

Liferay IN ACTION

The Official Guide to Liferay Portal Development

Richard Sezov, Jr.

MEAP

 MANNING





**MEAP Edition
Manning Early Access Program
Liferay in Action MEAP version 8**

Copyright 2011 Manning Publications

For more information on this and other Manning titles go to
www.manning.com

Table of Contents

Preface

Acknowledgments

About this book

About the authors

Part 1 Working with Liferay and Portlets

1. The Liferay difference
2. Getting started with the Liferay development platform

Part 2 Writing applications on Liferay's platform

3. A data-driven portlet made easy
4. MVC the Liferay way
5. Designing your site with themes and layout templates
6. Making your site social
7. Enabling user collaboration

Part 3 Customizing Liferay

8. Hooks
9. Extending Liferay effectively
10. A tour of Liferay APIs

Appendixes

- A. Liferay and IDEs
- B. Introduction to the Portlet API
- C. Inter-portlet communication
- D. How to contribute to Liferay

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

Part 1

Working with Liferay and Portlets

The first part of this book gives you an introduction to Liferay, showing you what it is, what it does, and how you can use its powerful features to implement your web site.

Chapter 1 gives you an introduction to the portal landscape and how Liferay leads in that space to provide the features that developers have wanted from the platform all along. You'll learn how Liferay's users, roles, communities, and organizations work and how to navigate its interface. You'll also see how to run a Liferay development project, using best practices that have been proven to work.

Chapter 2 hits the ground running by introducing you to Liferay's development tools. You'll learn how to use the Plugins SDK to create Liferay projects, and you'll write your first portlet, using the industry standard Portlet API.

This first part of the book prepares you for all of the nuances of Liferay's development platform that are yet to come in the rest of the book. You'll understand what Liferay is for and how you might use it to implement any web site. And you'll have already used Liferay's development tools to write your first portlet.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

The Liferay difference

This chapter covers

- Understanding portals then and now
- Exploring what Liferay is and how to work with it
- Defining basic Portal concepts
- Using Liferay to design a portal

Everybody needs a web site these days. Whether you're building one for a company, for a service organization, or for personal reasons, you need one. And when trying to decide how to build it, you've probably found a dizzying array of choices running on a dizzying array of platforms. So how do you go about choosing which platform is best?

If you're anything like me, you've got a list of a whole bunch of products. You created this list by going through the feature claims of a various software platforms that seem to do something close to what you want to do with your site. Now you're going through that list, testing the products out, weighing their strengths and weaknesses against each other, and weighing those against how well those products' underlying platforms will fit into your infrastructure.

If Liferay Portal isn't on your list, you should put it at the top right away. Liferay Portal is a Java-based open source portal, containing an unprecedented number of features which will help you to implement your site in as little time as possible. And once you have Liferay on your list, let me respectfully submit that your search can end with Liferay Portal, which is hands down the best platform upon which to build a web site.

I feel safe in saying this because I was a Liferay user for some time before I wound up working for them. So yes, I took the red pill,¹ so to speak, but I've also experienced Liferay from the outside, and so I know what it's like to be doing that search for a platform. I can tell you from experience that you're going to find working with Liferay to be a pleasure, and you'll be happy to know that using the platform that Liferay offers you will free you from limitations. It speeds up your development cycle and gives you features that you likely wouldn't have had time or the inclination to build yourself. Most of the time, potential Liferay users focus on Liferay as a product—because it boasts such a huge range of features—but they don't stop to consider the rich development platform it offers. By the end of this chapter, you'll have a good understanding of what Liferay is all about and what it can do for your web site. And I have no doubt that you'll find many reasons to choose Liferay for your next development project.

Choosing Liferay is also safe: You're putting yourself in a group with some of the largest organizations (with the largest web sites) out there that have also chosen Liferay. So if I can give you any advice, it would be to end your search with Liferay and begin learning how you can leverage the platform to build the site of your dreams.

This chapter will go over several several important topics. I'll show why Liferay calls itself a "portal," what a portal used to be, and how Liferay pioneered getting past its early limitations. We'll then take a helicopter ride over Liferay's feature set to see what it can do at a high level. After this, we'll delve into how Liferay helps you structure a web site. You'll also get to see what Liferay looks like by default and how you can navigate around it. And finally,

¹ From the 1999 film *The Matrix*.

using all the information we've presented, I'll show you how you can begin to imagine how your site might be implemented using Liferay Portal.

But first, to get our bearings, let's start by exploring why Liferay calls itself a portal and what that term has come to mean in the industry historically.

1.1 *The Java portal promise: from disappointment to fulfillment*

Liferay calls itself a portal. What do you commonly think of when you hear the word portal? As a big fan of sci-fi and fantasy, I tend to think of a doorway to another dimension or time like the portal that Kirk and Spock went through, chasing after McCoy to stop him from doing whatever he did to change the timeline. I'll tell you right away: Liferay Portal isn't that elaborate (but you've likely already figured that out). So why do we call it a portal? Let's start with the so-called official definition of a portal.

Portal

A portal is a web-based gateway that allows users to locate and create relevant content and use the applications they commonly need to be productive.

That comes from a bullet on a slide I've used to teach Liferay to prospective users. I might even have written that bullet, but I'm not sure. Generally, the reaction I get is a narrowing of the eyes, some pursed lips, and then heads begin nodding up and down. This tells me that people want me to think that what I've just said makes sense, but they're being kind and reserving judgment on my teaching abilities, because it actually made no sense at all.

The problem with definitions like that is that they try to say too much in one sentence. Liferay is many, many things, and you can't capture it all in one sentence. But just for fun, let's try it again.

Portal

A portal is designed to be a single web-based environment from which all of a user's applications can run, and these applications are integrated together in a consistent and systematic way.

That one's a bit closer when viewed in the context of the web. When we talk about Liferay as a development platform, that's exactly what we mean. At its base, Liferay is a container for integrated applications. Those applications are what make the difference between Liferay and competing products. You're free to use the applications you like, write your own, and disable the rest. And this is what sets Liferay apart. Figure 1.1 shows how you can easily mix and match your applications with Liferay's.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

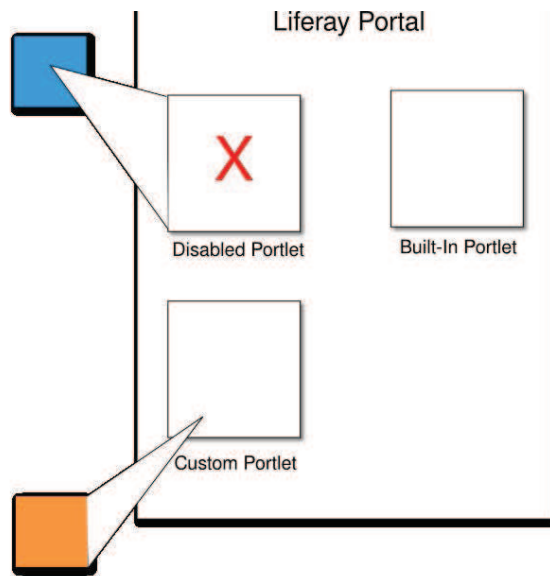


Figure 1.1 Liferay contains many built-in applications, called portlets. If there are some you'll never use, you can disable them. You can also write your own portlets and deploy them. These custom portlets are indistinguishable from portlets that ship with Liferay Portal.

As an analogy, think back to the eighties and early nineties. If you bought a computer and you needed to use it to write something, you also bought a word processor. If you then decided you wanted to calculate numbers with it, you bought a spreadsheet. And if you needed to store and retrieve data of some kind (perhaps for a mailing list), you bought a database. (Nobody created electronic slides back then; they used an overhead projector. And yes, I am dating myself.)

Most of the time, people would pick what was considered the best of whichever program they wanted. One vendor had the best word processor. Another had the best spreadsheet. A third had the best database. So if you had to perform all three functions, it was likely that you had three separate programs written by separate entities, but individually they were the best.

Pretty soon, people wanted to create graphs in their spreadsheets that they would insert into a word processing document that they would send to a mailing list stored in the database. The problem with that was that all of these programs were created by different vendors, and they didn't always work together all that well. Much effort on users' parts had to be spent on trying to get them to work together well.

You know the rest of the story. We wound up with office suites, consisting of programs written on the same platform that were designed to work together. Not only did this save us all some money (because buying the separate programs cost a fortune), it also gave us a level of integration that had so far been unavailable.

The same thing is happening today with software on the web. Liferay is an engine for running web sites. Liferay consists of the base engine as well as the many applications that run on that engine. When you use this platform, your applications can have a level of integration with the rest of Liferay's applications that will make your users' experience seamless and smooth. Why? Because the integrated experience is far better than the nonintegrated experience. This is the difference that makes Liferay stand above all the other portals out there. I have to say that because there are some who view the word "portal" with disdain, and sometimes this is with good reason. Let me explain further.

1.1.1 The Java portal disappointment

When Java portals were first announced, they were hailed as the solution to many of the problems facing enterprises and solution architects. The web had grown up. Instead of proprietary interfaces to everything, everybody had finally standardized on TCP/IP networking and open protocols such as HTTP, IMAP, SMTP, SOAP, and the like. Services, applications, and email operated on these open protocols, and products that once had relied

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

on proprietary protocols had now opened up to the web. Those products that didn't (or whose vendors had delayed it) were relegated to the dust bin of history. And once we had all of these siloed services speaking the same language, we needed something to bring it all together for the end user.

Enter the Java portal. The release of the Java portal specification came with the promise of bringing all of these services together in a single unified "web desktop." Not only would it unify everything for a corporation's internal applications, it would also be the hub of all B2B (business to business), B2C (business to consumer), B2E (business to employee), and even G2P (government to public) communication. It would be the presentation layer for the brand-new service-oriented architecture that you just finished (or were in the process of) implementing. It could also be a platform for new applications. And it could finally bring together your static web sites and your applications, which resided on separate application servers.

Do you think too much was promised? How's that old saying go? "If it seems too good to be true, it probably is?" Well, you're right. What happened? At least three issues emerged that prevented Java portals from achieving widespread acceptance.

For one, it was difficult to develop solutions using a portal. The initial Portal API turned out to be something like getting to the least common denominator. Instead of providing all the features developers would need to bring all this stuff together, it defined what seemed like the absolute minimum that all the vendors could standardize on and then left everything else up to the individual vendors. This meant that developers had to spend more time implementing features that should have been part of the platform in the first place. One example of this is that the initial standard did not include any way for portlets to communicate with each other.

Second, the portal servers themselves were too big and complex (not to mention hideously expensive), often taking days to get set up. And for the developer trying to get a development environment going, it was sometimes even worse. I can remember trying to work with one of the first portals (sorry, can't tell you which one it was) and finding it impossible to get a development environment properly configured on my laptop. At the time, I was a team lead and was trying to get this install process to a repeatable procedure for the rest of the developers on my team. My solution? I went to a conference, grabbed one of the presenters after his talk, and made him help me install the development environment on my machine. When he heard of my plight, he understood completely and told me everybody was having this problem and that they had to make this process easier.

Third, other things were happening in the industry at the same time. The Web 2.0 concept was beginning to get popular, and the portlet specification had left no room whatsoever for enabling a rich, client-side experience for the end user. In order to compete, portal vendors started to implement their own proprietary extensions to the portlet specification. We all know what this leads to: vendor lock-in, which is precisely what defining a standard is supposed to avoid.

At the same time Java portals were getting a bad rap, sites like Facebook and MySpace came out and pretty much implemented what portals were meant to do all along. And as they got more and more popular, suddenly other sites like Amazon.com and other software like Jira began to implement the social collaboration features Facebook and MySpace had, along with their slick, AJAX-enabled user interfaces. What powered all of these new and improved web sites? What enabled them to implement such rich features for the end user so quickly? You guessed it: open source.

Open source solves a lot of the problems inherent in the old Java portal paradigm. Open source projects don't wait around for committees to decide on things; they tend to implement what the users want as fast as possible. There are no barriers to entry with open source; the development tools and the software are made available for free. Open source products also tend to be lighter weight: you don't need a large, dedicated server to start building your solution. Development goes faster, because developers don't have to learn the entire architecture to be effective. And you don't need a huge initial investment to get started using an open source solution. You can start small (free) and then grow your application and hardware as your needs grow. Facebook is the perfect example of this: Implemented using PHP (which is an open source web development platform), the site has grown organically as its user base has grown. This is what the market really wanted. And as you're about to see, Liferay Portal provides the same kind of open source platform that has allowed many organizations to do the same.

1.1.2 Liferay keeps the Java portal promises

From the beginning, Liferay Portal has been an open source project. Its whole purpose for existence was to level the playing field so that smaller organizations such as nonprofits, small businesses, and open source projects could

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

take advantage of its platform without having to incur huge expenditures for either software or hardware. So right out of the gate it was doing things differently. An open source project doesn't have the luxury of making it difficult for developers to work on the platform. Instead, developers need to find the platform to be easy to work with, or the project will have major hurdles to community gestation. And if an open source project can't foster the birth and growth of a vibrant community, it's dead. So right away, Liferay was (and continues to be) easy for developers to use, adapting to many different development styles, and not requiring any specific tools to be installed beyond what is already in any Java developer's toolset.

This same philosophy translates to its size. Open source projects also don't have the luxury of being too big or taking up too many system resources; they may be running on new hardware or five-year-old hardware that was donated to a nonprofit that can't afford anything else. Liferay Portal is much smaller and simpler to configure than its competitors. Can you run Liferay on big hardware with a proprietary Java application server? Sure you can. Can you run it on a shared server with a small servlet container like Tomcat? Absolutely. Liferay Portal is provided as a standard .war file—only 125 MB in size—which can be installed on any application server, or as a "bundle," preinstalled in your open source application server of choice. You don't have to go through long installation routines and complex command-line incantations to get it working. If you use a bundle, installing Liferay is as easy as unzipping an archive and editing a text file to point it to your database.

And guess what? Instead of giving you by default an empty portlet container into which portlet applications can be installed, Liferay Portal comes with over 60 portlet applications included. These applications cover about all of the "standard" functionality you're likely to need in a web site: content management, forums, wikis, blogs, and much more—leaving you to implement only the features specific to your site. And for developers, your setup time will be measured in minutes, not hours. You also don't have to know everything about the architecture to be effective—it's really easy to get started.

Open source software also has to be innovative in order to compete with its proprietary competition. Liferay Portal was the first portal to implement that slick, Web 2.0 interface, back in 2006. The first time I saw a portlet being dragged across the browser window and dropped into another spot on the page, I was blown away, because I was used to the old, proprietary solutions that hadn't implemented that yet. Because Liferay Portal was open source, it could respond to market demands faster than the other guys, using the same standards they were using. You'll continue to see that in Liferay Portal, because the open source paradigm works. What users demand gets implemented, without sacrificing adherence to standards.

As far as standards go, Liferay is also based on widely used, standard ways of doing things. It adheres to the JSR-286 portlet standard. In addition to that, though, it includes utilities such as Service Builder to automatically generate interfaces to databases (something not covered by the standard). Under the hood, Service Builder is just Spring and Hibernate—widely used by Java developers everywhere. So you get the benefit of using the platform to get your site done faster, while still taking advantage of standards that keep your code free.

Now that I've spent so much time extolling the virtues of this magical, mystical thing known as Liferay Portal, you're probably anxious to see what this wonderful specimen I've described looks like.

1.2 Getting to know Liferay

Liferay Portal is an open source project that uses the LGPL open source license. This is the GPL license you know and love with one important exception: Liferay can be "linked" to software that is not open source. As long as you use Liferay's extension points for your custom code, you don't have to release your code as open source if you don't wish to. You can keep it, sell it, or do whatever you want with it; it's yours. If, however, you make a change to Liferay itself by modifying Liferay's source code and want to redistribute the product thereafter, then you need to contribute that change back to Liferay. So you get an important exception with the LGPL: You can still use Liferay as a base for your own product and either open source the result or sell it commercially if you wish. Or, if you want to change Liferay directly, you can contribute to the open source project. It's entirely up to you. You can download the open source version of Liferay Portal for free from Liferay's web site.

Alternatively, Liferay sells an Enterprise Edition of Liferay Portal. This is a commercially available version of the product that comes with support and a hot-patching system for bug fixes and performance improvements. There are web sites running on both versions of Liferay Portal, and both are perfectly appropriate for serving up your site.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

In this section, we'll take a quick tour of some of the things you can do quickly with Liferay to begin building a web site. We're going to play around with the interface a bit so you can get to know it a little better. Figure 1.2 shows the default Liferay Portal 6 user interface.

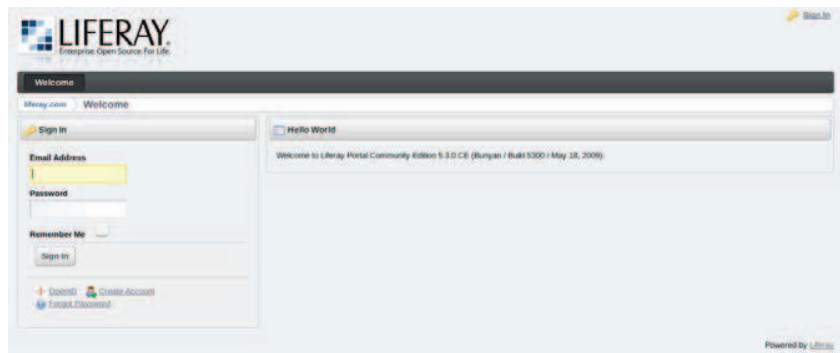


Figure 1.2 Liferay Portal 6, as it looks the first time you start it. It presents you with a basic interface at first, but as you'll see, you can easily jazz it up.

Okay, I agree; it doesn't look like much, does it? But there's an awful lot of power hidden in the humble interface that Liferay shows you by default. If you're ahead of the game and already have Liferay running, you can follow along. If not, just sit back and enjoy the ride: we'll go over how to get Liferay installed and running on your system in Chapter 2.

1.2.1 Liferay is an application aggregator

We've been saying that Liferay Portal is not just a product; it's a platform. This platform runs applications, and these applications are integrated together in ways that separate applications cannot be, by virtue of their shared platform.

What this means is that we can take that default Liferay page and load it up with integrated applications. Liferay makes doing something like that very easy. First, we have to log in as the default administrative user, whose user name is `test@liferay.com` and whose password is `test`. This will display the *Dockbar* (see figure 1.3) at the top of the page, which gives us access to several other functions.

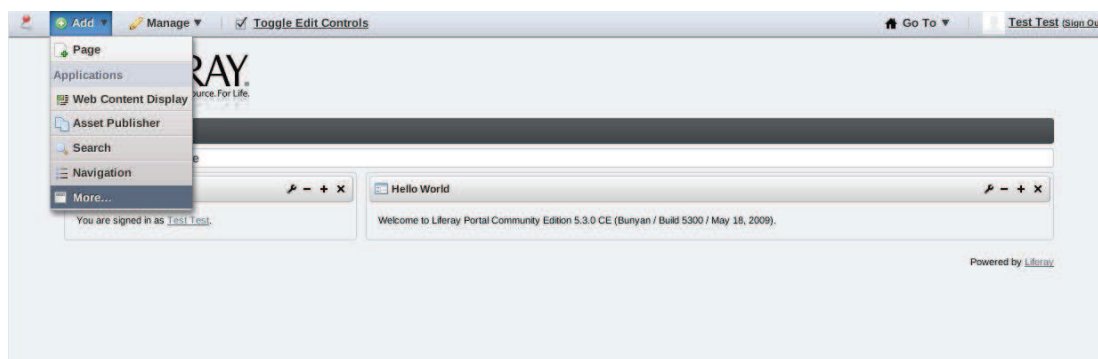


Figure 1.3 Hovering the mouse over the Add menu in the Dockbar opens a drop-down menu. To see a full list of available applications, click *More*.

We'll come to all of the things you can do with the Dockbar in a moment. For now, however, all we want to look at is applications, which you can access from the *Add* menu. Commonly used applications appear directly in the menu, but if you want to see the whole list, click *More*. This will pop up a fully searchable, categorized view of all

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

the applications that have been installed in your Liferay Portal by default. As an aside, by the time you are finished with this book, one of the things you'll be able to do is write your own applications, which can appear in this list.

We're going to fill this page with applications so you can see how Liferay aggregates them. You can browse the applications by opening the categories to which they're assigned. Or if you know the name of the application you're looking for, you can search for it by using the search bar at the top of the Applications window. Let's pick some cool applications to add to our page. Note that in a real-world web site, you'd likely never put all of these on one page—we're doing an experiment here to show the concept. To the left column, add Navigation, Activities, Dictionary, and Translator. To the right column, add Message Boards, Wiki, and Calendar. You can add an application to a specific column by dragging the application off the Applications window and dropping it into the appropriate column, as shown in Figure 1.4.

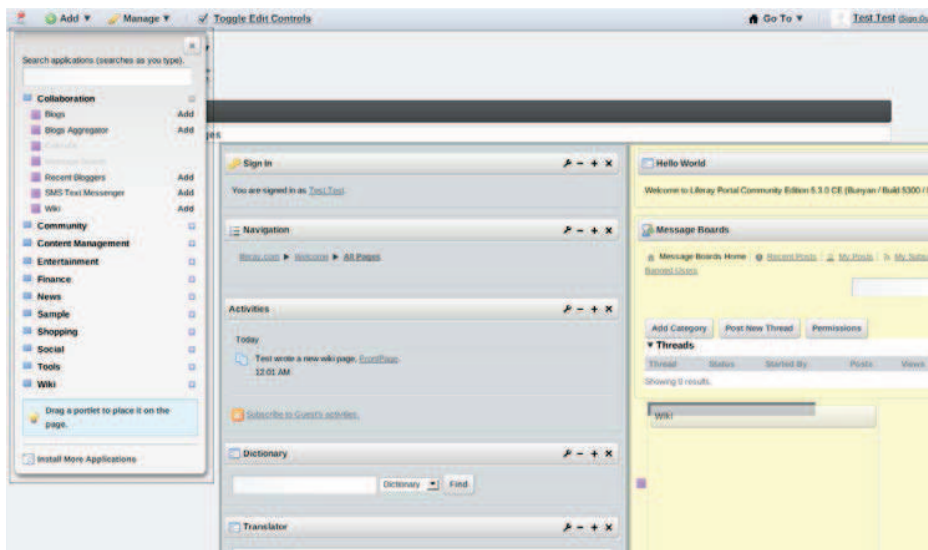


Figure 1.4 Most of the applications have been added. This screen shot was taken while dragging the Wiki application into the column on the right.

Now we have a single page with a whole bunch of applications on it. These applications can perform a lot of different functions.

The Message Boards application is a complete implementation of web-based forum software. If you're planning to have discussion forums on your web site, Liferay already has them built in. And the cool thing about it is you don't have to integrate anything. They already work with Liferay's user management and security features, as do all of Liferay's applications.

You've also added a Wiki application to the page. Again, this is a full-fledged wiki that you can use for whatever purpose suits you. As with the Message Boards application, the Wiki is integrated with Liferay's user management and security. But (and this is the cool part) the Wiki also is integrated with Liferay's Message Boards application, because it borrows functionality from that application to provide comment threads at the bottom of Wiki articles. Those threads will use your users' profile information (including pictures) in the threads to uniquely identify them in a consistent way throughout your site, which is yet another level of integration.

What about the Calendar? Again, it's totally integrated, complete with email notifications and more. And it's a full-fledged calendar application that supports export and import of calendar data from other applications.

The other applications are smaller, and I don't want to gloss over them, but you're probably getting the picture at this point. Let me point out one other thing, though, and that's the Activities application. Notice in figure 1.5 what it says.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

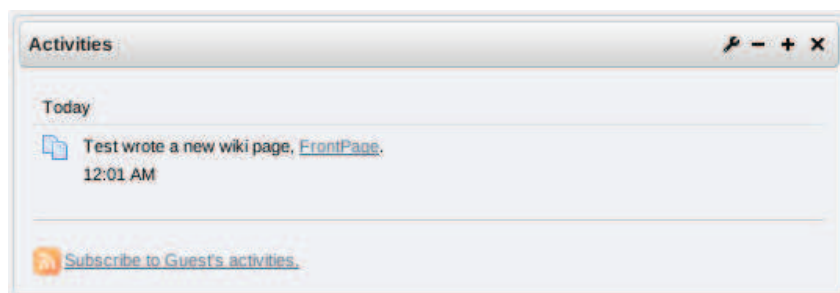


Figure 1.5 Every application in Liferay that uses the Social API can capture activities unique to that application. The Activities portlet displays those activities. Did you really create a new wiki page?

When you added the Wiki application to your page, you created a top-level Wiki page, which is by default called FrontPage. Because the Wiki application uses Liferay's Social API to capture its unique activities, the Activities application can report on what you did (and even provides an RSS feed of activities). Liferay has an API that lets you tap into this social capability. This opens up all kinds of possibilities for your own applications, doesn't it? (We'll cover this API in detail in chapter 6.)

Note Because Liferay is a portal, its applications are called portlets. I have been careful so far to refer to them only as applications, but for the remainder of this book, we'll use the terms portlet and application interchangeably.

Naturally, we'd never in the real world create a page such as this. Your users would throw conniption fits if they had to navigate such a thing. You might do better by your users if you put some of these applications on different pages. I just wanted to illustrate how integrated Liferay's applications are.

In addition to providing a development platform and a slew of applications out of the box, Liferay is also a powerful content management system (CMS).

1.2.2 Liferay is a content manager

If you have lots of web content that you wish to publish and you wish to publish that content using a workflow, or on a schedule, statically or dynamically, to staging or production, with templates or without, then you might want to check out Liferay's CMS.

You can access the web content functions from the same Applications window you've already seen; in fact, they're in their own category. But the quickest way to do it is to simply select Web Content Display, right from the *Add* menu. It's right in the menu for convenience—if you are building a content-rich web site, you'll use it a lot. Once it's added, you can drag it to whatever position on the page you want. Figure 1.6 shows this portlet added to the right column on the page.

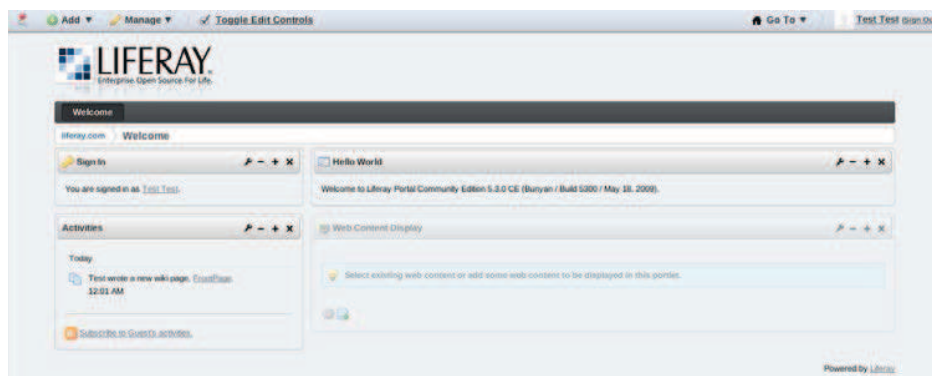


Figure 1.6 The Web Content Display portlet is added, but it has no content (yet). We'll remedy that very quickly.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

A Web Content Display portlet does what its name implies: it displays web content. So in order for it to do its job, you'll have to create some web content. You can do that very quickly by clicking the *Add Web Content* icon, which is the icon at the bottom-right. You are then brought to a form where you can add content. Figure 1.7 shows this form.

For now, don't worry about all the options on the right side of the screen (Structure, Template, Workflow, and so on.). For basic content management, all you have to do is start adding content. Give your piece of content a name and a description, and type some content into the editor. Notice in figure 1.7 that you can apply all sorts of formatting in the editor: fonts, tables, bullets, colors, and images.

Figure 1.7 Entering content in Liferay's Content Management System.

Once you've finished adding your content, notice the buttons at the bottom of the page. Though there is a whole workflow process you can go through, you are logged in as the portal administrator. This means you can short-circuit the workflow process by clicking the button marked *Save and Approve*. So go ahead and do that. You'll be brought back to your original portal page, and the Web Content portlet will contain the content you just added.

We could go further with Liferay's Web Content Management system, but suffice it to say that it's sufficiently powerful for whatever content needs you may have. For example, you can create your own structures with custom fields for your content, as well as templates to go along with your structures to display your content in exactly the way you want it displayed. You can stage your content on a staging server and have it published on a schedule of your choosing. You can write powerful, scripted templates in XSL, Velocity, or Freemarker.

You've seen so far that Liferay can be a platform or a UI for your applications, and it can also manage your site's content. The last ingredient that you need Liferay provides in spades, and that's a way for your users to find and collaborate with each other.

1.2.3 Liferay is a collaboration tool

Liferay Portal is ideal for setting up collaboration environments among workgroups. Whether you call these environments communities or virtual team rooms, Liferay can be used to help your team get their work done. It

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

does this by providing applications which are geared specifically toward document sharing and communicating with one another.

One of the portlets you can add to a page in Liferay is the Document Library. This application provides a facility for sharing documents with your entire team. It keeps a complete version history of all of your documents and is integrated with Liferay's permissions system. This integration allows you to grant access to shared documents or prevent some of your users from accessing sensitive documents. And if your users need an easier way to access the documents than the web interface provides, the Document Library supports WebDAV, allowing documents to be uploaded and downloaded through their operating system's familiar interface. Figure 1.8 shows both of these interfaces.

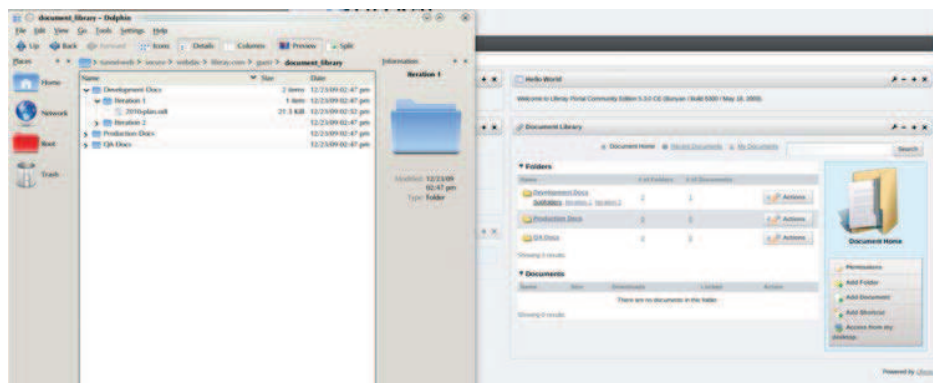


Figure 1.8 Accessing the same folders in the Document Library in the operating system via WebDAV or using the browser interface.

Documents are one thing, but what about communication? Liferay's portlets allow for communication right in context, so your users can keep all the relevant information in the right place. So the Document Library allows your users to create discussion threads right next to the documents they need to talk about. The Wiki does the same thing. And applications are provided for both chat and email, so that currently logged-on users can communicate in real time, no matter what their physical distance is from one another.

Need a group calendar? The Calendar portlet can be used for either individuals or for groups. Additionally, your users can all have their own individual blogs on their own pages which are then aggregated using the Blogs Aggregator to the community page. This enables you to display a "blog of blogs," allowing your team to stay updated on what everyone is doing. Combining this with Activities makes for a consistent, rolling list of what the team is up to.

All of the functionality I've mentioned so far is what is built in to Liferay (and there's more we haven't touched on). But Liferay is extensible too.

1.2.4 Liferay is anything you want it to be and any way you want it to look

Liferay offers a level of customization that is unparalleled, because you can modify *anything* in Liferay, from simple functionality changes all the way to making your own product out of it.

This book will systematically show you how to write your own portlets so that your applications can be added seamlessly to your Liferay-powered web site's pages in a way that is indistinguishable from the built-in portlets. You'll also learn how to customize Liferay's layout templates so that your page layouts can be what you want them to be. You'll also learn about hooks, which let you customize Liferay by substituting your own classes and JSPs in the place of Liferay's. And finally, you'll learn about Ext plugins, which let you override anything in Liferay with functionality of your own.

No discussion of customizing Liferay would be complete without covering themes. Using themes, you can transform Liferay's look and feel to make it look any way you want it to (see figure 1.9). In short, Liferay can be anything you want it to be, and it can look any way you want it to look. This gives you the power and flexibility you need to build your own custom site, with the functionality you need to get it done in a timely fashion.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

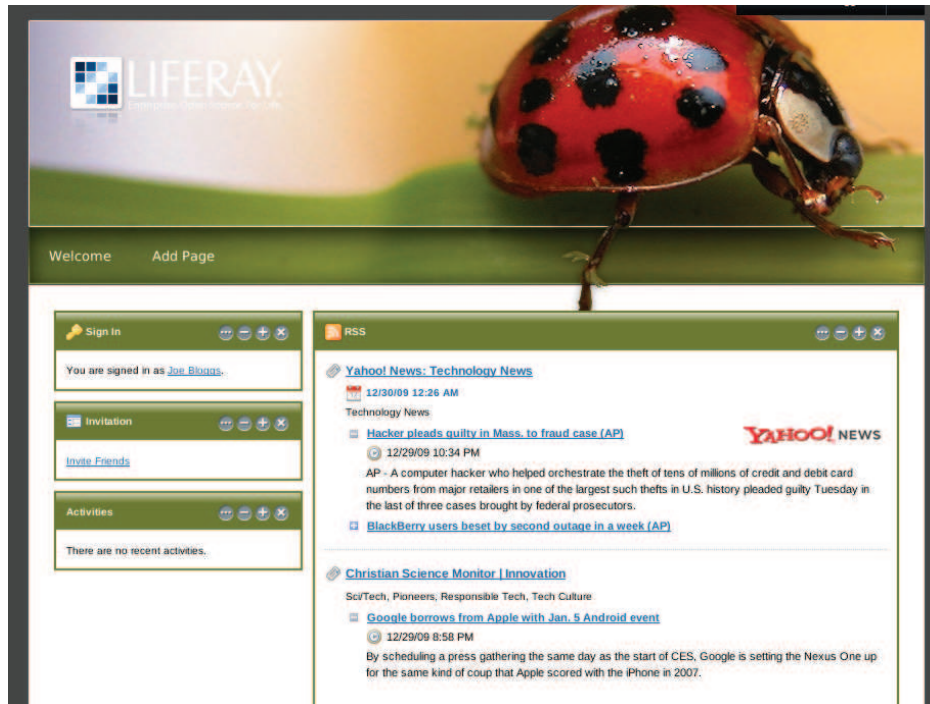


Figure 1.9 Liferay with a custom theme applied. This is just one of many themes in Liferay's community repository.

Liferay provides you the freedom to make your site look the way you want it to look, using skills you already have. Themes are nothing more than custom HTML and CSS applied to the page. So you'll have the same ability to design your site as you would have if you were writing the whole thing from scratch—except for the fact that you'll have less work to do, because of Liferay's built-in functionality and rich development platform.

Liferay also comes connected to two repositories of ready-made plugins which extend Liferay's functionality. One of these is a Liferay-provided repository, and the other is a community repository. The screen shot in figure 1.9 above is an example of a theme provided by Liferay's community through the community repository. Liferay's repositories make it very easy to both distribute and to install new software that runs on Liferay's platform, as you can see in figure 1.10.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

Portlet Plugin	Trusted	Tags	Installed Version	Available Version	Modified Date
Add New User Wizard 5.2.2.1 ID: robissoft/03/5.2.2.1/war A plugin portlet using Spring Wizard	No	sample, spring, wizard	-	5.2.2.1	3/22/09 1:24 AM
WOL 5.2.2.1 ID: liferay/world-of-liferay/5.2.2.1/war This is the World of Liferay portlet that integrates social networking features.	Yes	wol	-	5.2.2.1	2/27/09 5:49 PM
Web Form 5.2.2.1 ID: liferay/web-form/5.2.2.1/war This is the Web Form Portlet.	Yes	web form	-	5.2.2.1	2/27/09 5:49 PM
Mail 5.2.2.1 ID: liferay/mail-portlet/5.2.2.1/war This is the Liferay Mail portlet.	Yes		-	5.2.2.1	2/27/09 5:46 PM
Google Maps 5.2.2.1 ID: liferay/google-maps-portlet/5.2.2.1/war This portlet allows easy integration with Google Maps.	Yes	google	-	5.2.2.1	2/27/09 5:45 PM

Figure 1.10 Browsing Liferay's plugin repository from within the control panel. Installing any plugin is a simple matter of clicking on the plugin and then clicking the *install* button that appears with the full description of the plugin.

As you can see, a lot of functionality is built in to Liferay Portal, and it is also extremely easy to add functionality to Liferay Portal. So you can rest assured that the software you create on Liferay's platform will be easily installed by your users. Let's take a step back now so we can see what we have so far accomplished with just a few clicks.

1.2.5 So what has this little exercise accomplished?

Hopefully you see the power that Liferay gives to you. In about ten minutes—and without any additional software—we've created a web site that contains web content, forums, a wiki, displays users' activities, shares documents, and has a custom look and feel. We didn't have to get separate applications to do all of that—instead, all that functionality (and more) is already included in Liferay. And because we didn't have to use separate applications to implement what we wanted, we didn't have to spend any time integrating those applications. Users get the experience of being able to sign into your site once and then navigate to the content they have access to, and you don't have to do *anything* to make that work.

Pretty awesome, isn't it?

Obviously, this only scratches the surface. You're going to need ways of organizing and granting permissions to all those users you're going to have. In order to do this, you'll need to understand the reinforcement beams, foundation blocks, and structures Liferay gives you to support that portal full of users.

1.3 How Liferay structures a portal

Every portal is different in the way users, security, and pages are handled. Because these aspects of a portal are not covered by the JSR-286 standard, every portal vendor has implemented these concepts differently. So if you

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

are going to start developing on Liferay's platform, you'll need to understand how a Liferay portal is configured and organized. Don't worry: it's not all that complicated, though it may look that way at first. Once you start using the system, you'll get the hang of it very quickly.

In this section, we'll see how you can collect users into various categories and what those categories can do for you. We'll also see how Liferay makes it easy to create web pages in your site and how content is placed on them.

1.3.1 The high-level view

At its most basic level, a Liferay server consists of one or more portals. Portals have users, and these users can be categorized into various collections. Some of these collections can also have web pages that compose a portion of your site.

You can define many portals per portal server, and each portal has its own set of users and user collections. Figure 1.11 displays this graphically.

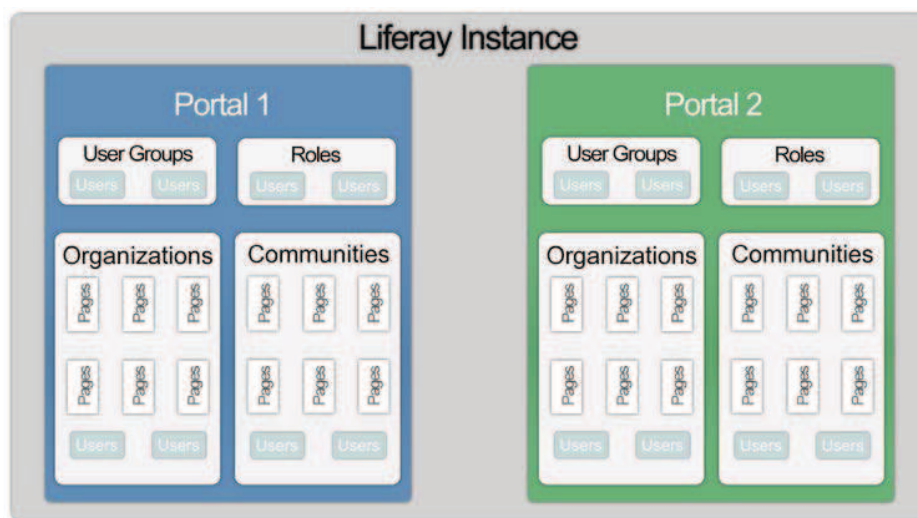


Figure 1.11 A single Liferay Portal installation can host many different portals, all with separate users and content.

As shown in figure 1.11, each portal has users, and those users themselves can be organized into several different types of collections: Roles, Organizations, Communities, User Groups, or any combination of those collections within that portal. Table 1.1 lists the collection types Liferay offers.

Table 1.1 Liferay Collection Types

Collection Type	Description
Role	Collects users by their function. Permissions in the portal can be attached to roles.
Organization	Collects users by their position in a hierarchy. Organizations can be nested in a tree structure. You would use organizations to represent things like a company's organizational chart.
Community	Collects users who have a common interest. They're single entities and can't be grouped hierarchically. By default, users can join and leave communities whenever they want, though you can administratively change it so that users are assigned to communities (or invited) by community administrators.
User Group	Collects users on an ad hoc basis. Defined by portal administrators

- Roles are inherently linked to permissions. You'd use a role to collect users who have the same permissions. A good example of this would be a Wiki Administrator role. This role would contain users who have permission to administer wikis.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

- Organizations are hierarchical collections of users. Users can be members of one or many of them, up and down the hierarchy. Membership in organizations gives users access to the pages of that organization. If you picture a hierarchical structure that represents a company called Inkwell, a user might belong to Inkwell, Sales Department, Mid-Atlantic Region. This would not only denote that employee's position in the company, but it would also give that employee access to the content he or she needs to do his or her job.
- Communities are ad hoc collections of users. Users can join and leave communities, and membership in communities gives them access to the pages in the communities of which they're members. You might have a community called Photography. Users of your site could join this community to share pictures.
- User Groups are defined by portal administrators. They can be used to collect users for purposes that tend to cut across the portal. For example, you might want to grant some users the ability to create a blog on your site. You would then create a User Group called Bloggers and create a page template for them that contains Liferay's Blog portlet. Regardless of these users' membership in other collections (as part of a hierarchy of organizations or as having joined several communities), User Groups provide a separate way of granting specific access to functions that don't depend on membership in other collections or on specific portal permissions.

That's the high-level view of a Liferay portal structure. While this describes a very powerful system for building your web site, it's only the basics. So let's move on to the next level.

1.3.2 Adding content to a collection with pages

Three types of collections can have not only users, but also pages. Pages are, of course, clickable, navigable web pages. Organizations and Communities can have any number of pages defined within them. Pages are organized into *Layouts*, and there are two types of Layouts: Public and Private. So each Organization or Community can have public pages, allowing them to configure a public web site which can be used by members and non-members of the Organization or Community. And they can also have private pages, which are only accessible by the members of the Organization or Community. So you can begin to see how you can build out your site and separate functionality out by whoever is accessing the site.

User Groups don't have pages *per sé*, but rather can have *Page Templates*. These are configured by portal administrators, and become useful for users' personal communities. By default, each user gets a personal community, which itself has public and private layouts. This is a personal web site which the end user can configure (or which can be fairly static—or not exist at all, depending on how you have set up the portal). Portal Administrators can create Page Templates for User Groups. These Page Templates can be populated with the portlets that administrators want users to have. When users are then placed into the User Group, any Page Templates are copied into those users' personal communities. So if, for example, you want certain users to have a Blog, you might create a Blog page with the Blogs portlet on it in a User Group called Bloggers. Any user you add to this user group would have this page copied automatically to his or her personal community, and he or she can begin blogging immediately.

If you haven't already figured it out, a Roles collection has no pages because roles are used solely to define permissions. For example, you could define a role which has permission to view certain pages. This is how roles work together with organizations, communities, and user groups.

Liferay Portal also has the idea of *scope*, the topic of our next discussion.

1.3.3 Configuring a portlet's scope

Scope allows some of the concepts mentioned above to be refined. One user collection that is refined by Scope is Roles. As stated above, Roles are the only collection to which permissions can be attached. So you can create a Role called Wiki Administrator. This role would have permissions to the Wiki portlet, allowing users in this role to create new wikis and add, edit, delete, and move pages. This role can be created under one of two scopes:

- Portal Role
- Community/Organization Role

If you created this role as a Portal Role, then any members of this role would have the defined permissions across the whole portal, in any community or organization. So this would allow users in this role to administer Wikis in whatever communities or organizations they have access to. You can, however, define the role in another

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

way, by scoping it only by community or organization. If the role were defined this way, then users would have the role's permission in only the community or organization in which that role was defined. So scope is very important when it comes to how permissions are defined.

Scope also comes into play with regard to certain of Liferay's built-in portlets. If you go back up to the Dockbar and click *Add > More*, you'll see that the portlets are marked with different icons. These icons tell you something about the portlets with which they are associated. But before we go over what they mean, let's take a look at some portal terminology first.

Sometimes I feel like Dr. Seuss when beginning to discuss this topic:

*If a portlet in a portal on a page in an org,
Has a data-set saved as its own data-store,
And the data would be different for other users' chores,
We call that a non-instanceable portlet!*

And...

*When a portlet in a portal saves its data on the disk,
And the user hits the data based on membership in this,
If the portlet is configured to have its own instances,
We call that an instanceable portlet!*

What I mean by this all has to do with scope.

NON-INSTANCEABLE PORTLETS

Let's stick with the Wiki example. If you place a Wiki portlet on a page, based on what I've described above, where is that page? Yes, you are correct: it's in a Community or Organization. That Wiki now belongs to that community or organization.

I cannot place another Wiki portlet on the same page, because that portlet is what Liferay calls *non-instanceable*. In other words, another instance of that portlet cannot be added to the community or organization: it is *scoped* just for the membership of that community or organization.

I can place another Wiki portlet on a different page in that community or organization, but that Wiki portlet will simply display the same data as the first one. In other words, it will still be the same portlet instance (see figure 1.12).

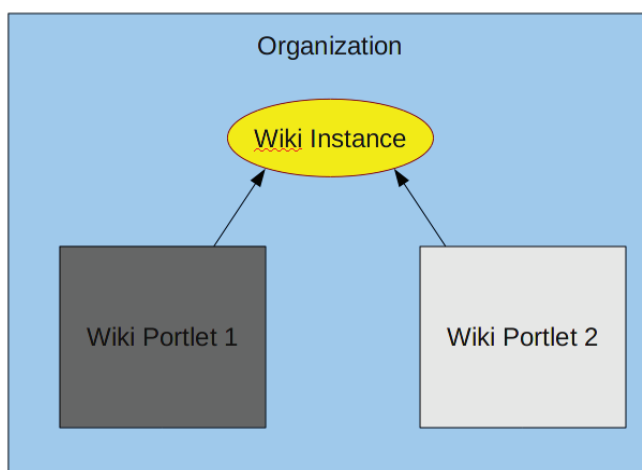


Figure 1.12 A non-instanceable portlet has its data scoped by the community or organization to which it belongs. No matter how many times you add it to a page with the community or organization, it will point to the same data.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

So for non-instanceable portlets, you get one instance of that portlet per community or organization.

This may seem like a limitation, but it is actually a powerful benefit. You can have lots of Wikis on your site, and they can all be kept completely separate from the others. For example, say you are building a web site for Do It Yourself-ers. Your audience likes to build stuff, but the “stuff” they want to build differs wildly. So your site has communities for many different topics, including topics on home renovation all the way to hobby-like topics, like building model rockets or platforms for model railroads. To serve the needs of these users, you might want to give them a Wiki so that they can add helpful tips and articles based on their experiences. But the model rocket group and a home improvement plumbing group are not going to have very much in common (or maybe they will, depending on the size of the rocket—but that's not what we're focusing on right now). So you can give them separate Wikis in their own communities very easily with Liferay. And, of course, because of Liferay's powerful way of collecting users into Communities, all of your users will be members of your site (i.e., the portal), but not necessarily of the same Communities. So they will have the freedom to navigate to the content that is most appropriate for them.

INSTANCEABLE PORTLETS

Other portlets in Liferay are *instanceable*. This means that as many of them can be placed on the same pages in any community or organization as you would like, and they all have their own sets of data. For example, the RSS Portlet is designed to show RSS feeds. You can add as many RSS portlets as you want to any page and configure each portlet to display different feeds, because this portlet is instanceable. The Web Content Display portlet is the same way: you can place as many Web Content Display portlets on a page as you wish, and each portlet can display a different piece of web content. When picking portlets from Liferay's Add > More window, the interface shows which portlets are instanceable and which are non-instanceable, as show in figure 1.13.

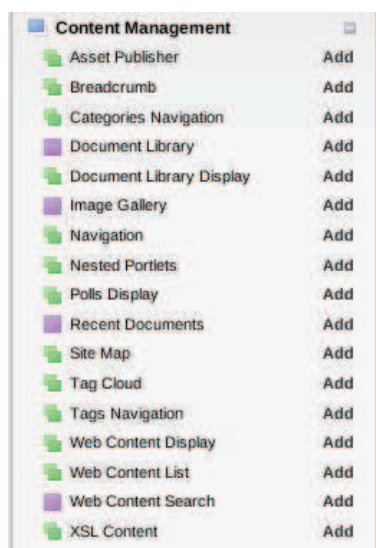


Figure 1.13 Instanceable and non-instanceable portlets in Liferay's Add window. Liferay's UI clearly shows you which portlets can be added to the same page and yet have different data (instanceable) and which cannot (non-instanceable).

You can tell which portlet is which in the user interface by looking at the icons in the Add > More window. If there's a green icon with two windows, the portlet is instanceable. If there's a purple icon with one window, the portlet is non-instanceable.

PAGE SCOPES

Sometimes, however, Liferay's default scopes need to be enhanced with more flexibility. So I'm going to backtrack a bit on what I said above. If you *really* need to have two non-instanceable portlets with different data sets in your community, you can do that. It's just not available by default (and wasn't available at all in older versions of the product). This has to be configured on a per-portlet basis.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Still using the Wiki portlet as an example, if you click the configuration icon in the portlet window (which looks like a wrench in the default theme), a menu will pop open. Click *Configuration* in this menu and all the configuration options for this portlet will be displayed. One of the tabs in this window is called *Scope* (see figure 1.14).

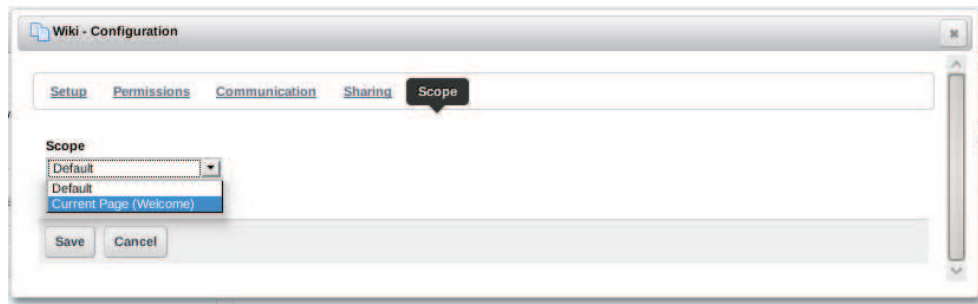


Figure 1.14 Changing the scope in the Wiki portlet.

Here, you can change the scope from the default to the current page. This lets you turn a non-instanceable portlet into a portlet whose instance is tied to the page instead of the community or organization. What this means is that you can add another page to this community or organization and place *another* Wiki portlet on that page. Once that's done, you can set that portlet to have either community/organization scope or page scope. And so on. You won't be able to add multiple Wikis to the same page, but this lets you have multiple non-instanceable portlets per community or organization, provided the portlet supports page scopes.

I don't want to delve too much into these concepts at this stage. You'll be taking advantage of scope soon enough in your code. For now, just let it all sink in, and let's turn to something more concrete: how to navigate around Liferay.

1.4 Getting around in Liferay

Liferay's user interface has a philosophy behind it: get out of the way of the user. For that reason, it hides a lot of power behind what looks like a very simple interface. One of the main UI elements is the Dockbar.

You have already been introduced to some of the functionality of the Dockbar, so let's see what other functions it provides. Figure 1.15 shows the Dockbar in full.



Figure 1.15 Liferay's Dockbar, which appears at the top of every page when a user is logged in.

Let's take each element in order from left to right.

PIN ICON

At the far left is a pin icon, which does what you would expect it to do: It pins the Dockbar to the screen so that no matter how far down you scroll, it stays at the top of the screen. This can be helpful if you're working with long pages and need to use the Dockbar's functionality to add portlets to the bottom of the screen. This is a toggle switch, so you can unpin the Dockbar by clicking the icon again.

Next in the Dockbar is the Add menu.

ADD MENU

You've already seen most of the functionality of this menu for adding applications to the page. It can add pages too. If you click *Add > Page*, a new page will be added next to the page you're on, and a field will appear, allowing you to name the page. There's a much more powerful page administration screen, but this function allows you to quickly add pages to your web site as you're working on it.

The next item in the Dockbar is the Manage menu.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

MANAGE MENU

Use the Manage menu to manage pages, page layouts, and more. This is where you get access to the interface which lets you group your pages in the order you wish—as well as nest them into subpage levels. You can also apply themes to whole layouts or to single pages. The Manage Pages screen is shown in figure 1.16.

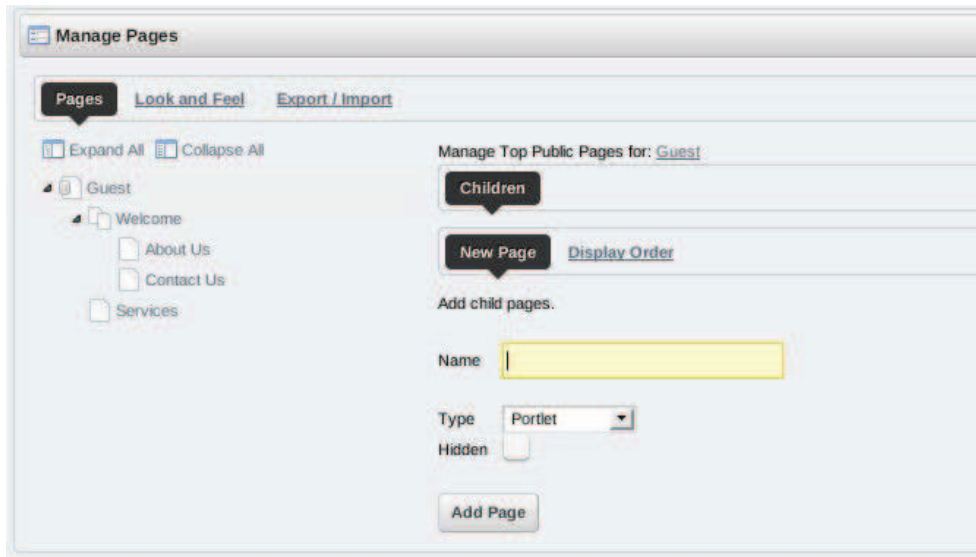


Figure 1.16 The Manage Pages screen allows you to nest your pages, change the display order by dragging and dropping them, change themes, and more.

Perhaps the most important item in the Manage menu, however, is the Control Panel.

Liferay's Control Panel is the central location where just about everything can be administered. If you're going to be administering a Liferay Portal, you'll spend most of your time here. The control panel is very easy to navigate. On the left side is a list of headings with functions underneath them. The headings are in alphabetical order, but the functions are in a logical order. Figure 1.17 shows the Control Panel, which purposefully uses a different theme from the default pages, so you can instantly tell where you are.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

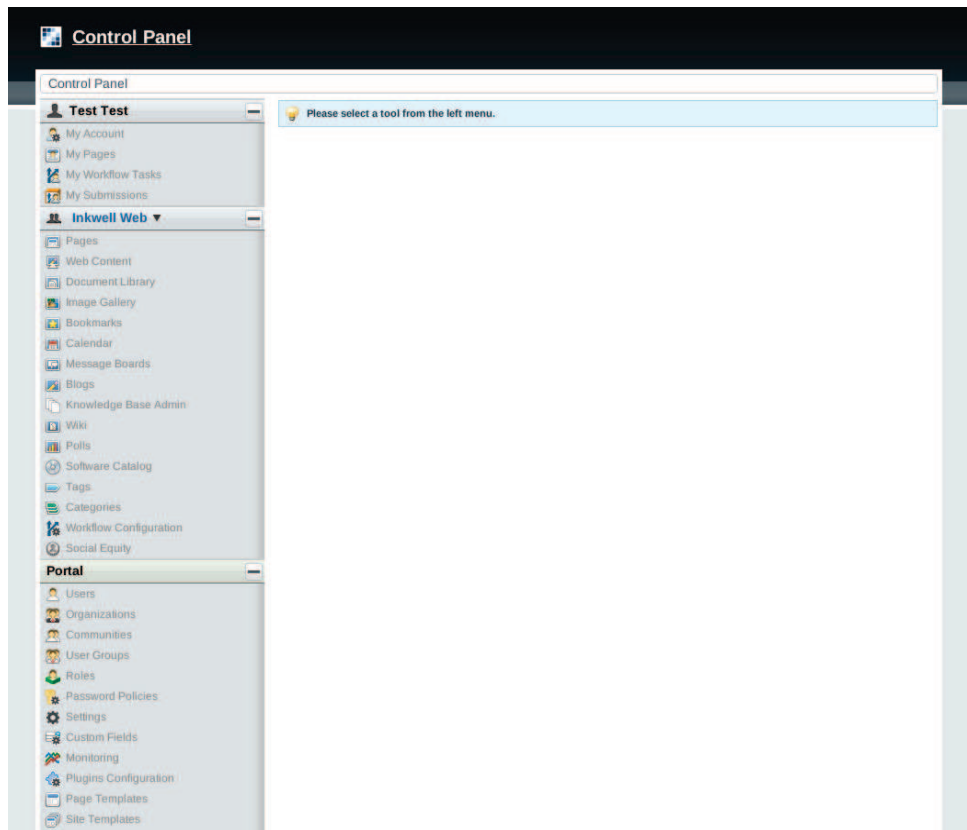


Figure 1.17 Liferay's Control Panel. It's divided into four sections: a section for the current user, a content section, a portal section, and a server section, which isn't visible in the figure above.

- **User Name**—The first heading is named for the logged-in user (Test Test in figure 1.17) and is used to manage the user's personal space. Here, you can change your account information and manage your own personal pages.
- **Content**—The Content section contains links to all of Liferay's content management functions. You can maintain web content, documents, images, bookmarks, and a calendar; administer a message board; configure a wiki; and more. These links are scoped for the particular community from which you navigated to the Control Panel, but this can be changed using the select box.
- **Portal**—The Portal section allows portal administrators to set up and maintain the portal. This is where you can add and edit users, organizations, communities, and roles as well as configure the settings of the portal.
- **Server**—The Server section contains administrative functions for configuring portal instances, plugins, and more.

TOGGLE EDIT CONTROLS

The next function in the Dockbar is not a menu; it's a toggle for the edit controls on the portlets. As an administrator, you get to see some icons in the title bars of the portlets on a page. These correspond roughly to the icons you might see in your operating system. There's an icon for closing a portlet, minimizing it, and for the configuration menu which you have already seen (we used this to change the scope of the Wiki portlet). If you are composing a page and would like to see something that more closely resembles what your users will see, you can use the Toggle Edit Controls link to turn off these controls.

Next, toward the end of the Dockbar, is the Go To menu (shown in figure 1.18).

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Go To MENU

Use the Go To menu to navigate to the various Community and Organization pages to which you have access. Each page name appears, along with its public and private layouts, if they have them.

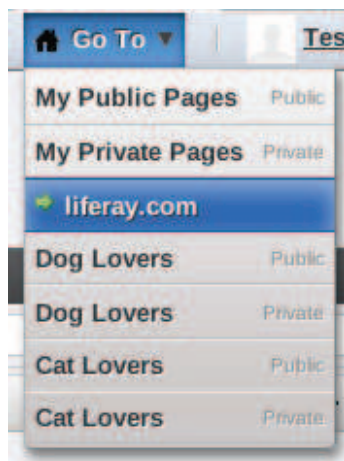


Figure 1.18 The Go To menu displaying public and private layouts for three communities: the default Guest community, Dog Lovers, and Cat Lovers. Notice that the default community only has public pages, so only one link appears.

The final link in the Dockbar will take you to your user account information in the Control Panel.

USER ACCOUNT

The User Account menu item opens a page where you can change your name and email address, upload a profile picture, and maintain all information about you. You can also sign out of the portal from here.

As you can see, Liferay packs a lot of power in a deceptively simple user interface. The intent of this small tour was to give you an idea of where you can go in Liferay and how to get there as you begin to build your site.

Even though we've now touched on several of the constructs that provide you with the building blocks you'll use to build a web site in Liferay, it's sometimes difficult to begin imagining how your site could be built using these building blocks, because many of the concepts are new and unique to Liferay. So let's spend a little time figuring out how you can imagine your site running in Liferay Portal.

1.5 Imagining your site in Liferay

Every successful web site does something unique, or does something in a way that is better than anyone else has done it before. While Liferay has tons of functionality out of the box, much of that functionality is a default implementation of features that are under the hood. What do I mean by that? Let me answer by giving you some examples.

Liferay portal has a collaboration API which contains features allowing users to post discussions, rate items, or tag content. This API has been used to provide everything from the Message Boards portlet to tagging wiki articles, to rating shopping cart items. This book will introduce you to these APIs, so that you can consider what kinds of applications *you* can build with these powerful features.

That, of course, is not the only API we'll cover. You'll also see Liferay's Social API, which gives you the ability to make your applications—indeed, even your entire web site—social. Your users will be able to connect with each other and share content and activities, and even share content and applications on other social networks. Again, the question remains: what will *you* do when given the power to build such applications?

The point of all of this is that when you're finished reading this book, you'll have the ability to make Liferay sing to *your* tune. And because there's so much power in the Liferay platform, you'll get a head start on building your site because the functionality you need is already built into the platform—all you need to do is implement it.

We'll go from the ground up in familiarizing you with Liferay development throughout the course of this book. For now, let's use the information we already have to begin imagining your site from within the Liferay constructs described in the preceding sections. Then later, as you see the full power of Liferay's development platform, you'll

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

see how easy it is to use Liferay as the foundation of your web site, and you can plan how to integrate the features of your applications with the power of the platform.

Portal design is best done by breaking up your site into small chunks and then designing each chunk individually. That way, you don't get overwhelmed by the largeness of your task, and before you know it, breaking it up into smaller chunks has enabled you to design the whole site!

In this section, we'll walk through a design process that is based on a set of forms that I've used with success to design many portals.

Tip For your convenience, the portal design forms can be found in Appendix B. Tear them out or duplicate and then fill them out as you work through this section.

We'll break out the design process into three main portal chunks:

- User Groupings
- Organizations and Communities
- Content

1.5.1 Asking the right questions

The first thing we want to do is figure out how we can get all of your ideas divided up into neat, organized chunks that can then be focused in on in more detail. Ask yourself the following questions:

- Will users be given freedom to sign up on the site?
- Will your user groupings be ad-hoc, static, or both? (If your user groupings will be ad-hoc, you know you'll be creating communities for your users to join and leave.)
- Will some regular users have access to things others won't? (If so, you know you'll be using Roles.)
- Will you be delegating administrative tasks to some users? (If so, you may have Community or Organization Administrators.)

Once you've answered these questions, go ahead and brainstorm the groupings or collections of users you may have.

1.5.2 Defining and categorizing collections

Don't worry about trying to define them as User Groups, Communities, Organizations, or Roles. Start figuring out some groupings. Some examples are anonymous visitors (potential customers), customers, community members, and specific groupings based on your web site. For example, if you're building a web site for do-it-yourselfers, you might come up with categories such as carpentry, plumbing, model rocketry, or even old computers.

At this point, you should have a good list of your groupings. Now combine that list with what you answered to the previous questions. Will any of the groups require pages? If so, you know which ones are Communities or Organizations. Are the groupings associated in any way? If so, how? You're now beginning to identify a possible organization hierarchy.

Are there some groupings that cut across the entire portal (such as a bloggers group)? If so, that's a likely candidate for a User Group, and you can begin thinking about whether these users should have page templates defined for them. Or it may be a good candidate for a Community, if the grouping should have its own set of pages. Once you've categorized your collections of users by Organizations, Communities, User Groups, and Roles, you can begin designing your content.

1.5.3 Designing content

Pages can be part of Organizations or Communities. By default, each can have public pages which everyone can see, as well as private pages which only authenticated members of that Organization or Community can see. Take each Organization and Community you've identified and determine the page hierarchy that will exist for each one. This may even help you to further define your Roles and User Groups.

When you are finished with this process, you should have a nice, high-level design for your web site. You may have something very simple, like Liferay's default: one community called Guest for everyone to use. Or you may

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

have something more complex. The point is, it's a start. From here, we can delve into the custom applications you need to write to make your site unique, as well as the customizations to Liferay that you need to make to satisfy all of your requirements. That's what the rest of the book is all about.

1.6 Summary

Liferay Portal is an ideal choice for building your web site. Using the unique constructs that the platform gives you, you can design a site that can handle any situation you can throw at it. Liferay Portal also offers you an unbeatable platform for building your web applications, as well as a ton of applications that are already implemented, in order to help jump start the creation of your site.

In addition, Liferay Portal frees you from the limitations of the old Java portal standard. As an open source project, it enables you to be as lightweight or as heavyweight as you want to be. And because it provides a multitude of tools and utilities for increasing developer productivity, you'll be able to get your site done faster.

Liferay gives you a powerful paradigm for organizing your users and getting them access to the content they want to see. You can use Communities, Organizations, Roles, and User Groups to make sure that the right content gets to the right people and that restricted content is protected so that only the proper users can view it.

Because Liferay is so easy to use, you can create complex web sites quickly. Because all of the common applications you need to run a web site are already included, it's a simple matter to pick the applications you need and drop them onto your pages. Because no further work is needed to integrate these applications, your time is freed up to focus on the applications you need to build that are unique to your web site.

As we move further into this book, you'll learn how to customize Liferay to make it look the way you want it to look, act the way you want it to act, and host the applications that you design and write. This is going to be an interesting journey for us, and I'm sure you'll find it as rewarding as I have. I hope you'll come along and take the red pill with me—it's going to be an exciting ride.

In the next chapter, we'll get Liferay Portal 6 installed, unpack and configure the Plugins SDK, and dive into creating our first portlet application.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

2

Getting started with the Liferay development platform

This chapter covers

- Installing a Liferay bundle and setting up a database
- Setting up the Plugins SDK
- Generating plugin projects
- Writing your first portlet

Liferay provides you with an extremely powerful development platform which allows you to do everything from providing your own portlet applications to customizing the platform's core functionality. You probably have all kinds of ideas of what you want to do with your web site: what your users' first experience should be, how they will interact, and even mundane things like what the registration process will be like. You have the full ability to define these features any way you want to with Liferay, but you need to understand where and how this is done before you start implementing your site. In this chapter, I want to give you some direction as to where to start and how to proceed.

But first things first: we most definitely need to get Liferay and its development environment installed before we start developing anything on it. We'll spend the first part of the chapter installing Liferay and the Plugins SDK, and then we'll move on to your first projects.

So let's get to it!

2.1 *Installing Liferay and the Plugins SDK*

Liferay Portal is extremely easy to install. Installing the Plugins SDK will take a little bit longer. Regardless, the process is pretty much the same on any operating system. I don't want to get into operating system wars here, though I do, of course, have a preference for what I use every day (doesn't everybody?). So for the purpose of this book, I will be operating system agnostic, as Liferay is, which benefits everybody.

The first thing you need to do before trying to install Liferay Portal is to make sure that you have the Java SDK installed. This is the *JDK*, or the Java Development Kit, not the *JRE*, which is the Java Runtime Environment. Why do you need the JDK? Because you'll be doing *development*, silly.

You need to do more than just install the JDK. On most—if not all—operating systems, the JDK install process does not set up one of the most important things for you: the `JAVA_HOME` environment variable. So you'll have to do this yourself. In order to explain this, I will have to define some rules for operating system agnosticity which are designed to keep everyone happy. This means that not everybody will be happy, of course, but I'm hoping you'll accept the trade of happiness for fairness.

- **Rule 1:** If something is done the same way in all operating systems, I will explain it only once.
- **Rule 2:** File paths will be denoted by forward slashes (/), because more operating systems use that than anything else.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

- **Rule 3:** Operating systems will be presented in alphabetical order. This means that (L)inux, (M)ac, and (U)nix all beat (W)indows. This may make Windows users unhappy, but for fairness, see the next rule.
- **Rule 4:** Operating systems of the same family will be presented together, unless there is some difference between them that requires explanation (there usually isn't). For this reason, I will generally lump Linux, Mac, and Unix together. Sorry guys, you get to go first, but I'm giving you a new name: **LUM**. Why LUM? Well, I only had one vowel to work with, and LUM sounds better than MUL, UML stands for Unified Modeling Language, and LMU is unpronounceable.

We're going to perform this install in three easy steps. First, we'll install the JDK, so we can get the Java runtime that underlies all of the technology upon which we'll be working. Next, we'll install the Liferay/Tomcat bundle. Finally, we'll install Liferay's Software Development Kit, which is called the Plugins SDK.

2.1.1 Installing the Java SDK

Okay; so first install the Java Development Kit. Most LUM operating systems make this really easy, as you can see from table 2.x below. For the rest, you'll have to download it from Oracle and follow the installation instructions.

Table 2.x

Mac	Linux	Unix & Windows
The JDK should be installed by default (that may change)	The JDK is available in your distribution's package manager. Liferay works with both the closed source JDK and the OpenJDK, so you're free to choose whichever one you want.	Download the JDK from Oracle: http://www.oracle.com/technetwork/java/javase/downloads/index.html

Once you have the JDK installed, you need to set that pesky `JAVA_HOME` environment variable.

LUM: This is fairly easy for LUM users. First, you need to know where your JDK is installed, and then you edit a hidden file in your home directory called `.profile` and set the variable to that location. Here's a sample of what that might look like:

```
JAVA_HOME=/usr/lib/jvm/java-6-openjdk
export JAVA_HOME
```

If you are using a non-default shell such as `csh` or `tsh`, you won't need that second line, and you should precede the first with `setenv`. Of course, if you are using `csh` or `tsh`, you probably already know that.

Windows: You'll need to navigate through the Control Panel to set environment variables.

- Windows 7 / Vista users—Select Control Panel > System > Advanced Settings > Advanced tab > Environment Variables (think they hid it well enough?).
- Windows XP users—Select Control Panel > System > Advanced tab > Environment Variables.

Once you get there, the dialog box looks the same on all three versions of Windows (figure 2.1).

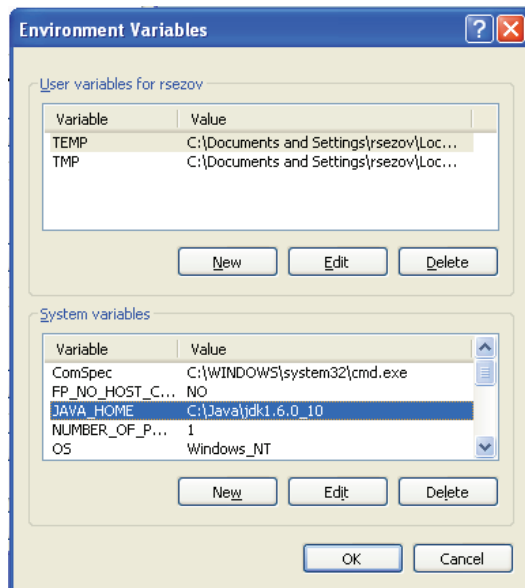


Figure 2.1 Use this dialog box to set environment variables on Windows operating systems.

Under System Variables, click New, set the `JAVA_HOME` variable to the location where you installed the JDK, and then click OK.

You've now got a Java Development Kit installed. You can now begin all kinds of development using Java, but of course we're going to focus on Liferay here. Because of that, the next thing we need to do is get Liferay installed.

2.1.2 Installing a Liferay bundle

Once you have your Java Development Kit all set up, you've made it to the easy part: installing Liferay. This is done in two steps:

- unzip the archive
- edit a text file

Liferay Portal can be installed on a wide variety of application servers and also comes bundled with a number of open source application servers. You can choose among Glassfish, JBoss, Jetty, JOnAS, Resin, or Tomcat, and you should certainly use whichever bundle is right for your environment or organization. If, however, you don't know which one to choose, I recommend using the Liferay-Tomcat 6.0 bundle, as Tomcat is small, fast, and takes up less resources than most other containers. Any supported container is fine, however, so you can use the container that is best for your organization.

Note: We'll be using the Tomcat bundle throughout this book.

Before we copy or unzip anything anywhere, let me recommend a way of keeping all your code organized. I always create a folder which we'll denote in this book as `[Code Home]`. Call this folder whatever you want to call it, and stick it wherever you want to stick it.

Organizing your code on Windows

I recommend that you create the `[Code Home]` folder in the root of your drive. Why? Because Windows' file system, NTFS, has a limitation on the total number of characters that can make up a path. Though you can nest folders as deeply as you want, the entire path can be comprised of no more than 256 characters. This means that if you put your code in a folder like `c:\Documents and Settings\Administrator\Java`, you've already wasted 44 characters out of your total of 256. Because of Java's package naming conventions, you can

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

very quickly create a path that is too deep for Windows to handle if you don't put your source code somewhere near the root of the drive (consider Liferay's package `com.liferay.portal.security.permission.comparator`, which is already inside a folder structure of `portal-impl/src`, as an example).

Now that you have [Code Home] created, create a folder inside that called `bundles`.

Download the latest Liferay-Tomcat bundle and unzip it to [Code Home]/`bundles`. To start Tomcat, navigate to the [Code Home]/`bundles/[bundle home]/tomcat-[version]/bin` folder and run the startup command.

LUM	Windows
<code>./startup.sh</code>	<code>startup.bat</code>

Liferay Portal will start and your browser will automatically launch so that you can view your portal. By default, open source versions of Liferay Portal ship with a sample welcome page and web site already included, for a fictional company called 7cogs, which you can see in figure 2.2. This is a great way for you to click around and see how an actual implementation may work.



Figure 2.2 This is the sample web site that comes included with a Liferay bundle. It showcases many of the things that were described in Chapter 1, giving you a nice example of all of those elements (content management, forums, wiki, and more) all working together to implement a single web site.

You could leave Liferay configured this way, but this is not the most optimal configuration, as your portal is using an embedded database called HSQL to store all of its data. It's far better to use a real database. And you'll also want to make sure you start with a clean database, not one that already has a sample web site in it. So next

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

we're going to take a page from the administration side of things and get your development server connected to a standalone, clean database.

2.2 A crash course in Liferay server administration

If you want to set up Liferay as a real server, you'll be far better served by checking out Liferay's documentation. Because we're only concerned with having a good development environment, we'll concentrate only on connecting your Liferay bundle to a standalone database (hence, the "crash course" instead of the "full course"). This is generally for stability reasons: HSQL is great for demos and stuff like that, but if you're going to start doing development, it's far better to use a standalone database. This configuration more closely mirrors a production configuration, and it provides your data with a bit more stability, as the database is not running in the same process as Liferay. And when debugging, you may find it easier to use the data querying tools that your database vendor provides with your database. Liferay also performs better when connected to a standalone database, because a database server is designed to have multiple connections to it at the same time, rather than running in-process and queuing up requests. And finally, rather than storing the data inside the bundle, the data is stored with your database server, keeping it separate from your Liferay installation.

2.2.1 Setting up a database

Here's the heart of the crash course. What follows is the same exact procedure you would use to set up Liferay as a real server. The only difference is that you'll be doing all of this locally on your machine, while in production you would likely have your database and your Liferay installation on separate machines.

As I had a recommendation for which bundle to use, similarly I recommend that you use MySQL for this purpose, as it is small, free, and very fast. It's also very easily obtained: on Linux, it's available in your package manager. If you're on Mac or Windows, it is easily downloaded and installed.

Again, if you use a different database, there is no reason not to use that database as long as you have the resources to run it. Liferay supports all of the widely-used databases in the industry today.

To install MySQL and its utilities, you will need four components: *MySQL Server*, *MySQL Query Browser*, and *MySQL Administrator*. The first component is the server itself, which on Windows will get installed as a service. The second component is a database browsing and querying tool, and the third is an administration utility that enables the end user to create databases and user IDs graphically. If you are running Windows or Mac, download these three components from MySQL's web site (<http://www.mysql.com>).

Once you have a running MySQL server, you'll want to do two things: set the root password and create your database. By default, MySQL does not have an administrative (root) password set. You should definitely set one. To do this, drop to a command line and issue the following command:

```
mysqladmin -u root password NEWPASSWORD
```

Instead of NEWPASSWORD, you would, of course, type the password that you want (maybe 1337h4x0r). Next, you can start the MySQL command line utility via the following command:

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

Once you launch it, it will display some messages and then a MySQL prompt:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 119
Server version: 5.1.37-lubuntu5 (Ubuntu)
```

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
mysql>
```

At the command prompt, type the following command:

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

MySQL should return the following message:

```
Query OK, 1 row affected (0.12 sec)
```

You'll be back at the MySQL prompt. You can type `quit` and press enter, and you'll return to your operating system's command prompt.

Note that on some Linux distributions MySQL is configured so that it will not listen on the network for connections. This is done for security reasons, but it prevents Java from being able to connect to MySQL via the JDBC driver. To fix this, search for your `my.cnf` file (it is probably in `/etc/`, `/etc/mysql` or `/etc/sysconfig`). There are two ways in which this may be disabled. If you find a directive called `skip-networking`, comment it by

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

putting a hash mark (#) in front of it. If you find a directive called `bind-address` and it is configured to bind only to localhost (127.0.0.1), comment it out by putting a hash mark (#) in front of it. Save the file and then restart MySQL.

Liferay defines the folder it resides in as *Liferay Home*. The home folder is very important to the operation of Liferay. This folder is the top level folder which you extracted from the .zip file. Liferay creates certain resources that it needs in this folder. It contains some folders (`data`, `deploy`, and if you're using Liferay Enterprise Edition, `license`), and there is a configuration file called `portal-ext.properties` which you can place here to change some of the Liferay configuration. We'll edit this file to connect Liferay to your database.

If you're using Liferay Community Edition, before you connect Liferay to your database, you'll need to remove the sample web site so that you can start with a fresh, clean installation like the one you saw in Chapter 1. This is extremely easy to do: all you have to do is undeploy the plugin that creates the sample site.

2.2.2 Removing the sample web site

To undeploy an application in Tomcat, all you need to do is navigate to the folder where the applications are stored and delete the folder that contains the application.

Note: If you're using Liferay Enterprise Edition, you can skip this step.

In a Liferay-Tomcat bundle, Tomcat is located in `[Liferay Home]/tomcat-[version number]`. Inside of this folder is a folder called `webapps`, which is where Tomcat stores the applications that are installed. Go into this folder and you will see a list of folders containing Liferay and various plugins.

The one you want to delete is called **sevencogs-hook**. Remove this folder by either deleting it or moving it to another location on your system. That's all you need to do to prevent the sample web site from being created when Liferay first starts. Make sure that you do this any time you're setting up Liferay for development or as a real server, as you always want to start with a clean database. And speaking of databases, now that we've removed the sample web site, we're ready to connect Liferay to our MySQL database.

2.2.3 Connecting Liferay to the SQL database

To point your Liferay bundle to your database, create a file called `portal-ext.properties` in your Liferay Home folder. This file overrides default properties that come with Liferay. You are going to override the default configuration which points Liferay to the embedded HSQL database.

To connect your installation of Liferay Portal to your database, add the appropriate template for your particular database to your newly created `portal-ext.properties` file. The template for MySQL is provided as an example below.

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

You would provide the user name and password to the database as values for the `username` and `password` directives.

Note If you're using a different database, you'll find templates for the other databases in Liferay's documentation.

Save the file. You can now start your application server.

If you're JDBC-savvy, you'll notice one thing is missing from the instructions above: the JDBC driver for MySQL. Liferay includes the MySQL driver for convenience, so you don't have to worry about downloading it and making it available to your Liferay bundle. You'll find it in the Tomcat bundle in `[Tomcat Home]/lib/ext/mysql.jar`. If you are using a different database, you'll need to copy your database's JDBC driver to this folder before starting Liferay.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Note For a production machine, you would also generally connect Liferay to a mail server so that it can send mail notifications. Since on a developer machine it is likely that there is no mail server running, this step is not necessary.

Now that your Liferay bundle is set up, we can move on to installing the Plugins SDK, which you'll use for creating Liferay development projects.

2.3 Setting up the Plugins SDK

Building portlet and theme projects in the Plugins SDK requires that you have a utility from Apache called Ant installed on your machine. This utility is a scriptable build tool, and is what the Plugins SDK uses to provide its IDE-agnosticity. After you install Ant, you'll be able to install the Plugins SDK.

2.3.1 Installing Ant

If you already have Ant installed or if it's built into the IDE you will be using (it is on every IDE I can think of), you can skip this step.

Download the latest version of Ant from <http://ant.apache.org>, and uncompress the archive into an appropriate folder of your choosing.

Next, you'll have to set two more environment variables like you did for the Java SDK.

- `ANT_HOME` needs to point to the folder to which you installed Ant.
- `ANT_OPTS` contains the proper memory settings for building projects.

After you set these variables, there's one more step, because you'll likely want to be able to run this from the command line—especially if you're the command-prompt-plus-text-editor type of developer. This step adds the Ant binary to your `PATH`, so that you can call it from anywhere on the command line. You can use the `ANT_HOME` environment variable as a quick way to add the binaries for Ant to your `PATH`.

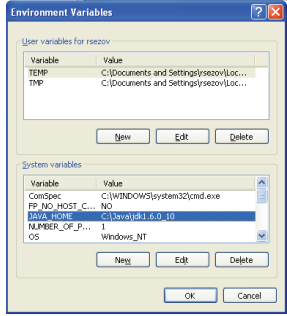
LUM

Step	Action	Result
1	Modify your <code>.profile</code> file	(assuming you installed Ant in <code>/java</code>): <pre>ANT_HOME=/java/apache-ant-1.8.2 ANT_OPTS="-Xms256M -Xmx512M" PATH=\$PATH:\$ANT_HOME/bin export ANT_HOME ANT_OPTS PATH</pre>
2	Log out and log back in	New settings now take effect.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Licensed to fortiss GmbH <ziaie@fortiss.org>

Windows

Step	Action	Result
1	Return to the Environment Variables dialog you used to set JAVA_HOME.	
2	Under System Variables, select <i>New</i> .	
3	Set Variable Name to ANT_HOME. Set Variable Value to the path to which you installed Ant (e.g., c:\java\apache-ant-1.8.2). Click OK.	
4	Select <i>New</i> again, then scroll down until you find the PATH environment variable. Select it and select <i>Edit</i> .	
5	Add %ANT_HOME%\bin to the end or beginning of the path.	
6	Select OK, and then select OK again.	

On any operating system, open a command prompt and type `ant` and press Enter. If Ant attempts to run and you get a “build file not found” error, you have correctly installed Ant. If not, check your environment variable settings and make sure they are pointing to the directory to which you unzipped Ant.

2.3.2 Installing the Plugins SDK

Now you're ready to install the Plugins SDK. This is even easier than installing Liferay.

- Go to your [Code Home] folder.
You should already have a folder in here called `bundles`, where you installed Liferay.
- Download or otherwise obtain the Plugins SDK from Liferay (<http://www.liferay.com/downloads/liferay-portal/additional-files>).
- Unzip the file to [Code Home]/plugins.

That's it! It's installed.

The Plugins SDK contains many subfolders. The two you'll be working in mostly are the `portlets`, `themes`, and `hook` folders. It is here that you will place your portlet and theme plugin projects.

2.3.3 Configuring the Plugins SDK

You will notice that the Plugins SDK contains a file called `build.properties`. This file contains the settings for where you have Liferay installed and where your deployment folder is going to be. You don't want to customize this file. Open the `build.properties` file in a text editor. At the top of the file is a message, “DO NOT EDIT THIS FILE” as shown in figure 2.3.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

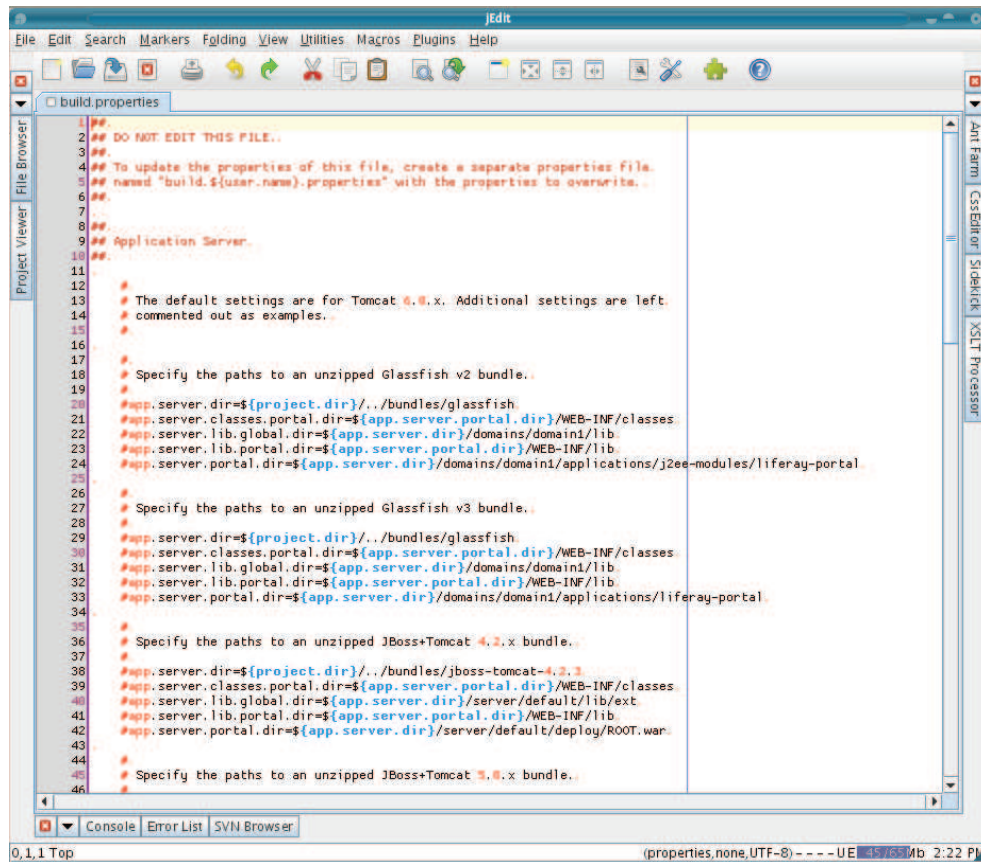


Figure 2.3 Do not edit the `build.properties` file that comes with the Plugins SDK. Override it with your own instead.

Create a new file in the same folder called `build.[username].properties` where `[username]` is your user ID on your machine. This is the ID you log in with every day. For example, if your user name is `cooldude`, you would create a file called `build.cooldude.properties`.

If you're using the default setup I've described so far, you won't have to do very much to configure the Plugins SDK.

- Open your `build.[username].properties` file.
- Add the following line:
`app.server.dir=[full path to Liferay bundle install]`

where `[full path to Liferay bundle install]` is the path to Tomcat in your Liferay bundle.

For example, a Liferay bundle extracts into a folder of its own. Inside that folder is a folder for Tomcat. Use the full path to Tomcat for this directive: this is the `server` directory. This tells the scripts in the Plugins SDK where everything else is relative to your Liferay install, because all paths (such as the deploy path) are defined relative to the server directory.

- Save and close the file.
- You're ready to begin creating projects.

2.3.4 Configuring a non-Tomcat application server

If you downloaded a Liferay bundle other than the Tomcat bundle, you'll need to customize the properties relating to your Liferay bundle. Simply copy the section for your application server of choice out of the

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

`build.properties` file and paste that section into your `build.[username].properties` file, and then comment the section back in.

If you're not using a bundle, but have installed Liferay manually on a proprietary application server, you will have to customize the properties listed in table 2.1.

Table 2.1 `build.properties` for application servers

Property	Purpose
<code>app.server.dir</code>	This is the folder into which you have installed your development version of Liferay. It is a best practice to put this in a folder called <i>bundles</i> next to the <i>plugins</i> folder. This is the default configuration.
<code>app.server.classes.portal.dir</code>	This folder defines where the WEB-INF/classes folder in the Liferay installation can be found.
<code>app.server.lib.global.dir</code>	This folder is where .jars that should be on the global class path can be copied.
<code>app.server.lib.portal.dir</code>	This folder defines where the WEB-INF/lib folder in the Liferay installation can be found.
<code>app.server.portal.dir</code>	This folder defines where the Liferay installation can be found. If you use Eclipse as your development environment, you may also find it convenient to modify the following property:
<code>java.compiler</code>	The default value for this is the standard compiler that comes with Java, <i>modern</i> . You can also use the Eclipse compiler, ECJ. ECJ is an alternate Java compiler with fast performance that allows you also to replace code in memory at run time (called <i>hot code replace</i>). If you set this option to use ECJ, the ant script will install it for you by copying <i>ecj.jar</i> to your Ant folder. This may or may not work with the version of Ant that runs from within various integrated development environments.

After you save the file, you should have a directory structure that looks like this:

```
[Code Home]/bundles/[Liferay Bundle]
[Code Home]/plugins
```

You are now ready to start developing for Liferay, because you've set up a Liferay bundle and connected a Plugins SDK to it. Congratulations!

Let's dive right in and create a portlet plugin.

Ext Plugin caveat

If you've been using Liferay's platform for a while, you might be considering using the Ext plugin for most, if not all, development. If I may give you a piece of advice for right now: don't. With Liferay 6 (and to some extent, 5.2), the Ext plugin is recommended only in a few of the rarest scenarios. For a more thorough discussion of this topic, see Chapter 9.

2.4 Developing a portlet plugin

By now, you're probably saying something like, "Plugins, plugins plugins! All you can talk about is plugins! I have one simple question: what the heck is a plugin!?" Hopefully, that's all you're saying, and you're not following it up with some variant of "\$%&@!" Without any further ado, here's your answer:

Plugins are .war files.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Yes, that's all they are. If you have ever done a Java web application, you already know about 75% of what you need to know to write a portlet, theme, layout template, or hook. Plugins (portlets, themes, layout templates, hooks, and web modules) are how one adds functionality to Liferay, and they have many benefits:

- Plugins can be composed of multiple smaller portlet and theme projects. This reduces the complexity of individual projects, allowing developers to more easily divide up project functionality.
- Plugins are completely separate from the Liferay core. Portlet plugins written to the Java standard are deployable on any portlet container.
- Plugins can be hot deployed (i.e., deployed while the server is running) and are available immediately. This prevents any server downtime for deployments.

Enough theory. Let's get down to using the Plugins SDK. In this section, you'll see how to use the Plugins SDK with just a text editor and Ant. If you're an IDE user, please see Appendix A, where you'll find how to set up your projects with Liferay IDE / Studio, Eclipse, or NetBeans.

The Plugins SDK is both a project generator and a location where your projects are stored. You'll create projects in the folders where they belong, and those folders will contain one or multiple projects of that type. You can then open the projects in an IDE or in a text editor in order to work on them.

If you check individual projects into a source code repository, you'll want to check them out into a fully configured Plugins SDK, because the ant scripts in the projects depend on ant script within the Plugins SDK for their functionality. Let's see how we would create new projects.

2.4.1 Creating a portlet plugin: Hello World

Creating portlet plugins with the Plugins SDK is really easy. As noted before, there is a *portlets* folder inside the plugins SDK folder. This is where your portlet projects will reside. To create a new portlet, first decide what its name is going to be. You need both a project name (without spaces) and a display name (which can have spaces). When you have decided on your portlet's name, you are ready to create the project.

From the *portlets* folder, enter the following command:

LUM:

```
./create.sh <project name> "<portlet title>"
```

Windows:

[???

As a first exercise, let's create the classic example, which, of course, is Hello World. Create a portlet with a project folder name of *hello-world* and a portlet title of *Hello World*:

LUM:

```
./create.sh hello-world "Hello World"
```

Windows:

```
create.bat hello-world "Hello World"
```

You should get a BUILD SUCCESSFUL message from Ant, and there will now be a new folder inside the *portlets* folder in your Plugins SDK.

This *hello-world* folder is your new portlet project. This is where you will be implementing your own functionality.

At this point, if you wish, you can check this project or your whole Plugins SDK into a source code repository in order to share your project with others. And of course, if you're the command-line-plus-text-editor type of developer, you can get to work right away.

Alternatively, you can open your newly created portlet project in your IDE of choice and work with it there. If you do this, you may need to make sure the project references some .jar files from your Liferay installation, or you may get compile errors. Since the ant scripts in the Plugins SDK do this for you automatically, you don't get these errors when working with the Plugins SDK. Appendix A shows you how to do this specifically for Eclipse and NetBeans, but this information applies to all IDEs. Liferay IDE is also covered there, and it makes this process easy and painless.

Regardless of what tool you've used to add functionality to the plugin, when you've implemented some functionality, you'll want to deploy your plugin in order to test it.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

2.4.2 Deploying the Hello World plugin

You can actually deploy this portlet to Liferay right now if you wanted to, because it's already a Hello World portlet (how easy was that?).

To deploy the portlet, go into its directory at the command prompt and type the following command (on all operating systems):

```
ant deploy
```

The portlet will be compiled and deployed to your running Liferay server (you do have Liferay Portal running, right?).

If you are watching the logs, you'll see status messages which look like this:

```
04:07:41,558 INFO [AutoDeployDir:176] Processing hello-world-portlet-6.0.0.1.war
04:07:41,560 INFO [PortletAutoDeployListener:81] Copying portlets for
/home/me/code/bundles/deploy/hello-world-portlet-6.0.0.1.war
...
04:07:43,263 INFO [PortletAutoDeployListener:91] Portlets for
/home/me/code/bundles/deploy/hello-world-portlet-6.0.0.1.war copied successfully. Deployment
will start in a few seconds.
04:07:44,827 INFO [PortletHotDeployListener:250] Registering portlets for hello-world-portlet
04:07:46,729 INFO [PluginPackageUtil:1391] Finished checking for available updates in 5092 ms
04:07:48,839 INFO [PortletHotDeployListener:376] 1 portlet for hello-world-portlet is
available for use
```

When you see the "available for use" message, your plugin is deployed and can be used immediately. This applies for all plugin types, except for the Ext plugin.

To add the plugin to a page:

1. Log in to Liferay Portal using the administrative credentials (user name: *test@liferay.com*; password: *test*).
2. From the Dockbar, select *Add > More*.

Your generated portlet project by default is placed in the *Samples* category (we'll see how to change this later).

3. Open the Samples category.

You should see your portlet displayed:



Figure 2.4 Your Hello World portlet is deployed. Notice that there are two Hello World portlets displayed, with different icons. The grayed out one is already on the page we're browsing, and is unavailable because it's non-instanceable.

By default, portlets are generated as instanceable portlets, so you'll see your portlet appear with the green icon.

4. Drag the portlet out onto the page. It should look like:

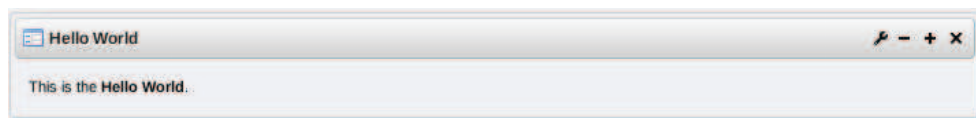


Figure 2.5 The Hello World portlet. This will definitely not win any awards, but it was certainly easy to create.

Obviously, it is very easy to create a Hello World portlet: we didn't even have to write any code. So let's make our portlet actually do something.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

2.5 Making Hello World into Hello You

Hello World examples are all over the Internet, and they are used to introduce topics all the time. And maybe I'm just a rebel at heart, but I can't stand them. In my experience, Hello world examples are not generally all that helpful, as they don't tend to actually introduce anything useful. So we'll try to add a little functionality using the Portlet API for our first example. We'll call this the "Hello You" portlet. This portlet will display a standard "Hello" message when it is first rendered in View Mode.

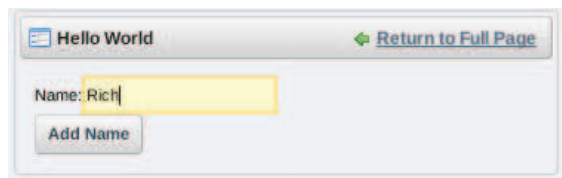


Figure 2.6 The Hello You portlet will allow you to use the portlet's Edit Mode to enter your name, and will then use that name when displaying its "hello" message.

The functionality we're going to add is in edit mode: we'll allow a user to enter his or her name. This name will be stored using the Portlet API as a portlet preference, similar to the way a Weather portlet developer would store a zip code in order to display the proper weather map. We will then allow the portlet to display the user's name in the "Hello" message when that user logs in.

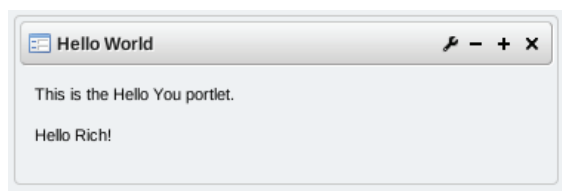


Figure 2.7 Once your name is entered, your name will be saved as a Portlet Preference, and will thereafter always be displayed when you log into the portal. Each preference is linked to the user who is logged in, so if somebody else logs in and sets a different name, that name will be displayed for that user. If you log in again, your name will be displayed. You get all this for free just by being on a portal platform.

This example will introduce to you several features of the Portlet API:

- Portlet Modes
- Portlet Actions and how they are processed
- Portlet Preferences

This will hopefully be a better foundation for building portlets than a simple "Hello World" portlet would provide. There's one additional thing I should mention: as I'm sure you've noticed, I've been throwing around some terms which relate to the Portlet API. If you need a short introduction to the Portlet API (upon which the Liferay development platform is based), you'll find one in Appendix B. This should give you the basic foundation you'll need for this first example and the rest of the chapters, which will build specifically on Liferay as a platform. If you are interested in exploring the Portlet API at a deeper level, I highly recommend Ashish Sarin's book, *Portlets in Action*.

For now, let's get to the portlet.

2.5.1 Anatomy of a portlet project

A portlet project is made up at a minimum of three components:

- Java Source
- Configuration files
- Client-side files (*.jsp, *.css, *.js, graphics, etc.)

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

These files are stored in a standard directory structure which looks like figure 2.8. The example is a fully deployable portlet which can be deployed to your configured Liferay server by running the `deploy` ant task. In fact, you already did this in the previous chapter.

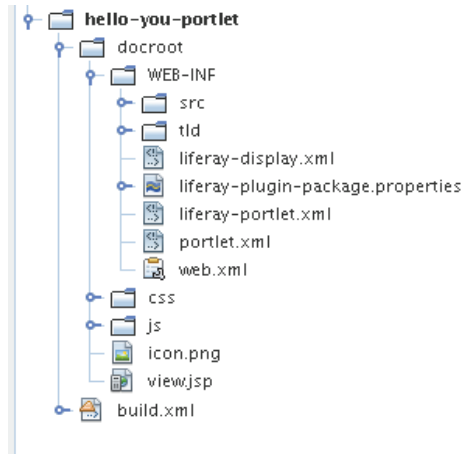


Figure 2.8 This is the folder structure of the Hello You portlet (or any portlet project, for that matter). As you can see, it is very simple to follow and there is a place to put all of your components.

The various files and folders have different functions, listed in Table 2.2.

Table 2.2 Folders of a portlet project

Folder	Description
docroot	This folder is the “root” directory of your application. As you can see, it has several subdirectories.
WEB-INF	This is the standard WEB-INF folder for web modules. Since portlets are web modules too, we will put our configuration files here. You can see that the directory contains several configuration files already. We will go over these separately.
WEB-INF/src	This folder contains the source code for the portlet.
build.xml	This is the Ant build script which you will use to compile and deploy your project.

The default portlet is configured as a standard Java portlet which uses separate JSPs for its three portlet modes (view, edit, and help). Only the `view.jsp` is implemented in the generated portlet; the code will need to be customized to enable the other modes. For Hello You, we'll only implement Edit mode.

In addition to the standard portlet configuration files, the Plugins SDK generates a project which contains some Liferay-specific configuration files:

Table 2.3 Liferay-specific project configuration files

Configuration file	Purpose
<code>liferay-display.xml</code>	This file describes for Liferay what category the portlet should appear under in the <i>Add > More</i> window.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

<code>liferay-portlet.xml</code>	This file describes some optional Liferay-specific enhancements for Java portlets that are installed on a Liferay Portal server. For example, you can set whether a portlet is instanceable, which means that you can place more than one instance on a page, and each portlet will have its own data. The DTD for this file explains all of the possible settings.
<code>liferay-plugin-package.properties</code>	This file describes the plugin to Liferay's hot deployer. One of the things that can be configured in this file is dependency .jars. If a portlet plugin has dependencies on particular .jar files that already come with Liferay, you can specify them in this file and the hot deployer will modify the .war file on deployment so that those .jars get copied from Liferay into the WEB-INF/lib folder of the .war file. This prevents you from having to include .jar files (such as Spring, Struts, or Hibernate) in your portlet project that are already used by Liferay.

Okay; enough lists of what is possible to do; let's put all of these files, directories, and code together to make something.

Open the Hello World project that you created in the last chapter. We're going to turn this into the Hello You portlet. To do this, we'll create our own portlet class, implement Edit mode in the portlet, and define our first portlet action.

2.5.2 Configuring Hello You

The first thing you should take a look at is the `portlet.xml` file, which you'll find in the `WEB-INF` folder. This is the configuration file for the portlet. You will find a line in there that looks like this:

```
<portlet-class>com.liferay.util.bridges.mvc.MVCPortlet</portlet-class>
```

This defines what Java class implements the portlet. By default, projects are generated to use Liferay's `MVCPortlet`, which we'll get to later. It has certain benefits to the experienced developer, but it also abstracts much of the portlet lifecycle from you, which we don't want to do just yet. So we're going to implement our own portlet class the way the portlet specification recommends so that you can get familiar with the Portlet API first. Once you understand the basic Portlet API, you'll be free later to move past it to the `MVCPortlet` and to other frameworks if you wish.

Since we aren't going to use `MVCPortlet`, we have to change this line to point to the portlet class we are going to create.

1. Modify this line so that it looks like this:

```
<portlet-class>[CA]
    com.liferay.inaction.portlet.HelloYouPortlet[CA]
</portlet-class>
```

Next, we need to tell the portal that our portlet implements edit mode as well as view mode (it assumes view mode; otherwise there would be no point to your portlet).

2. Change the `<supports>` tag in your `portlet.xml` so it reads like this:

```
<supports>
    <mime-type>text/html</mime-type>
    <portlet-mode>view</portlet-mode>
    <portlet-mode>edit</portlet-mode>
</supports>
```

You may have noticed in `portlet.xml` another tag called `<init-param>`. This tag, as you have probably figured out, defines initialization parameters which can be used in your portlet. The default project defines a parameter called `view-jsp` which defines the location of the JSP file that will be used to display the portlet in view

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

mode. This parameter can thus be used in the portlet class to forward processing over to the `view.jsp` file in the project. This worked in the initial portlet because this functionality was implemented already in `MVCPortlet`; now that we're using our own portlet class, we will have to implement it ourselves.

3. Below your definition of your portlet class, add the additional initialization parameter below the existing one like this:

```
<init-param>
    <name>edit-jsp</name>
    <value>/edit.jsp</value>
</init-param>
```

Imagine Darth Vader's voice: "All of our configuration is now complete."

We can now start implementing the logic of the portlet.

1. Create a package in your `src` folder called `com.liferay.inaction.portlet`.
2. In this package, create a Java class called `HelloYouPortlet.java`.

This class should extend the `GenericPortlet` class, which is included with the Portlet API and is available in every portal implementation.

So far, your class should look like the following code.

```
package com.liferay.inaction.portlet;

import javax.portlet.GenericPortlet;

public class HelloYouPortlet extends GenericPortlet {
}
```

We've now got the basic structure in place. Next we'll implement the logic for the Hello You application.

2.5.3 Portlet initialization and implementing view mode

The first thing we want to do is implement the Portlet API's `init()` method to pull the values from our initialization parameters into our portlet class. All we need to do is define two instance variables to hold these values and then implement the method:

```
protected String editJSP;
protected String viewJSP;

public void init() throws PortletException {
    editJSP = getInitParameter("edit-jsp");
    viewJSP = getInitParameter("view-jsp");
}
```

Easy, right? We get access to the `getInitParameter()` method because we're extending an existing class in the Portlet API. So anything that is in those initialization parameters gets pulled into these variables.

Now we can actually implement the default view of our portlet. This is done by implementing a method called `doView()`. As you can imagine, there are similar methods for the other portlet modes, and we will also be implementing `doEdit()`. The functionality for this mode is simple: we want to retrieve a preference that may or may not have been stored for the logged in user. If we don't find one, we'll simply print "Hello!" If we do find one, we will print a message that takes that preference and displays it in the message. So if the user's name is Mortimer Snerd, we'll print "Hello Mortimer Snerd!"

Listing 2.1 The default view of your portlet

```
public void doView(RenderRequest renderRequest,
    RenderResponse renderResponse)
    throws IOException, PortletException {
    PortletPreferences prefs = renderRequest.getPreferences();
    String username = (String)prefs.getValue("name", "no");
    if (username.equalsIgnoreCase("no")) {
        username="";
    }
    renderRequest.setAttribute("userName", username);
    include(viewJSP, renderRequest, renderResponse);
}
```

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

src/com.liferay.inaction.portlet/HelloYouPortlet.java

In this code we first grab the `PortletPreferences` and check to see if the preference is there. The method for getting the preferences wants to know what the preference you're looking for is called as well as a value to return if it couldn't find the preference. So we provide *name* for the preference and *no* for the value to return if the preference wasn't found. We could have just as easily provided an empty string instead of the word *no*, but I think this is cooler, because I certainly couldn't have gotten away with the two extra bytes on my old Timex Sinclair 1000.

If we find a name, we then store it as an attribute in the request. If not, we simply store an empty string. We then forward to the `view.jsp` file which was previously defined in our `portlet.xml` file.

Note that we're using some objects here that are very similar to `HttpServletRequest` and `HttpServletResponse`, both of which you are likely already familiar with. These are very similar to their servlet counterparts in that you can set parameters and attributes, retrieve information about the portlet's environment, and more. In our case, we're retrieving an object called `PortletPreferences` from the portlet instance.

We're also calling a convenience method called `include()`, which looks like listing 2.2.

Listing 2.2 Shortening code with an `include()` convenience method

```
protected void include(
    String path, RenderRequest renderRequest,
    RenderResponse renderResponse)
    throws IOException, PortletException {

    PortletRequestDispatcher portletRequestDispatcher =
        getPortletContext().getRequestDispatcher(path);

    if (portletRequestDispatcher == null) {
        _log.error(path + " is not a valid include");
    } else {
        portletRequestDispatcher.include(
            renderRequest, renderResponse);
    }
}
```

src/com.liferay.inaction.portlet/HelloYouPortlet.java

Just like you can in a servlet-based web application, you can forward request processing to a JSP from a portlet. And just like in a servlet-based web application, you have to chain a whole bunch of methods together in order to accomplish it. To make our code more neat, we can simply create a convenience method called `include()` which chains all those methods together for us so that all we have to do is call this one method (which has a nice, short name) to forward processing to a JSP.

We've included a check here to make sure that the `PortletRequestDispatcher` we get out of the `PortletContext` is not null. If it is null, we log that. To use this method, therefore, we'll also have to enable logging in our portlet. Liferay ships with Apache's Commons Logging classes, which make it easy to add log entries from our portlets. We only need to add an instance variable to the class for our `_log` object:

```
private static Log _log = LogFactoryUtil.getLog(HelloYouPortlet.class);
```

To complete the view mode of this portlet, we now have to implement the JSP to which we are forwarding, which you can see in its entirety in listing 2.3.

Listing 2.3 Implementing the JSP for view mode

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<jsp:useBean id="userName" class="java.lang.String" scope="request"/>    #1

<portlet:defineObjects />

<p>This is the Hello You portlet.</p>
<p>Hello <%=userName %>!!</p>
```

docroot/view.jsp

#1 Gets bean out of request

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

In the first line, we declare the portlet tag library. This again is a part of the Portlet API, and is a standard across all portals. The next line should look familiar: we pull a bean out of the request that we had made available in the `doView()` method of our portlet (#1). There's no difference in how this is done versus doing it with a servlet. The tag in the third line comes from the portlet tag library, which we declared in the first line. The `defineObjects` tag registers several objects with the JSP that may be useful: `renderRequest`, `renderResponse`, and `portletConfig`.

About backward compatibility

The tag library declaration above is the declaration for Portlet 1.0. This has been done for backward compatibility, as we are not using any features in this first portlet that require Portlet 2.0. This portlet, therefore, can be deployed on any Portlet 1.0 container (such as Liferay 4.4.x) or any Portlet 2.0 container (such as Liferay 5.0 and above).

The Portlet API's `RenderRequest` and `RenderResponse` objects correspond to the `HttpServletRequest` and `HttpServletResponse` objects with which you may be familiar, and we've already used these in our portlet class. The `PortletConfig` object corresponds roughly with the `ServletConfig` object you would use in a Servlet, in that it holds data about the portlet and its environment. Because you may need to use these objects in your JSP's display logic, these three variables are defined through the `defineObjects` tag, and it's a good idea to always put this tag at the top of your JSPs. So even though we're not using them in this simple portlet, we're starting off with a good habit of just sticking it in there anyway.

Otherwise, the logic of this JSP is very simple. We pull the bean we stored in the request and print a "Hello" message. If the bean has a value, that value will be printed after the message; if not, nothing will be printed.

At this point, view mode is complete. You can actually deploy the portlet as it is and it will display a hello message. Since we haven't implemented edit mode yet, though, it won't have any functionality, so that's what we'll do next.

2.5.4 Implementing edit mode

Before we jump into the code for edit mode, I need to tell you something about URLs in portlet applications. Most web developers are used to manipulating the URL directly. That is not something you can do in a portlet application. In most cases, portlet URLs are linked with Portlet actions which cause the portlet to do some processing. Portlet actions are defined by using a special portlet URL object called an `ActionURL`. Because a portlet is a fragment of a page that is assembled at run time by the portlet container, developers cannot simply define their own URLs as they can in regular web applications. Instead, they must be created programmatically. This is a bit of a paradigm shift, but it's fairly easy to make the transition. It's important that to know that the contents of URLs must be generated by the portal server at run time. Why? So that there are no conflicts between URLs generated by different portlets.

For example, consider a search portlet that sits on a page. This portlet can search for any uploaded content. Say another portlet is placed on that page by a user. This other portlet contains a list of customers and has a search box at the top.

If developers knew ahead of time that these portlets would be placed on the same page, they could make sure that the URLs to the two separate search functions were different. But they didn't know ahead of time that a portal administrator would do this, and so both search URLs point to `/search` in the application's context. Can you see how there would be a conflict? Which application's search would get called?

For this reason, the Portlet API provides URL objects. Developers can create URL objects programmatically and even set parameters in them. This allows the portlet container to generate the actual URL strings at runtime, preventing any conflicts like the one mentioned above. Since our implementation of edit mode consists of a very simple form that users will fill out and submit, we need a URL for the submit button. For this reason, we will have to create one using the API. So our `doEdit()` method in listing 2.4 reflects this.

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

Listing 2.4 Implementing doEdit() in our portlet

```

public void doEdit(RenderRequest renderRequest,
                  RenderResponse renderResponse)
    throws IOException, PortletException {
    renderResponse.setContentType("text/html");
    PortletURL addName = renderResponse.createActionURL();
    addName.setParameter("addName", "addName");
    renderRequest.setAttribute("addNameUrl", addName.toString());
    include(editJSP, renderRequest, renderResponse);
}
src/com.liferay.inaction.portlet/HelloYouPortlet.java

```

When you add the code above, you'll also have to add the import `javax.portlet.PortletURL`.

More about backward compatibility

The first line of code above—which sets the content type—is only required in the 1.0 (JSR-168) version of the API. It doesn't hurt anything, so we have left it in the method above to maintain backwards compatibility, but if you are using Portlet 2.0 (JSR-286), it's not necessary.

We've created an `ActionURL` and added a parameter to it called *addName*. We then put the URL in the request object so that it may be used in the JSP to which we'll be forwarding the processing.

Let's implement the JSP for edit mode next. Create a file in the `docroot` folder called `edit.jsp`. Remember that we earlier pointed to this file by using an initialization parameter in `portlet.xml`. Listing 2.5 shows the `edit.jsp` file.

Listing 2.5 The JSP for edit mode

```

<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<jsp:useBean class="java.lang.String" id="addNameUrl" scope="request" />
<portlet:defineObjects />

<form
  id = "<portlet:namespace />helloForm"
  action="<%=addNameUrl %>"
  method="post">
  <table>
    <tr>
      <td>Name:</td>
      <td><input type="text" name="username"></td>
    </tr>
  </table>
  <input type="submit" id="nameButton" title="Add Name" value="Add Name">
</form>
docroot/edit.jsp

```

In the code above we first declare the tag library for the Portlet API as we did before, and we then make our URL available to the JSP by using the `useBean` tag to retrieve it from the request object and place it in a variable called `addNameUrl`.

After this is a simple HTML form. Notice that we use a portlet tag called `namespace` in front of our form name. This is done for similar reasons that portlet URLs exist: you could have multiple forms with the same name on the page, and then they would conflict with each other. By using this tag, you allow the portal server to prepend a unique string to the front of your form's name, thereby ensuring that no other portlet will contain a form with the same name as yours.

Note also that the form's action is set to the value of the action name parameter in the `ActionURL` we created in the portlet class and then retrieved as a bean in the JSP. This URL has the parameter `addName` in it, which will be captured by our `processAction` method when this form is submitted so that processing can be directed to the right place.

Speaking of `processAction`, that is now the only missing element in our portlet. This method is part of the standard Portlet API. The `processAction` method is there to provide a place where portlet actions can be

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

processed. A default implementation is provided with the API using annotations (see Appendix C for further information), but we're going to override it for this portlet and provide our own implementation.

Portlet URLs can be of two types: a `RenderURL` or an `ActionURL`. A `RenderURL` simply tells the portlet to render itself again. Whatever parameters have been defined for the portlet at that time take effect, and the portlet re-draws itself. In contrast, an `ActionURL` immediately forwards processing to the `processAction` method, where logic can be in place to determine which action has been taken by the user and then the appropriate processing can occur. We'll implement our own version of this so that we can see how it works, and then later on we'll take advantage of other (better) implementations. Since we have only one action, the logic will be pretty simple, as you can see in listing 2.6.

Listing 2.6 Processing a portlet action

```
String addName = actionRequest.getParameter("addName");
if (addName != null) {
    PortletPreferences prefs = actionRequest.getPreferences();
    prefs.setValue("name", actionRequest.getParameter("username"));
    prefs.store();
    actionResponse.setPortletMode(PortletMode.VIEW);
}
```

src/com.liferay.inaction.portlet/HelloYouPortlet.java

You'll also need to add the following two imports to the top of your class:

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

This code searches for a parameter in the portlet's `ActionRequest` object called `addName`. We created this parameter earlier as part of an `ActionURL` in our `doEdit()` method, which we forwarded to our `edit.jsp` and then used as the action for the form. Simply by having the form's action point to this URL, it directs processing to the four lines of code in the `if` statement above. In larger portlets, this `if` block would be longer (under this implementation), because there would be multiple actions whose names you'd have to be checking in order to determine which action needed to be performed.

In any case, the parameter—called `username`—will be found in the request, because it was a field on the form. The Portlet API is then accessed in order to store a preference for this particular user. The preference is called `name` and we use whatever value was in the parameter. Yes, I know: we're not doing any field validation. Normally you would do that before storing anything, but I wanted to keep this example as simple as possible. This key/value pair will now be stored for that portlet/user combination.

The last thing this code does is set the portlet mode back to view mode. When that is done, `doView()` will be called, and the user will now get the "Hello <name>" message.

To summarize: we now have all of the processing for the portlet's edit mode in place. The `doEdit()` method creates a URL with an action name parameter of `addName`. Processing is then forwarded to the `edit.jsp` file where this parameter is used as the action of a form. The form contains one field, which the user will use to type his or her name. When the form is submitted, the portlet's `processAction` method runs and retrieves the action's `name` parameter, which will have the value `addName`. Checking for this value leads us to code which stores the name the user submitted as a `PortletPreference`. To keep the example simple, we have not implemented any validation on the data submitted.

Rather than look at it piecemeal as we did above, it is sometimes easier to see how things work by showing the whole example. So listing 2.7 shows what the entire portlet should look like. We're only going to be able to do this now, at the beginning of the book, with this simpler example, so take advantage of it and look it over to make sure you understand the portlet's structure as a whole.

Listing 2.7 The complete Hello You portlet

```
package com.liferay.inaction.portlet;

import java.io.IOException;

import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;
import javax.portlet.GenericPortlet;
```

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>

```

import javax.portlet.PortletException;
import javax.portlet.PortletMode;
import javax.portlet.PortletPreferences;
import javax.portlet.PortletRequestDispatcher;
import javax.portlet.PortletURL;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

/**
 * The Hello You portlet, a simple example demonstrating
 * Portlet Modes, Portlet Actions, and Portlet Preferences.
 *
 * @author Rich Sezov
 */
public class HelloYouPortlet extends GenericPortlet {

    public void init() throws PortletException {
        editJSP = getInitParameter("edit-jsp");
        viewJSP = getInitParameter("view-jsp");
    }

    public void doEdit(RenderRequest renderRequest,
        RenderResponse renderResponse)
        throws IOException, PortletException {
        renderResponse.setContentType("text/html");
        PortletURL addName = renderResponse.createActionURL();
        addName.setParameter("addName", "addName");
        renderRequest.setAttribute("addNameUrl", addName.toString());
        include(editJSP, renderRequest, renderResponse);
    }

    public void doView(RenderRequest renderRequest,
        RenderResponse renderResponse)
        throws IOException, PortletException {
        PortletPreferences prefs = renderRequest.getPreferences();
        String username = (String) prefs.getValue("name", "no");
        if (username.equalsIgnoreCase("no")) {
            username = "";
        }
        renderRequest.setAttribute("userName", username);
        include(viewJSP, renderRequest, renderResponse);
    }

    public void processAction(
        ActionRequest actionRequest,
        ActionResponse actionResponse)
        throws IOException, PortletException {
        String addName = actionRequest.getParameter("addName");
        if (addName != null) {
            PortletPreferences prefs =
                actionRequest.getPreferences();
            prefs.setValue(
                "name", actionRequest.getParameter("username"));
            prefs.store();
            actionResponse.setPortletMode(PortletMode.VIEW);
        }
    }

    protected void include(
        String path, RenderRequest renderRequest,
        RenderResponse renderResponse)
        throws IOException, PortletException {

        PortletRequestDispatcher portletRequestDispatcher =
            getPortletContext().getRequestDispatcher(path);
    }
}

```

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>


```

        if (portletRequestDispatcher == null) {
            _log.error(path + " is not a valid include");
        } else {
            portletRequestDispatcher.include(
                renderRequest, renderResponse);
        }
    }

    protected String editJSP;
    protected String viewJSP;

    private static Log _log = LogFactory.getLog(HelloYouPortlet.class);
}
src/com.liferay.inaction.portlet/HelloYouPortlet.java

```

As you can see, there's not a lot of code needed to implement this functionality. In later chapters, however, you'll see how Liferay's platform helps you streamline things even more. For now, congratulations! You've completed writing your first portlet! Let's deploy and test it next.

2.6 Deploying and testing your portlet

When we generated this project with the Plugins SDK, it came with its own Ant script which makes it very easy to build and deploy the project. If your IDE supports Ant, you can deploy the project right from within your IDE. If you're not using an IDE, you can run a simple command to deploy your project.

First, start your Liferay server. Again, if you're using an IDE and have followed the instructions from the previous chapter, you can do this right from within your IDE. Once Liferay has started, keep a window open to Liferay's console so you can watch the deploy take place.

Next, run the `deploy` task from the Ant build script. To do this from the command line (on any operating system), issue the following command from the project folder:

```
ant deploy
```

Once you receive the BUILD SUCCESSFUL message, turn your attention to the Liferay console. You should see your portlet deploy right away.

Your new portlet is now indistinguishable from any other portlet deployed in Liferay. Go to `http://localhost:8080` and log in using the administrative credentials:

User Name: test@liferay.com

Password: test

Go up to the Dockbar and click *Add > More*. You'll see many categories of portlets displayed. Open the one labeled *Sample*. The Hello World portlet is listed there. Drag it off the list and drop it in the right-most column, if it's not there already. If you deployed the portlet right after you generated it, the portlet should already be there.

You will now see your portlet displayed. You can close the *Add > More* window by clicking the red X in its top right corner.

The default message of "Hello!" is being displayed in the portlet. This is the way it's supposed to function: you haven't set your Portlet Preference yet, so the portlet does not know your name. To get to Edit Mode, click the button in the title bar of the portlet that has a wrench on it, and then click the *Preferences* link (Liferay Portal displays a portlet's edit mode as a Preferences menu item). You will be brought to the portlet's edit mode, and your `edit.jsp` will be displayed.

Type your name and click the *Add Name* button. Your Portlet Preference will be stored, and because you changed the mode of the portlet back to View in your `processAction` method, the portlet will redisplay itself in view mode. Because your Portlet Preference is now set, the portlet will display the name you entered.

Congratulations! You have just written your first portlet!

2.6.1 Changing the portlet's category and name

Notice that when you added the portlet, you had to select it from the *Sample* category in the *Add > More* window. You also had to add the "Hello World" portlet, but now our portlet is called "Hello You." The portlet is in a suboptimal category because the generated portlet project defaults to this category, and the portlet has the wrong name because we created "Hello World" early in the chapter. So let me show you how to create your own category

Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=642>