Table of Contents

Preface

Evolution of this book

Who should read this book?

What is not covered in this book?

Conventions

About the authors

Updates

Publisher's Note

System Requirements

1. Introduction

Definition of a Portal

What are portlets?

Portlet Specification (JSR286)

Why Liferay and what makes it special?

2. Setup

Prerequisites for getting Liferay up and running

Create a prepackaged bundle and store on the cloud

a. Installing JDK

b. Installing MySQL

c. Installing Ant

Getting Liferay up and running

a. Unzip the liferay tomcat bundle

Once the prerequisite software is setup, we are ready to start installing the liferay tomcat bundle. To do this, follow these steps:

a) Download a fresh copy of Liferay Portal bundled with Tomcat from the Liferay website at http://www.liferay.com/downloads/

b) Unzip this to a folder of your choice.

c) We will call this folder <Liferay-Home>

b. Remove all default plugins

a) Delete all folders under <Liferay-Home>/tomcat6.x.x/webapps/, except for the folder ROOT and tunnel-web.

b) Navigate to <Liferay-Home>/tomcat6.x.x/bin/.

c) Double-click on startup.bat(on a windows machine), or run the following command "sh startup.sh" (on a *nix based machine).

d) Open a browser of your choice, and enter the url "http:localhost:8080".

c. Make Liferay work with a production ready database

a) We need to setup a database now.

b) Open the MySQL Query Browser, (or a command terminal) and log in to the MySQL server.

c) Create a database named "lportal".

d. Starting up the Liferay Server

a) Navigate to <Liferay-Home>/tomcat6.x.x/bin.

b) Double click on the file startup.bat (on a Windows machine), or open a command terminal and type "sh startup.sh" (on a *nix machine). This will startup Liferay.

c) Open a browser of your choice, and enter the following url: http://localhost:8080.

Liferay Plugins SDK

a. Installation

b. Configuration

c. Create simple portlets

Development Tools

a. Installing the latest version of Eclipse IDE

b. Installing Liferay Eclipse IDE

c. Creating a simple portlet using Liferay IDE

d. Anatomy of a portlet

Supporting Tools

a. MySQL Query Browser

b. Mozilla Firefox and Firebug

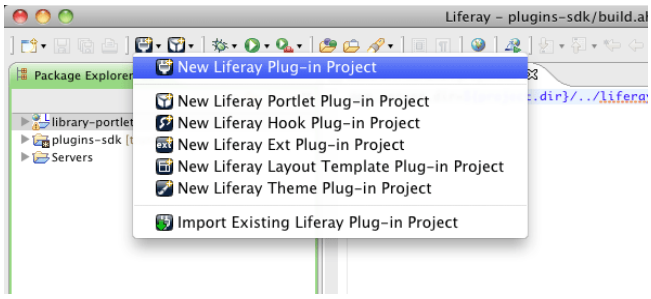
c. An Unzip utility – WinRAR / WinZip

3. Library Management System

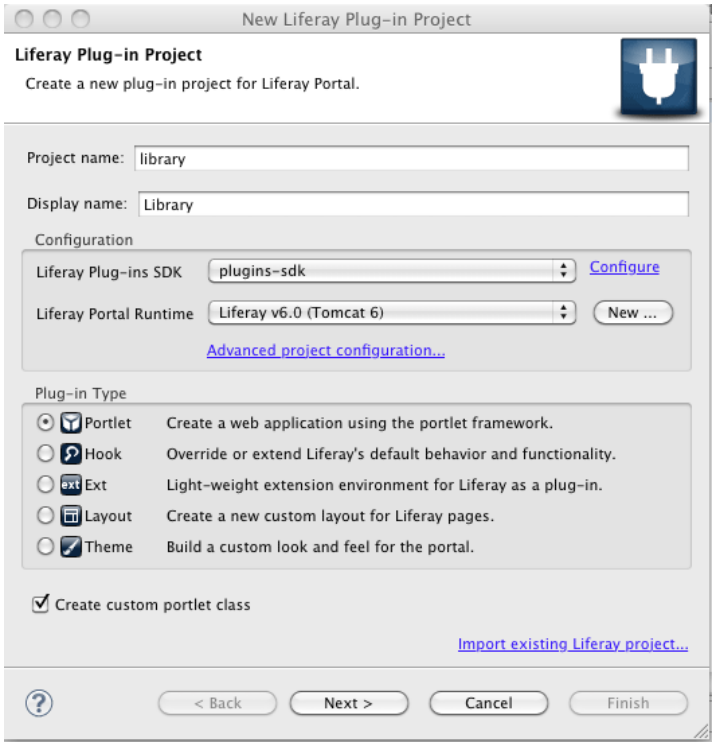
In this chapter we will see how to put the foundation for a hypothetical "Library Management System" which we are going to build through-out this book. In every chapter that is going to follow we will keep adding a new feature, improving our LMS. Let us start with the liferay eclipse IDE to create the basic portlet for LMS.

Create a new Liferay Plug-in Project

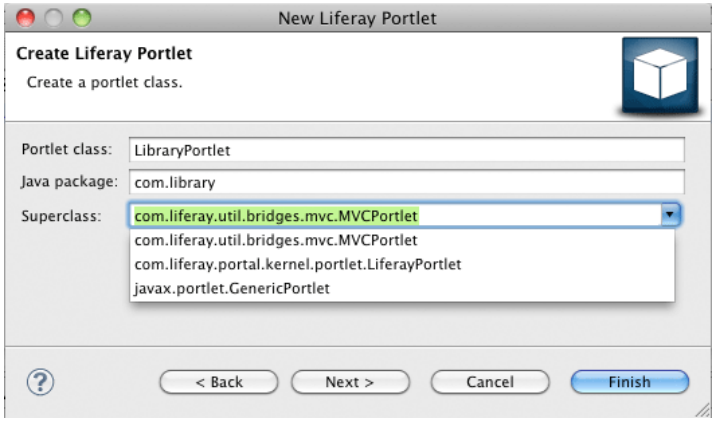
Click on the option "New Liferay Plug-in Project" as shown.



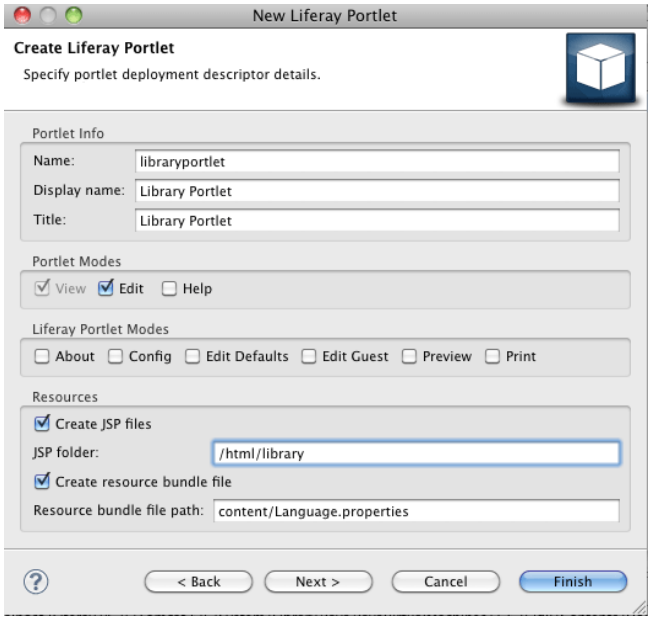
The dialog will open,



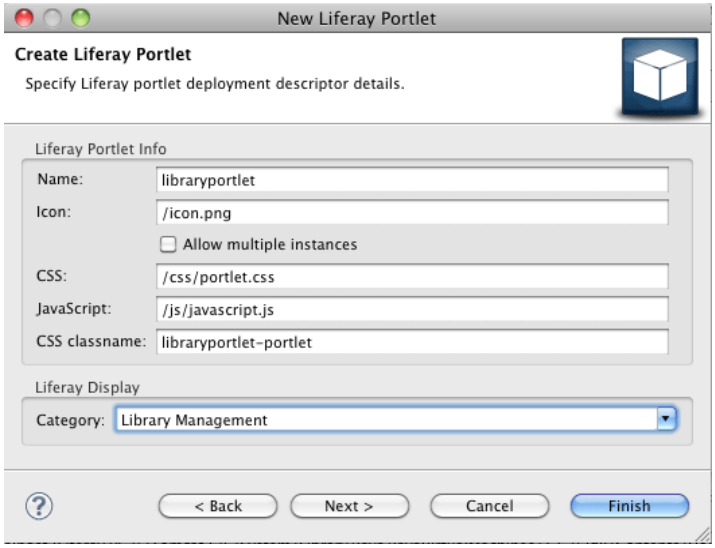
1. give the project name as “library” (all small letters)
2. make sure the plug-ins SDK and Liferay Portal Runtime are configured properly
3. check “Create custom portlet class”
4. click “Next”



1. change the portlet class to “LibraryPortlet”
2. java package as “com.library”
3. select superclass of this portlet as “com.liferay.util.bridges.mvc.MVCPortlet”
4. click “Next”



1. modify Display name and Title to have a space between the words Library and Portlet
2. check “Edit” portlet mode
3. modify JSP folder to “/html/library”
4. check the option “Create resource bundle file”
5. click “Next”

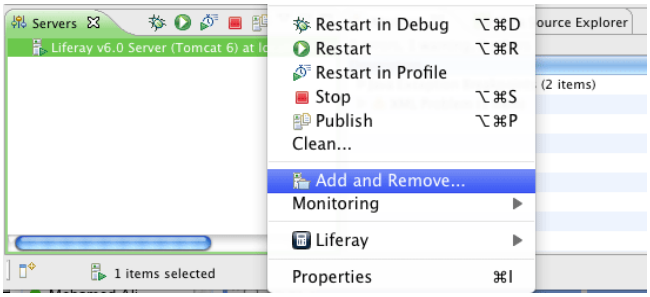


1. uncheck “Allow multiple instances” to make this portlet “non-instansable”
2. specify a new Category - “Library Management”
3. click “Finish”

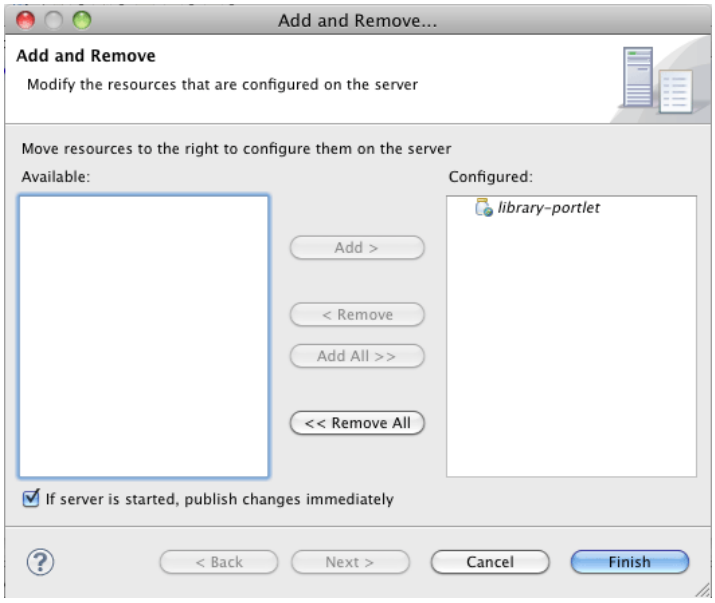
Now you see a new eclipse project with name “library-portlet”.

Deploying “library-portlet” to the server

1. right click on the server “Liferay v6.0”
2. you will see the options
3. click “Add and Remove...”



In the next dialoy, move “library-portlet” from left to right and click “Finish” as shown below.



Observe the server console and confirm that you get the message -
“*1 portlet for library-portlet is available for use*”.

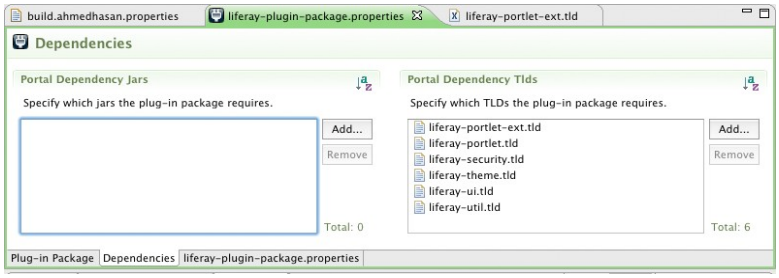
Adding the portlet to a page

1. open the browser and go to http://localhost:8080
2. login as omni admin – test@liferay.com / test
3. click on Add → Page to add a new page “Our Library”
4. go to the page and add our new portlet by clicking Add → More
5. place the porltet in such a way that it is on the right-hand side as shown in the image below.

Some code clean-up before we start

Keeping your code base clean is an extremely important exercise. Through-out this book, we will practically demonstrate this. As a first step we will remove from "tld" files from the file system and list them in the file "liferay-plugin-package.properties" as portal-dependency-tlds. So whatever tld that we specify in this list, will be copied from the portal and used while building the WAR file for this portlet.

1. Open "liferay-plugin-package.properties" file from the path "library-portlet/docroot/WEB-INF"
2. Add the six tld files as shown in the following figure
3. delete all the tld files from WEB-INF/tld
4. Save the project, re-deploy and check our library portlet is working fine as before.



We will more about this file in the subsequent chapters.

4. Creating a Form

Establishing a basic page flow

In this chapter, we will see how to create some basic page flows for the library portlet we have just created.

Create a new page

Add one new JSP file called "update.jsp" inside "docroot/html/library" and put some dummy contents and Save this file.

```
<h1>Add / Edit Form</h1>
```

Modify view.jsp

1. Open view.jsp
2. remove the line -

   "`This is the <b>Library Portlet</b> portlet in View mode.`"
3. enter the code to link to "update.jsp"
4. check the portlet and you should see a link to "update.jsp".

```
<portlet:renderURL var="updateBookURL">
```

```
	<portlet:param name="jspPage" value="/html/library/update.jsp"/>
</portlet:renderURL>

<br/><a href="<%= updateBookURL %>">Add new Book &raquo;</a>
```

create init.jsp

create a new file init.jsp where all common stuff will be put. This file in turn will be included in all other JSP's of this portlet. This way, we need not have to repeat the same code again and again in all JSP files.

Remove these lines from view.jsp and paste into init.jsp

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

<portlet:defineObjects />
```

insert this line at the top of all other JSP files we have so far.

```
<%@ include file="/html/library/init.jsp" %>
```

Create a link to come back.

1. re-open update.jsp and give a link back to the main page.
2. `<a href="<portlet:renderURL/>">« Go Back</a>`

Creating a form to add book

In this step we will further modify our update.jsp to define a simple form to add a book. Before the "Go Back" link, let us have this code,

```
<%
	PortletURL updateBookURL = renderResponse.createActionURL();
	updateBookURL.setParameter(
		ActionRequest.ACTION_NAME, "updateBook");
%>

<form name="<portlet:namespace/>fm" method="POST" action="<%=
updateBookURL.toString() %>">
	Book Title: <input type="text" name="<portlet:namespace/>bookTitle" />
	<br/>Author: <input type="text" name="<portlet:namespace/>author" />
	<br/><input type="submit" value="Save" />
</form>
```

The interfaces "PortletURL" and "ActionRequest" will report problem. To get rid of them just add the following imports in your "init.jsp".

```
<%@page import="javax.portlet.PortletURL"%>
<%@page import="javax.portlet.ActionRequest"%>
```

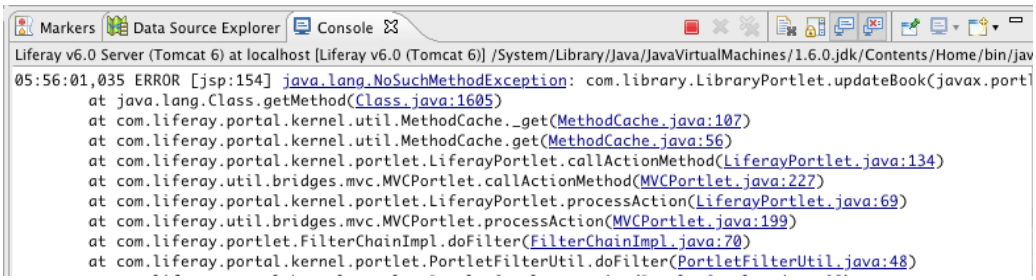
Inside the JSP scriplet we have programmatically declared a variable “updateBookURL” which is of type “actionURL”. We have also set one attribute for this object, the ACTION_NAME.

Once you save all files and the portlet gets deployed, check the “Add Book” page and you will see something like this,

Enter some values and click “Add”, you will get some error on the page

“**Portlet is temporarily unavailable.**”

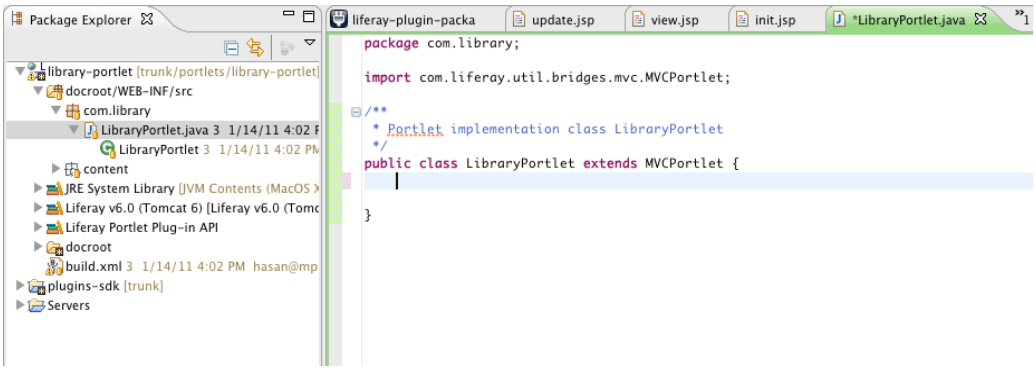
Let us check the eclipse console to know what is causing the problem,



It is clear from the message that the portal server is unable to find a method “updateBook”

In the next step let us see how and where to add this method.

Modify LibraryPortlet.java



Open the portlet class and add a new method – “updateBook”,

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

      String bookTitle = ParamUtil.getString(actionRequest, "bookTitle");
      String author = ParamUtil.getString(actionRequest, "author");

      System.out.println("Your inputs ==> " + bookTitle + ", " + author);
}
```

Now re-deploy the portlet and check the code in our "updateBook" method is being called properly. Once you enter the details of a book and submit the form you should get the message on the console,

```
Your inputs ==> Liferay In Action, Richard Sezov
```

Learnings from this chapter

1. RenderRequest and ActionRequest - difference
2. URL formation – declarative using tags and programmatic
3. <portlet:namespace/>
4. Did you notice the "ParamUtil" class. List down all other api's of this class.

Converting Simple HTML form to AUI form

In this section we will see how to convert simple HTML form we have just created to an AUI form and use the AUI elements.

Insert the AUI taglib definition in init.jsp,

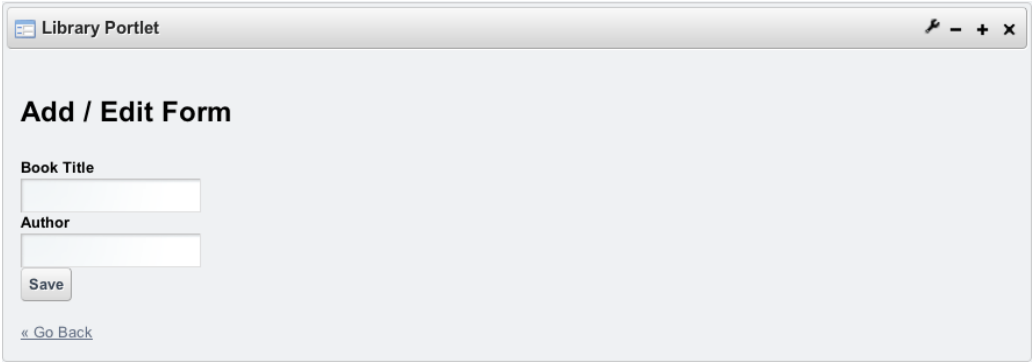
```
<%@ taglib uri="http://liferay.com/tld/aui" prefix="aui" %>
```

Replace our HTML form with the AUI form,

```
<aui:form name="fm" method="POST" action="<%= updateBookURL.toString() %>">
      <aui:input name="bookTitle" label="Book Title"/>
      <aui:input name="author"/>
      <aui:button type="submit" value="Save"/>
</aui:form>
```

1. when you use AUI, you need not have to explicitly give <portlet:namespace/>
2. ensure that you have explicitly specified the method attribute for <aui:form> tag.
3. See we have specified the "label" attribute only for bookTitle and not for author. Why?

Once our changes are deployed, you will see a more cleaner / better form,



We will see more applications of Alloy UI (AUI) in subsequent chapters.

For a detailed discussion on AUI, please refer to the below Liferay Wiki article,

http://www.liferay.com/web/guest/community/wiki/-/wiki/Main/Alloy+UI+Forms+(aui)

Form Validation using jQuery

In this section, we are going to show you how to use jQuery in our portlet. Though we will see more applications of jQuery and various jQuery plugins in the later chapters, here we will see how to add some validation to our "Update Book" form using jQuery validation plugin.

1. refer to the javascript that you need in your portlet.
    1. header-portal-javascript
    2. footer-portal-javascript
    3. header-portlet-javascript
    4. footer-portlet-javascript
2. open the liferay-portlet.xml under WEB-INF and insert the below tag in the appropriate location.

```
<header-portlet-javascript>
      http://ajax.microsoft.com/ajax/jquery.validate/1.7/jquery.validate.min.js
</header-portlet-javascript>
```

3. open "update.jsp"

To know more about integrating jQuery with liferay, read the following liferay Wiki articles.

http://www.liferay.com/web/jonas.yuan/blog/-/blogs/building-jquery-based-plugins-in-liferay-6

http://www.liferay.com/web/julio.camarero/blog/-/blogs/can-i-have-different-jquery-versions-in-liferay

http://www.liferay.com/web/nathan.cavanaugh/blog/-/blogs/using-jquery-or-any-javascript-library-in-liferay-6-0

5. Creating a Service Layer

What is a service layer?

<to be done>

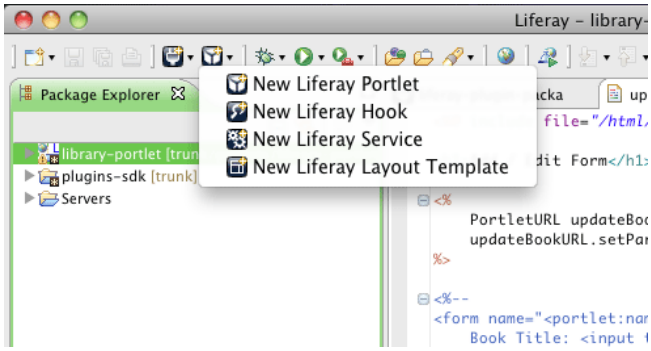
Services offered by a service layer

<to be done>

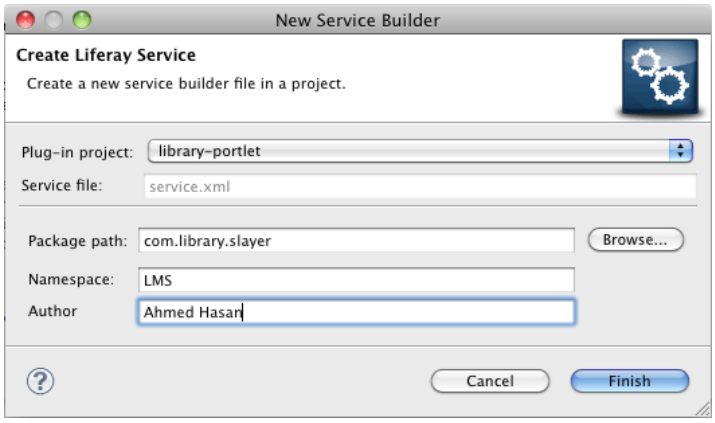
Generating a service layer

In this section, we will generate a service layer that will help us to persist the book information whenever it gets added to our system.

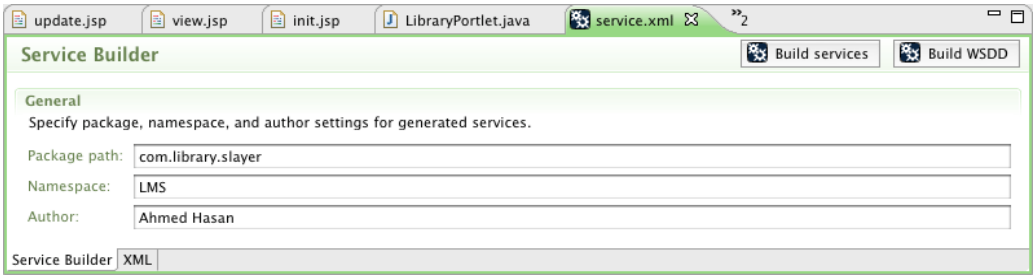
Select the project in eclipse and click “New Liferay Service”



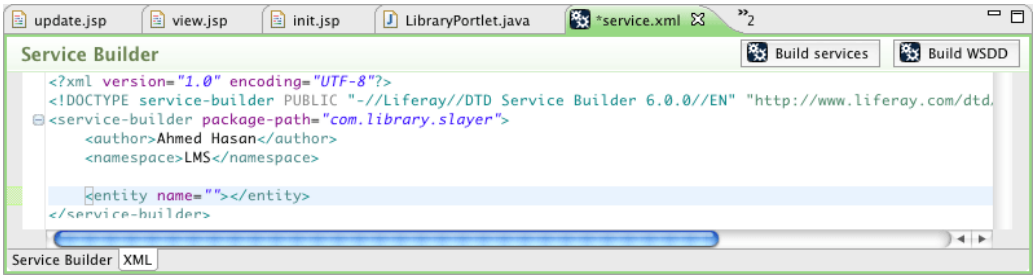
In the next popup, give the properties of the service layer we are going to generate like, package path and Namespace. We recommend the package path to look something like “com.library.slayer”, where slayer stands for service layer, so that all the generated files go under this package and do not interfere with the files that we create for the portlet.



Once you click “Finish”, you will see the service.xml file opening as below.



Click the XML tab on this window to see the corresponding XML file.

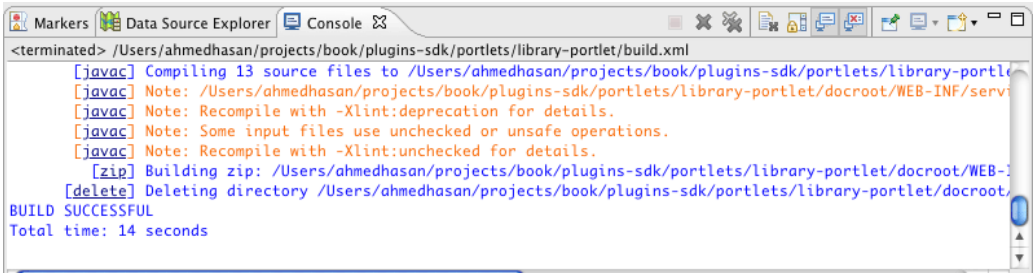


Now edit the service.xml file by replacing the “entity” element with the following entity definition.

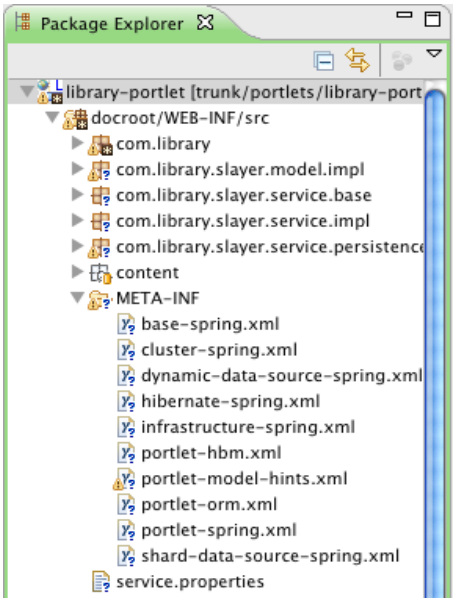
```
<entity name="LMSBook" local-service="true" remote-service="false">
      <!-- PK fields -->
      <column name="bookId" type="long" primary="true"  />

      <!-- UI fields -->
      <column name="bookTitle" type="String" />
      <column name="author" type="String" />

      <!-- Audit fields -->
      <column name="dateAdded" type="Date" />
</entity>
```

In this most simplistic service.xml file, we have defined one field as primary key, two UI fields and one audit field. We have also specified **local-service** as *true* and **remote-service** as *false*.

Now save the changes and click “Build services” button. If all well you will see the “BUILD SUCCESSFUL” message on the console as below. If not, something might be wrong. Please go back and check your service.xml as there could be some syntactical errors.



Now you see the complete list of files that are generated by the service layer.



Invoking the service layer api

In this section, we will see how we are going to integrate our application with the persistence api of service layer just got created. We will make use of the "addLMSBook" to persist the book that gets added by the user.

Let us open our LibraryPortlet.java file and insert the below code to the "updateBook" method,

```
LMSBook book = new LMSBookImpl();

// set primary key
long bookId = 0L;
try {
     bookId =
          CounterLocalServiceUtil.increment(
                    this.getClass().getName());
} catch (SystemException e) {
     e.printStackTrace();
}
book.setBookId(bookId);

// set UI fields
book.setBookTitle(bookTitle);
book.setAuthor(author);

// set audit field(s)
book.setDateAdded(new Date());
```

```
        // insert the book using persistence api
        try {
            LMSBookLocalServiceUtil.addLMSBook(book);
        } catch (SystemException e) {
            e.printStackTrace();
        }
```

Also make sure that you have made the necessary imports to this java file.

Save all your changes and observe that the portlet is getting deployed properly.

Go to your form and add a book.

Confirm the book information is getting inserted to the database by opening the MySQL Query Browser.

You will see a new table "LMS_LMSBook". Keeping adding more books through the form and see the records are getting inserted into this table.

Congratulations!! You have successfully integrated a service layer api with our application.

In the next section we are going to see how to retrieve the records of our table and show in a new page, "list.jsp".

Retrieving the records – use another API

1. create a new jsp file with name "list.jsp" under "/html/library" and insert the include for init.jsp
2. insert this code in the list.jsp

```
<%@ include file="/html/library/init.jsp" %>

<h1>List of books in our Library</h1>
```

3. Open view.jsp and insert a link to list.jsp,

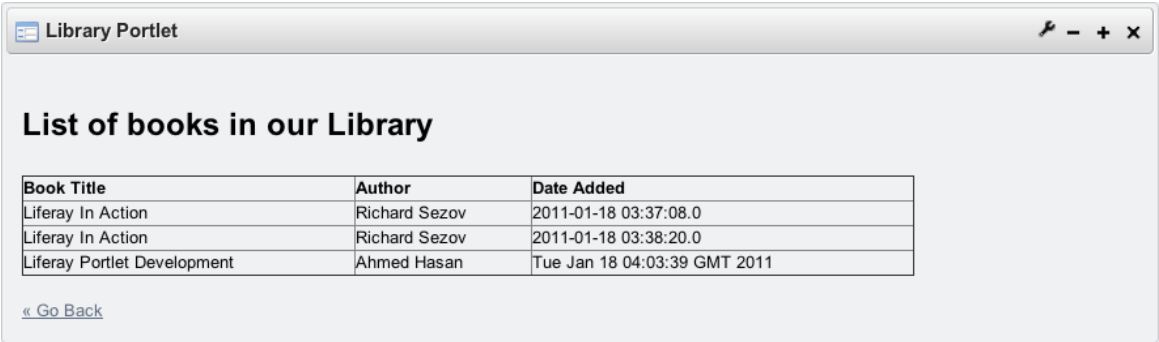
```
<%
    PortletURL listBooksURL = renderResponse.createRenderURL();
    listBooksURL.setParameter("jspPage", "/html/library/list.jsp");
%>

<a href="<%= listBooksURL.toString() %>">Show all books &raquo;</a>
```

4. Go to the browser and check the new link is correctly taking you to the "List of books"
5. Add the following code to list.jsp to get the list of books and display

```
<%
    int count = LMSBookLocalServiceUtil.getLMSBooksCount();
    List<LMSBook> books =
        LMSBookLocalServiceUtil.getLMSBooks(0, count);
```

```
%>

<table border="1" width="80%">
      <tr>
            <th>Book Title</th>
            <th>Author</th>
            <th>Date Added</th>
      </tr>

      <%
            for (LMSBook book : books) {
                  %>
                        <tr>
                              <td><%= book.getBookTitle() %></td>
                              <td><%= book.getAuthor() %></td>
                              <td><%= book.getDateAdded() %></td>
                        </tr>
                  <%
            }
      %>
</table>

<br/><a href="<portlet:renderURL/>">&laquo; Go Back</a>
```

6. Go to the portlet in browser and click on "Show all books", you will see all the books getting listed out as below.



Congratulations!! you have successfully retrieved all the books of our library and displayed them on an new page. We have made use of two new API's of LMSBookLocalServiceUtil – `getLMSBooksCount()` and `getLMSBooks(0, count)`.

Problem when you refresh the page

Now, you will observe a new problem when you do a page refresh by pressing "F5" immediately after the book gets inserted. Whenever you press "F5" a new record gets inserted to the database which is not a desired behavior. In this section, we will see how to get rid of this problem and also sensitizing you to take care of such issues during your real time development.

1. setting a redirectURL on your JSP

   open "update.jsp" and insert a hidden variable "redirectURL" as the first element of the AUI

form.

```
<aui:input type="hidden" name="redirectURL"
           value="<%= renderResponse.createRenderURL().toString() %>"/>
```

2. Changes to the portlet class

   Open LibraryPortlet.java and add the following lines to the end of the “updateBook” method.

```
// gracefully redirecting to the default portlet view
String redirectURL = ParamUtil.getString(actionRequest, "redirectURL");
actionResponse.sendRedirect(redirectURL);
```

3. Save all your changes and check the refresh problem got suitably addressed by now.

Learnings from this chapter

5. Improving our List with Search Container

References

1. Liferay Portal Adminstrator's Guide -