localhost:8080/tunnel-web/axis/Portal_OrganizationService`

WEB SERVICES IN PLUGINS

- ❖ You can make your own services in your plugins available as web services.
 1. Set remote-service="true" in your entity configuration in service.xml.
 2. Build your service.
 3. Build wsdd
 4. Deploy your portlet.
- ❖ Access `http://127.0.0.1/your-portlet/axis`
 - ✓ You should see a list of links to WSDL documents for your services.

CONCLUSION

- ❖ It is easy to use Liferay's web services.
- ❖ Since Liferay uses standard protocols you can use your favorite WS tools to access it.
- ❖ Some common uses of the services are:
 - Maintenance operations
 - Imports and exports of data
 - Application integration

Notes: _____



SPRING

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.

FRAMEWORKS

- ❖ A framework is a support structure in which another software project can be organized and developed.
- ❖ Frameworks minimize dependency of application components by providing a plug-in architecture.
- ❖ The thing that makes Spring so powerful is that it combines Dependency Injection, Inversion of Control, and Aspect Oriented Programming.

DEPENDENCY INJECTION

- ❖ Traditionally, if one object were dependent on another, it would create an instance of the other object using the new operator, or it would obtain an instance from a factory class.
- ❖ The DI approach is to provide an instance of the dependent object at run time by an external process.
- ❖ Spring implements this via an XML file which defines the dependencies between objects.
- ❖ Developers can configure dependencies in the XML file rather than declaring them explicitly in code, and they will be injected by Spring at run time.

INVERSION OF CONTROL

- ❖ With the Spring Framework, Inversion of Control occurs via Dependency Injection, and the two terms are interchangeable.
- ❖ Without Inversion of Control, the “coupling” of classes that depend on each other is done explicitly (declared inside the code itself).

ASPECT ORIENTED PROGRAMMING

- ❖ Aspect Oriented Programming (AOP) is the ability to modularize functionality, known as concerns.
- ❖ You can then add this functionality to your classes without making a call to the functionality or coding it directly in your class.
- ❖ This is done by the use of interceptors which Spring can cause to run before and / or after a method is called.

ASPECT ORIENTED PROGRAMMING

- ❖ AOP is generally used for Transaction management, logging – code litter, things unrelated to code.
- ❖ Separation of concern: breaking down a program into distinct parts that overlap in functionality as little as possible.

SPRING IN LIFERAY

- ❖ Spring handles the middle tier, our service layer, using IOC and AOP.
- ❖ Spring also handles our business and data service layers (impl classes and hibernate injections).
- ❖ Liferay utilizes Spring for its Transaction Management:
TransactionProxyFactoryBean
- ❖ Transaction Management – portal-spring.xml

DEPENDENCY INJECTION

- ❖ Spring hides the implementation of our services from the UI/presentation layer.
- ❖ In Action classes, you never call XXXServiceImpl directly; you call Util, which references the factory.
- ❖ Spring always looks up the configured one in the Spring XML.
- ❖ With dependency injection, your code doesn't depend on a certain implementation. Instead it's injected based on what's defined in the XML.

IoC EXAMPLE

- ❖ Suppose a customer wanted to provide his/her own implementation of a class or extend Liferay's.
- ❖ The goal is to replace Liferay's implementation with something else.
- ❖ Using Spring, all the customer needs to do is implement an appropriate class and swap out the class name in the spring XML file.

AOP/TX MANAGEMENT

- ❖ The User Service has a method called addUser()
- ❖ Let's say we wanted to add Tx-management to this method.
- ❖ Traditionally we would:
[tx.begin();
add user commands,
tx.commit();
if something goes wrong, then
 tx.rollback()
]

AOP/TX MANAGEMENT

- However, this isn't in Liferay's code.
- As a consequence of using AOP, this functionality is kept separate from the code.
- `org.springframework.transaction.interceptor.TransactionProxyFactoryBean` acts as a transaction proxy for the underlying service, provided by Spring.

Notes: _____



CHOOSING A DEVELOPMENT STRATEGY

Copyright © 2000 – 2011 Liferay, Inc.
All Rights Reserved.

No material may be reproduced electronically or in print, duplicated,
copied, sold, resold, or otherwise exploited for any commercial purpose
without express written consent of Liferay, Inc.

GOALS

- Overview of the available methods for extending Liferay Portal's functionality
- Understand the compromises involved in each of the methods
- Evaluate how to combine the different methods for the needs of your organization

LEVELS OF EXTENSIBILITY

- JSR-286 Portlet
- Extension Points
- Liferay Services, Utilities and Liferay-specific Attributes
- Extending Liferay Classes
- Overwrite JSPs
- Use of Internal APIs
- Modifying Liferay Source

- Liferay Portal is extensible at several levels
- Each level of extensibility offers a different compromise of flexibility vs ease of migration to future versions
- Choosing the appropriate level for the task at hand allows for easier future maintainability
- *Keep your code in the green!*

WWW.LIFERAY.COM

LIFERAY.

LEVEL 1: JSR 286 Portlets

- Developed as independent software components
- Distributed and deployed as WAR files
- Fully portable to other JSR 286 compliant portal servers
- JSR 168 still fully supported by Liferay as well

WWW.LIFERAY.COM

LIFERAY.

LEVEL 2: EXTENSION POINTS

- ❖ The extension points or hooks allow creating custom classes for the most common extensibility needs
- ❖ Deployed using a Hook or EXT Plugin
- ❖ Configurable through portal(-ext).properties
- ❖ Examples:
 - Authentication chain, Upgrade and verification processes, Deployment processes, Database access and caching, User fields generation and validation, session events, permissions, model listeners, ...

LEVEL 3: PUBLIC APIs & COMPONENTS

- ❖ Liferay provides enhancements to the portlet specification that you can take advantage of in your portlets
 - liferay-portlet.xml, liferay-display.xml
- ❖ Decrease development time and increase productivity by leveraging Liferay utility classes and methods
 - PortalUtil, PropsUtil, etc.
- ❖ Liferay employs a Services Oriented Architecture that exposes the majority of Liferay features through a service layer. Your custom portlets can access these services.
 - User Service, Document Library Service, etc.

LEVEL 4: OVERRIDING CLASS IMPLEMENTATIONS

- ❖ Using Hooks, it is possible to provide customized implementations of Liferay services.
- ❖ Follow best practices to minimize upgrade risk
 - Always provide a unique java package; do not use the same package as the implementation you are overriding.
 - Extend the original impl and only overwrite the methods needed.
 - Do not add new public methods.
 - If you must add new methods to a service, create a new service referencing the service you want to extend.

LEVEL 5: OVERWRITING JSPs

- ❖ JSPs should be overwritten using Hooks, not EXT Plugins
- ❖ You can customize any of the JSPs used by Liferay's portlets and management tools
- ❖ Very flexible extension mechanism
- ❖ Must be used with caution
- ❖ Recommended method for certain extensions

LEVEL 6: USING INTERNAL APIs

- ❖ EXT Plugins allows native access to portal class loader.
- ❖ Plugins can also access the portal class loader via PortalClassInvoker.
- ❖ ServiceBuilder simplifies class loading issues.

LEVEL 7: MODIFYING LIFERAY SOURCE

- ❖ Liferay Portal Community Edition is distributed with source code under the Lesser GNU Public License (LGPL) v2.1.
- ❖ Liferay Portal Community Edition source can be modified as long as you are in compliance with this license.
- ❖ Liferay Portal Enterprise Edition does not support direct modification of the source.
- ❖ Recommended method: Sponsored Development
- ❖ Alternative: Develop for project and contribute back

DEVELOPMENT STRATEGY

- To determine your development strategy, ask yourself if your project requires any of the following:
 - Changes to Liferay's default user interface that are *not* theme-related?
 - Integration with other products that is not yet provided by Liferay?
 - Modification to an algorithm Liferay uses
 - Modification of the behavior of one of Liferay's built-in portlets

DEVELOPMENT STRATEGY

- If you answered yes to any of the questions on the previous slide, you will need to use a hook or EXT plugin.
- Next, ask yourself the following question:
 - Would my changes be a useful addition to the product? Does it enhance the product in some way that is usable by other organizations?
 - If so, you may wish to contact Liferay to sponsor having the feature added into the core product itself.
 - Alternatively, you can implement the feature yourself and contribute it back to the project. Either way, everybody wins!

Notes: _____



SUMMARY

Copyright © 2000-2011 Liferay, Inc.

All Rights Reserved.

No material may be reproduced electronically or in print, duplicated, copied, sold, resold, or otherwise exploited for any commercial purpose without express written consent of Liferay, Inc.



YOU ARE NOW A LIFERAY DEVELOPER!

- Over the course of the past few days, you've been exposed to many powerful features of Liferay's development platform.
- The information in this course equips you to build your site on Liferay's platform, taking advantage of the features and conveniences the platform has to offer.

WHAT'S NEXT?

- In the next year, Liferay will be creating an Advanced Development course. Watch our website for an announcement.
- Further information is also provided in *Liferay in Action*, Liferay's official guide to development, which is published by Manning Publications. You can find more information about the book here: <http://manning.com/sezov>.
- Please keep in touch on Liferay's forums and if you can, help us out by contributing plugins, core code, or wiki articles. We love collaborating with our community!

WWW.LIFERAY.COM



We have very much enjoyed working with you, and wish you much success in your Liferay projects!

WWW.LIFERAY.COM

