Documentation

Liferay provides a rich store of resources and knowledge to help our community better use and work with our technology.

Liferay Portal 6.2 Developer's Guide

Developing Apps with Liferay IDE

Even if you're a grizzled veteran Java developer, if you're going to be doing a lot of development for your Liferay Portal instance, consider using Liferay IDE. When Liferay IDE is paired with the Plugins SDK or Maven and a Liferay runtime environment, you have a one stop development environment where you can develop your Liferay plugins, build them, and deploy them to your Liferay instance.

Liferay IDE is an extension for Eclipse IDE and supports development of plugin projects for the Liferay Portal platform. You can install Liferay IDE as a set of Eclipse plugins from an update site. The latest version of Liferay IDE supports development of portlets, hooks, layout templates, themes, and Ext plugins. To use Liferay IDE, you need the Eclipse Java EE developer package using Indigo or a later version.

In this section we'll show you how to install Liferay IDE, set up projects for your applications, and deploy them to your portal. We'll get you started with the basics of developing your Liferay application in Liferay IDE. The guide has other chapters geared to each specific plugin type (e.g., the *portlets* chapter covers portlet development, the *hooks* chapter covers hook development, etc.). But, as we create a Liferay portlet project in this chapter, you'll get the gist of how Liferay IDE helps you create all types of plugins easily.

We'll also introduce you to Liferay's Service Builder. It helps you leverage Hibernate's Object-Relational Mapping capabilities and gives you the capability to automatically generate code to access Liferay object data. We'll point out the various editor modes Liferay IDE provides for creating your data entities, relating

them, and building services around them. This section gives you a quick tour, but we've dedicated an entire chapter later in this guide to give Liferay's Service Builder the attention it deserves; check it out, we think you'll be impressed.

To install and set up Liferay IDE, follow the instructions in the first two subsections below. If you're already using *Liferay Developer Studio* (the king of Liferay's development tools), which comes with Liferay Portal Enterprise Edition, skip to the section titled *Testing and Launching your Liferay Server*–Liferay IDE comes preconfigured in Developer Studio.

Installing Liferay IDE

Liferay IDE is an extension of Eclipse IDE; before you install Liferay IDE, let's make sure your Eclipse release can run Liferay IDE, and that you're using a supported version of Java. Then we'll show you the installation process–we give you two choices, depending on whether you want to enter an update site URL for your Eclipse release.

Requirements

Make sure you have a supported Java JRE and Eclipse release:

- Java 6.0 JRE or greater.
- Liferay IDE must be run in one of the following Eclipse releases:
  Eclipse Kepler Java EE (4.3.x)
  Eclipse Juno Java EE (4.2.x)
  Eclipse Indigo Java EE (3.7.x)

Next, we'll show you how to install Liferay IDE.

Installation Steps

To install Liferay IDE and specify an Eclipse update URL, follow these steps:

1. Install Eclipse Kepler, Juno, or Indigo from the Eclipse website.

2. Run the Eclipse executable file (e.g., `eclipse.exe`).

3. When Eclipse opens, go to *Help → Install New Software...*.

4. In the *Work with* field, enter the update site URL `http://sourceforge.net/projects/lportal/files/Liferay IDE/2.0.0 GA1/updatesite/` and press *Enter*.

5. Liferay IDE features should be detected. Select them and click *Next*.

6. After calculating dependencies, click *Next*, accept the license agreement, and click *Finish* to complete the installation.

7. Restart Eclipse to verify that Liferay IDE is properly installed.

8. After restarting Eclipse, go to *Help → About Eclipse*; if you see a Liferay IDE icon badge as in the screenshot below, it's properly installed.

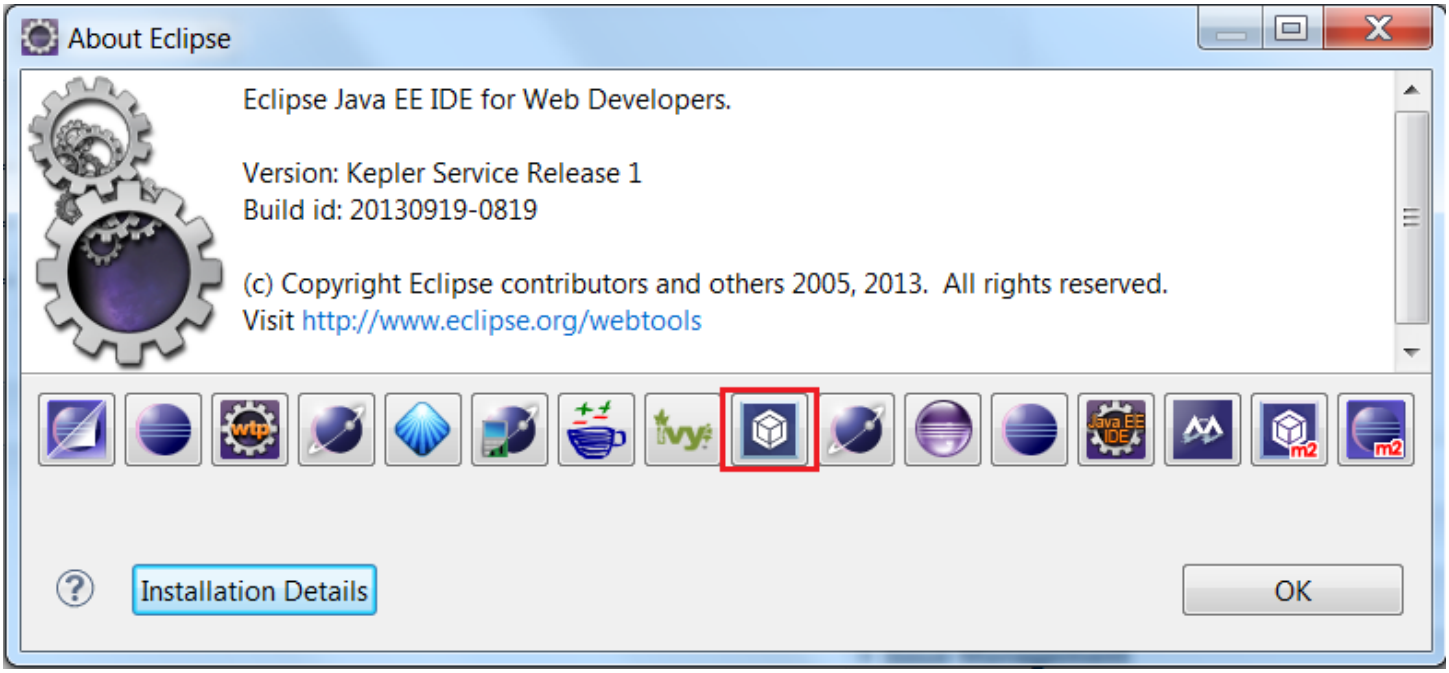

Figure 2.1: Once you've installed Liferay IDE, you can find the Liferay IDE logo in Eclipse by clicking *Help → About Eclipse*.

Next, we'll show you how to install Liferay IDE without specifying an Eclipse update URL.

Alternative Installation

To install Liferay IDE from a `.zip` file, follow these steps:

1. Install Eclipse Kepler, Juno, or Indigo from the Eclipse website.

2. Download the IDE 2.0 `.zip` file from http://sourceforge.net/projects/lportal/files/Liferay IDE/2.0.0 GA1/ for your operating system.

3. Run the Eclipse executable file (e.g., `eclipse.exe`).

4. When Eclipse opens, go to *Help → Install New Software...*.

6. Liferay IDE features should be detected. Select them and click *Next*.

7. After calculating dependencies, click *Next*, accept the license agreement, and click *Finish* to complete the installation.

8. Restart Eclipse to verify that Liferay IDE is properly installed.

After restarting Eclipse, you can verify that Liferay IDE is installed by going to *Help → About Eclipse* and finding the Liferay IDE icon badge.

Let's set up Liferay IDE now that you have it installed.

Setting Up Liferay IDE

Now that you have Liferay IDE installed, either from a downloaded zip file or from the update site appropriate for your Eclipse version, you need to perform some basic setup. This section describes the setup steps to perform so you can develop your Liferay portal and test your customizations.

Before setting up Liferay IDE, let's make sure you have all the appropriate software packages installed.

Requirements

Before setting up Liferay IDE, you need to have appropriate versions of Liferay Portal, Liferay Plugins SDK and/or Maven, and Eclipse. Make sure you satisfy these requirements before proceeding:

1. Liferay Portal 6.0.5 or greater is downloaded and unzipped.

2. Liferay Plugins SDK 6.0.5 or greater is downloaded and unzipped, and/or any version of Maven is installed. If you're using the Plugins SDK, make sure the Plugins SDK version matches the Liferay Portal version.

3. You've installed an appropriate Eclipse IDE version for Java EE Development, and the Liferay IDE extension–see the *Installation* section if you haven't already done this.

Note: Earlier versions of Liferay (e.g., 5.2.x) are not supported by the Liferay IDE.

Let's set up your Liferay Plugins SDK.

Setting Up the Liferay Plugins SDK

Before you begin creating new Liferay plugin projects, a supported Liferay Plugins SDK and/or Maven installation and Liferay Portal must be installed and configured in your Liferay IDE. If you're thinking, "Wait a second, buster! You told me earlier that the Plugins SDK and Maven could be used without Liferay IDE!", then you're right. In the second half of this chapter, we'll explain how to use the Plugins SDK and Maven on its own, with a text editor. Here, we explain the easiest way to use the Plugins SDK: by running it from Liferay IDE.

1. In Eclipse, open the *Installed Plugin SDKs* dialog box–from your *Windows* dropdown menu, click

Preferences → Liferay → Installed Plugin SDKs.

2. Click *Add* to bring up the *Add SDK* Dialog.

3. Browse to your Plugins SDK installation. The default name is the directory name; you can change it if you want.

4. Select *OK* and verify that your SDK was added to the list of *Installed Liferay Plugin SDKs*.

Note: You can have multiple Plugins SDKs configured. You can set the default Plugins SDK by checking its box in the list of *Installed Liferay Plugin SDKs*.

Let's set up your Liferay Portal Tomcat runtime and server.

Liferay Portal Runtime and Server Setup

You can run Liferay on any application server supported by Liferay Portal. Here, for demonstration purposes, we'll set up our Liferay runtime on the Tomcat application server. The steps you'd follow for any other supported application server would be similar. For a list of Liferay bundles with other application servers, please visit Liferay's Downloads page. For instructions on installing Liferay manually on other application servers, please refer to the Installation and Setup chapter of *Using Liferay Portal 6.2*.

1. In Eclipse, open the *Server Runtime Environments* dialog box–go to *Window → Preferences → Server → Runtime Environments*.

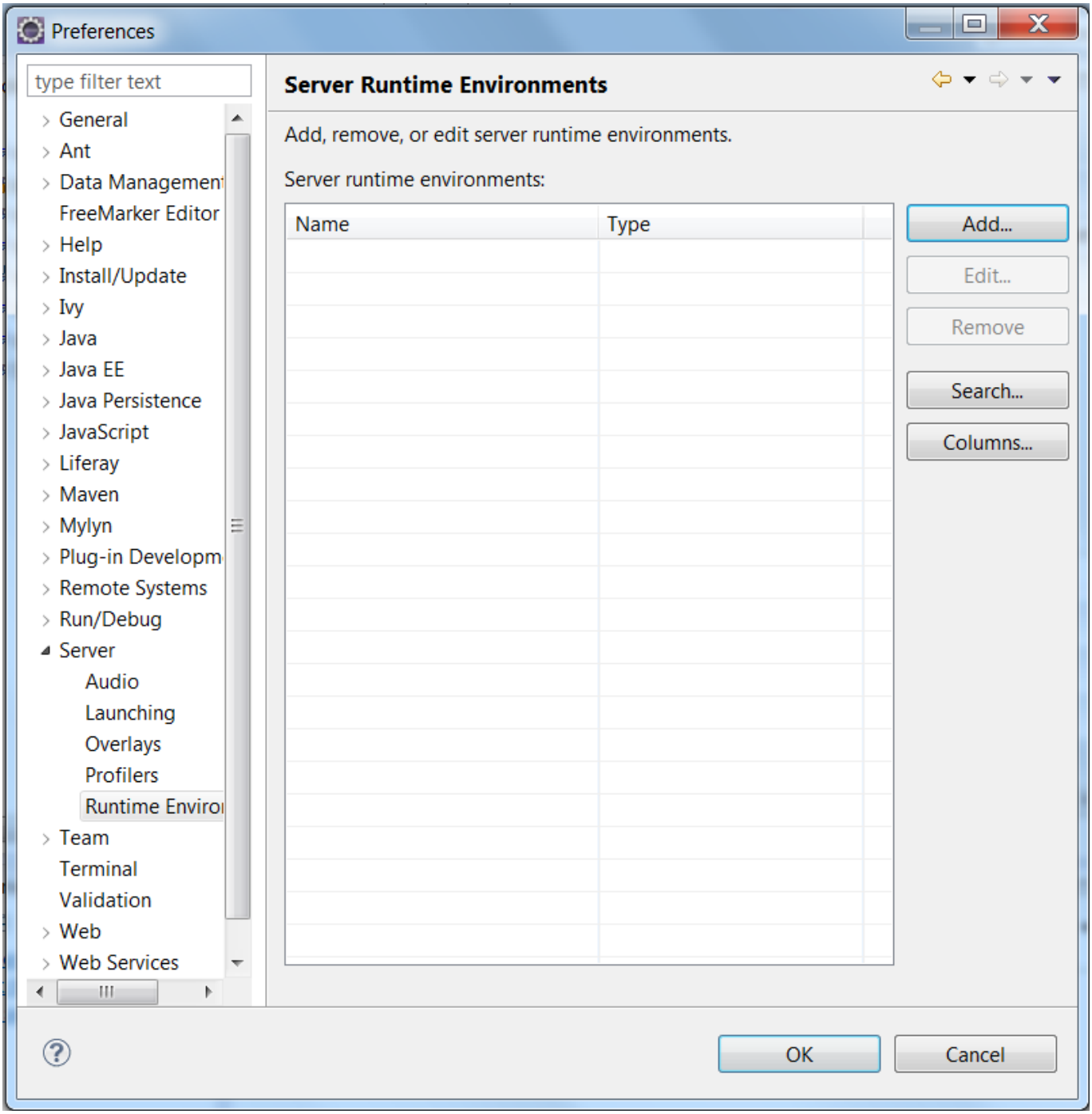

Figure 2.2: Liferay IDE provides wizards for creating new Liferay server runtime environments.

2. Click *Add* to add a new Liferay runtime; find *Liferay v6.2 (Tomcat 7)* under the *Liferay, Inc.* category and click *Next*.

3. Click *Browse* and select your `liferay-portal-6.2.x directory`.

4. If you've selected the Liferay portal directory and a bundle JRE is present, it is automatically selected as the server's launch JRE. If no JRE bundle is present, then you must select the JRE to use for launch by clicking *Installed JREs...*.

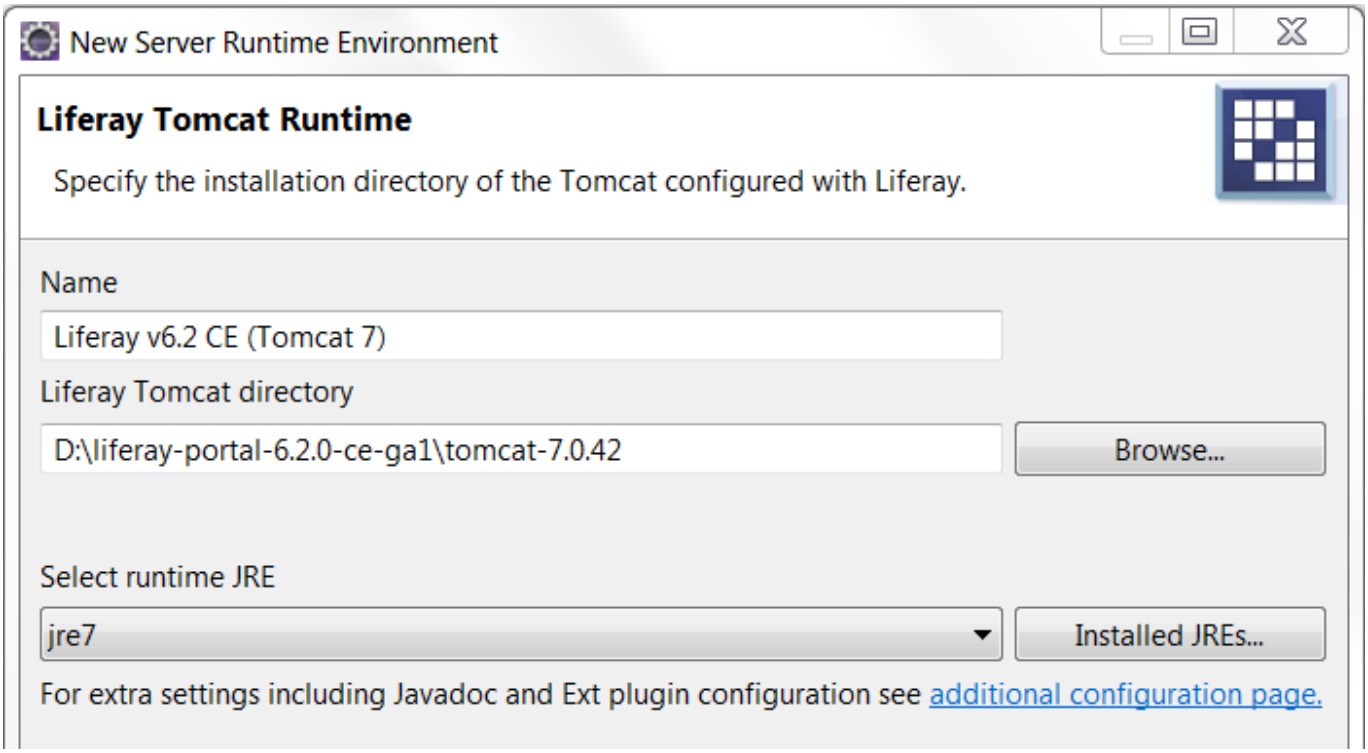

Figure 2.3: If you have multiple JREs installed on your system, choose the one which should run Liferay.

5. Click *Finish*; you should see your Liferay portal runtime listed in *Preferences → Server* &rarr *Runtime Environments*.

6. Click *OK* to save your runtime preferences.

7. If you haven't created a server, create one now from the *Servers* view in Liferay IDE; then you can test the server.

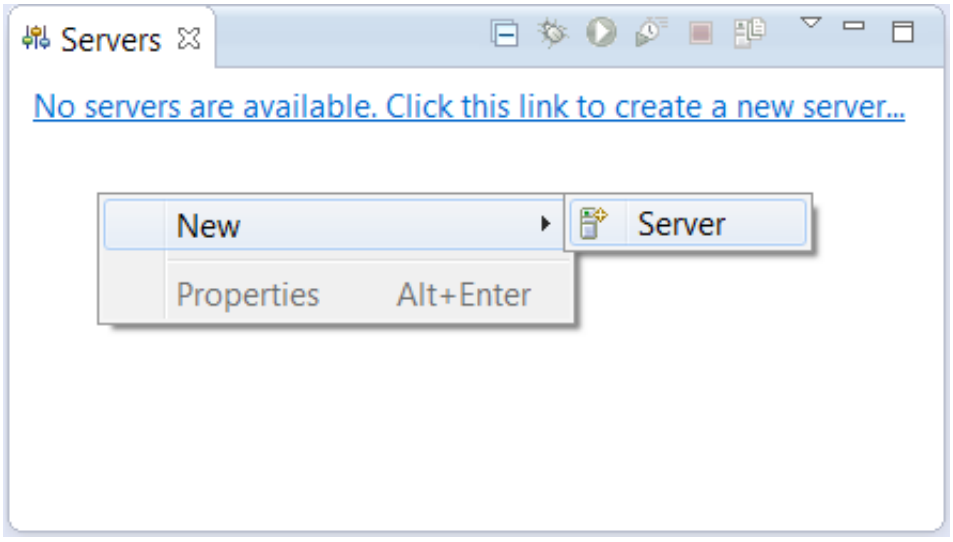

Figure 2.4: If you haven't created a Liferay server yet, you can do so from the *Servers* tab in Liferay IDE.

8. Scroll to the *Liferay, Inc* folder and select *Liferay v6.2… Server*. Choose the *Liferay v6.2…* runtime environment that you just created.

Now your server is set up. Let's launch it and perform some tests!

Launching and Testing Your Liferay Server

Once your Liferay Portal Server is set up, you can launch it from the Servers tab in Eclipse. You have a few options for launching and stopping the server once it's selected in the Servers tab.

From the *Servers* tab:

- Click on the green *Start the Server* button to launch it (or use *Ctrl+Alt+R*).
- Click on the red *Stop the Server* button to stop it (or use *Ctrl+Alt+S*). You'll only see this button if the server is running.
- Right click the server and select *Start*.
- Right click on the server and select *Stop*.

Once the server is launched, you can open Liferay portal home from the *Servers* tab by right clicking your Liferay Tomcat server and selecting *Open Liferay Portal Home*.

Next, you'll learn to create new Liferay projects in Liferay IDE.

Creating New Liferay Projects

Plugins for Liferay Portal must be created inside of a Liferay project. A Liferay project is essentially a root directory with a standardized structure containing the project's (and each of its plugins') necessary files. Since each plugin type requires a different folder and file structure, let's create a project to illustrate the process. Have you heard of the hip new social networking site for noses, *Nose-ster*? Harold Schnozz, the site's founder, wants to capitalize on the site's popularity by providing users with the ability to organize local meetings and events. For instance, there's a really active group of noses in Minneapolis, MN, who'd like to schedule a regional dance in January, which they're calling the Frozen Boogie. Why does this concern us? Mr. Schnozz has hired us to develop the necessary portlets to allow users to create and view events on the Nose-ster portal.

If you've been following our Liferay IDE configuration instructions, your Plugins SDK and Liferay portal server have already been configured in Liferay IDE. Now let's create a new Liferay plugin project in Liferay IDE.

1. Go to *File → New → Liferay Plugin Project*.

2. In the project creation wizard, you'll name and configure your project.

   We'll create a plugin project that we'll use throughout this guide. First, we'll create a bare bones plugin project; then, we'll manually add an additional plugin to the project and add additional configurations.

   2.1. Provide both a *Project Name*, which is used to name the project's directory, and a *Display Name*, which is used to identify the plugin when adding it to a page in Liferay Portal. Our demonstration project will have the project name *event-listing-portlet* and the display name *Event Listing*.

   2.2. Leave the *Use default location* checkbox checked. By default, the default location is set to your current workspace. If you'd like to change where your plugin project is saved in your file system, uncheck the box and specify your alternate location.

   2.3. Select the *Ant (liferay-plugins-sdk)* option for your build type. If you'd like to use *Maven* for your

build type, navigate to the *Developing Plugins Using Maven* section for details.

2.4. Your newly configured SDK and Liferay Runtime should already be selected. If you haven't yet pointed Liferay IDE to a Plugins SDK, click *Configure SDKs* to open the *Installed Plugin SDKs* management wizard. You can also access the *New Server Runtime Environment* wizard if you need to set up your runtime server; just click the *New Liferay Runtime* button next to the *Liferay Portal Runtime* dropdown menu.

2.5. Under *Plugin Type*, indicate which plugin type your project will hold by selecting one from the list. You can choose from *Portlet*, *Service Builder Portlet*, *Hook*, *Layout Template*, *Theme*, or *Ext*. Liferay IDE provides handy wizards for creating new Liferay projects. Our demonstration project will hold service builder portlets for the Nose-ster organization, so make sure *Service Builder Portlet* is selected.

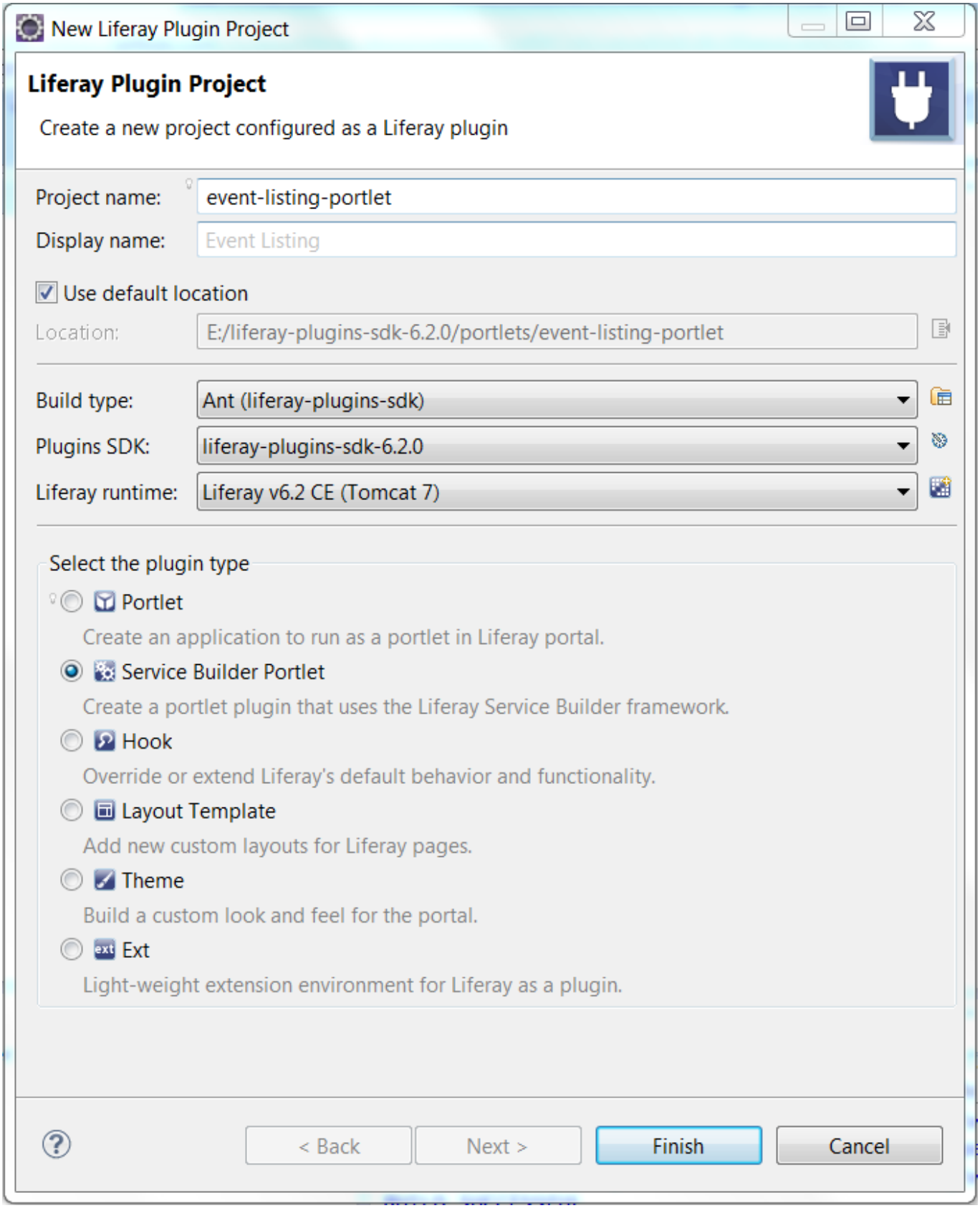

Figure 2.5: The wizard for creating a new service builder portlet project uses the information you specify to customize various configuration files in the new project.

Great! You've created a Liferay portlet project!

You can find more information on Liferay's plugin frameworks in the chapter on portlet development. In that chapter, we'll discuss the plugin creation wizard in more detail.

Note: We're creating the event-listing-portlet project now so that we can highlight how Liferay IDE simplifies project creation. For more information on creating portlets, please see the chapter of this guide on portlets. Similarly, for more information on themes, layout templates, hooks, or Ext plugins, please refer to the

appropriate chapter of this guide.

Our *event-listing-portlet* plugin project should appear in the Eclipse package explorer. The project was created in the plugins SDK you configured, under the directory corresponding to the plugin type the project contains. Here's the generalized directory structure for portlet projects created in Liferay IDE/Developer Studio:

- `PROJECT-NAME/`
    - `docroot/WEB-INF/src`
    - `build.xml` **- Common project file**
    - `docroot/`
        - `css/`
            - `main.css`
        - `view.jsp`
        - `js/`
            - `main.js`
        - `META-INF/`
            - `MANIFEST.MF`
        - `WEB-INF/`
            - `lib/`
            - `tld/`
                - `aui.tld`
                - `liferay-portlet-ext.tld`
                - `liferay-portlet.tld`
                - `liferay-security.tld`
                - `liferay-theme.tld`
                - `liferay-ui.tld`
                - `liferay-util.tld`
            - `liferay-display.xml`
            - `liferay-plugin-package.properties` **- Common project file**
            - `liferay-portlet.xml`
            - `portlet.xml`
            - `web.xml`
        - `icon.png`
        - `view.jsp`

All projects, regardless of type, are created with a `build.xml` and a `liferay-plugin-package.properties` file–we've highlighted each of them with the note **Common project file** in the directory structure above. The `build.xml` file allows Liferay IDE to use Ant to automatically compile and

deploy your plugins. Another default file is `liferay-plugin-package.properties`. This file contains important metadata for your project. Liferay IDE's *properties* view gives you a simple interface to inspect or specify the file's fields, including your project's dependencies and deployment context, display name, and Liferay version. If you publish your project as an app to Liferay Marketplace, the value of the `name` property in `liferay-plugin-package.properties` is used as the app's name. The value of the `liferay-versions` property is used on Liferay Marketplace to specify the versions of Liferay on which your application is intended to run. Next, you need to deploy your new plugin project to your Liferay Server.

Deploying New Liferay Projects to a Liferay Server

You have a plugin project, but you need to deploy it onto your Liferay Server. The easiest way to deploy a plugin project is to drag the project from the Package Explorer view onto your Liferay runtime in the Servers view. Alternatively, you can use the following method:

1. Select your new plugin project then right click the Liferay Server in the *Servers* tab.
2. Select *Add and Remove…*.
3. Select your plugin project and click *Add* to deploy it to the server.
4. Click *Finish*.

Deploy your project. You should see the project get deployed to your Liferay server; in the console you'll see a message, like the one below, indicating that your new portlet is available for use.

```
INFO  [localhost-startStop-2][PortletHotDeployListener:490] 1 portlet for
event-listing-portlet is available for use
```

Open *Liferay Portal Home* (`http://localhost:8080/` for a fresh Liferay installation) and log in with your administrator account. If this is your first time starting Liferay, follow the instructions in the setup wizard. For more setup wizard details, see the Using Liferay's Setup Wizard section of Chapter 15 in *Using Liferay Portal 6.2*.

Once you're logged in, click *Add → More*; expand the *Sample* category and click the *Add* link next to your Event Listing application. Your *Event Listing Portlet* shows on the page.

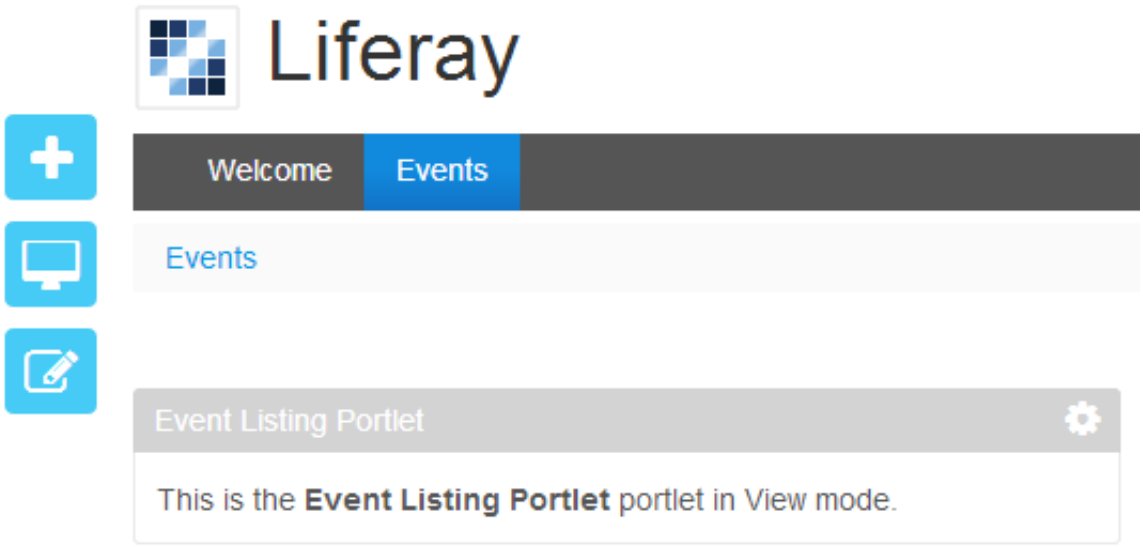

Figure 2.6: Voila! You can add your brand new portlet (empty for now) to any page.

Great, now you can create projects in Liferay IDE! Next, let's learn how to create new plugins inside of existing projects in Liferay IDE. But before we do, let's clean out the bare-bones portlet from our event-listing-portlet project.

The portlet project wizard conveniently creates a default portlet named after the project. However, for demonstration purposes, we want to begin creating portlets with a clean slate in our project. Let's tear out the default Event Listing portlet by removing its descriptors and it's JSP.

1. Open the portlet's `docroot/WEB-INF/liferay-display.xml` file and remove the `<portlet id="event-listing" />` tag.

2. Open the `docroot/WEB-INF/liferay-portlet.xml` file and remove the `<portlet>...</portlet>` tags and code residing between those tags.

3. Navigate to the `docroot/WEB-INF/portlet.xml` file and remove the `<portlet>...</portlet>` tags and code residing between those tags.

4. Remove the `docroot/view.jsp` file.

Super! You've cleaned out the default portlet from the project. Now you're ready to start creating the example plugins.

Creating Plugins

Liferay projects can contain multiple plugins. If you've followed the instructions from the earlier section on creating new Liferay projects, you should already have created the event-listing-portlet project. In this section we'll add two portlets to the event-listing-portlet project: the Location Listing portlet and the Event Listing portlet. This illustrates the general process for creating plugins inside of an existing Liferay project. Later in this guide, when we complete developing the Event Listing and Location Listing portlets, they'll allow users to add, edit, or remove events or locations, display lists of events or locations, search for particular events or locations, and view the details of individual events or locations. For now, we'll show you how to create both portlets in the event-listing-portlet project.

Your Liferay IDE's Package Explorer shows your Event Listing plugin project. Since it's a portlet type project it has a skeleton in place for supporting more portlet plugins. Let's start by creating the Location Listing portlet.

Use the following steps to create the Location Listing portlet:

1. Right click on your `event-listing-portlet` project in Liferay IDE's *Package Explorer* and select *New → Liferay Portlet*.

2. The *New Liferay Portlet* dialog box appears with your plugin project *event-listing-portlet* selected as the *Portlet plugin project* by default. It's a good idea to name your *Portlet class* after the name of your portlet. We'll name the class *LocationListingPortlet* in this example. Name your *Java package* after the plugin's parent project, so it will be *com.nosester.portlet.eventlisting*, and leave the *Superclass* as *com.liferay.util.bridges.mvc.MVCPortlet*. Alternatively, you could have selected *com.liferay.portal.kernel.portlet.LiferayPortlet* or *javax.portlet.GenericPortlet* for your superclass.

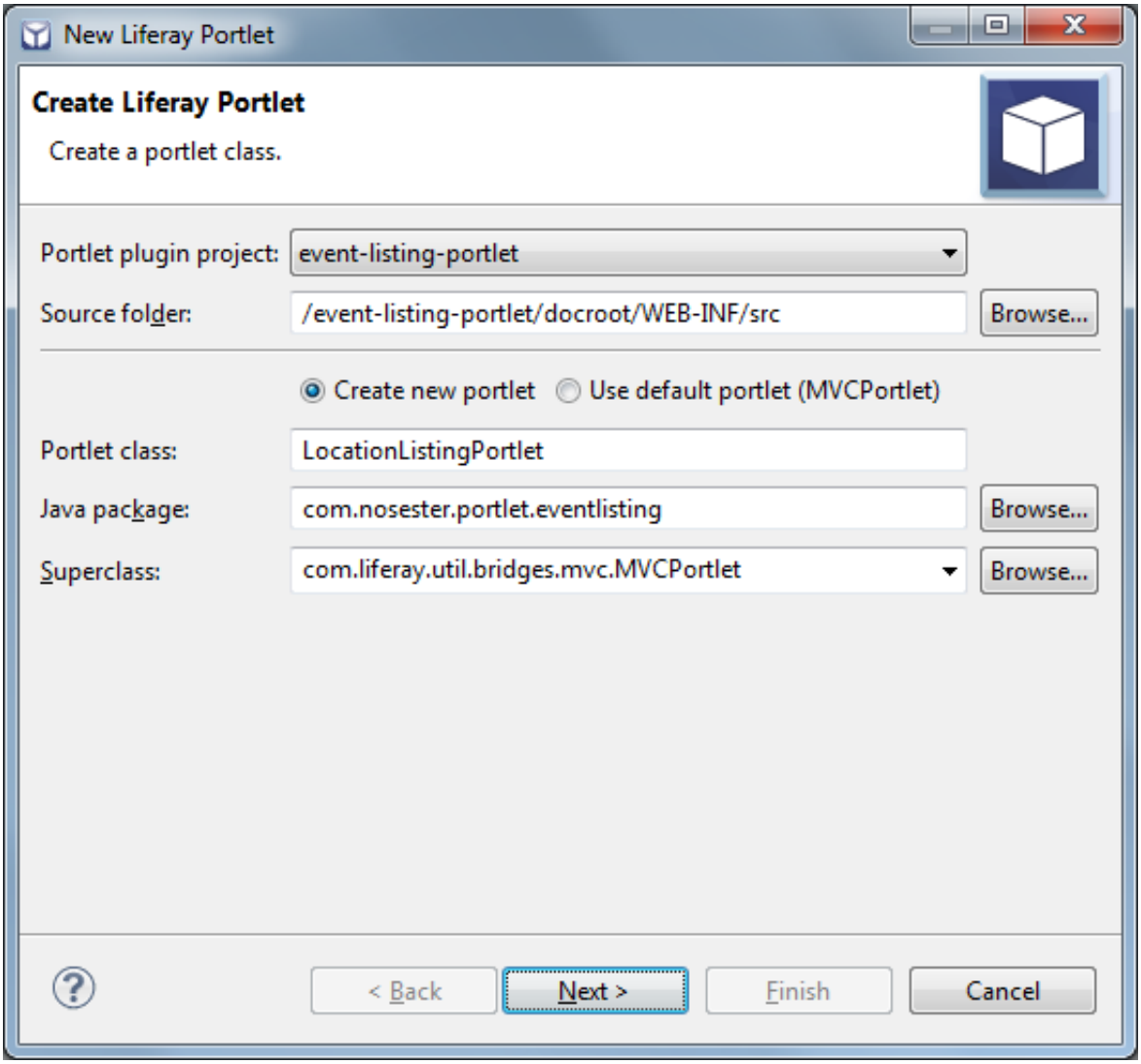

Figure 2.7: Liferay IDE's portlet creation wizard makes creating a portlet class is easy.

Here are the portlet class values to specify for the example Location Listing portlet: - **Portlet plugin project:** *event-listing-portlet* - **Source folder:** */event-listing-portlet/docroot/WEB-INF/src* - **Portlet class:** *LocationListingPortlet* - **Java package:** *com.nosester.portlet.eventlisting* - **Superclass:** *com.liferay.util.bridges.mvc.MVCPortlet*

Click *Next*.

3. In the next window of the *New Liferay Portlet* wizard, you'll specify deployment descriptor details for your portlet. First enter the *Name* of your portlet–in our example, this will be *locationlisting*. Next, enter the portlet's *Display name* and *Title*; we'll specify both as *Location Listing Portlet*. In this window, you can also specify which portlet modes you'd like your portlet to have. *View* mode is automatically selected. There are also options for creating resources: you can specify the folder where JSP files will be created as well as whether or not a resource bundle file will be created. We'll leave the *Create JSP files* box flagged, specify *html/locationlisting* as the JSP folder, and flag the *Create resource bundle file* box.

   Here are the portlet deployment descriptor details to specify for the Location Listing portlet: - **Name:** *locationlisting* - **Display name:** *Location Listing Portlet* - **Title:** *Location Listing Portlet* - **JSP folder:** *html/locationlisting*

   Click *Next*.

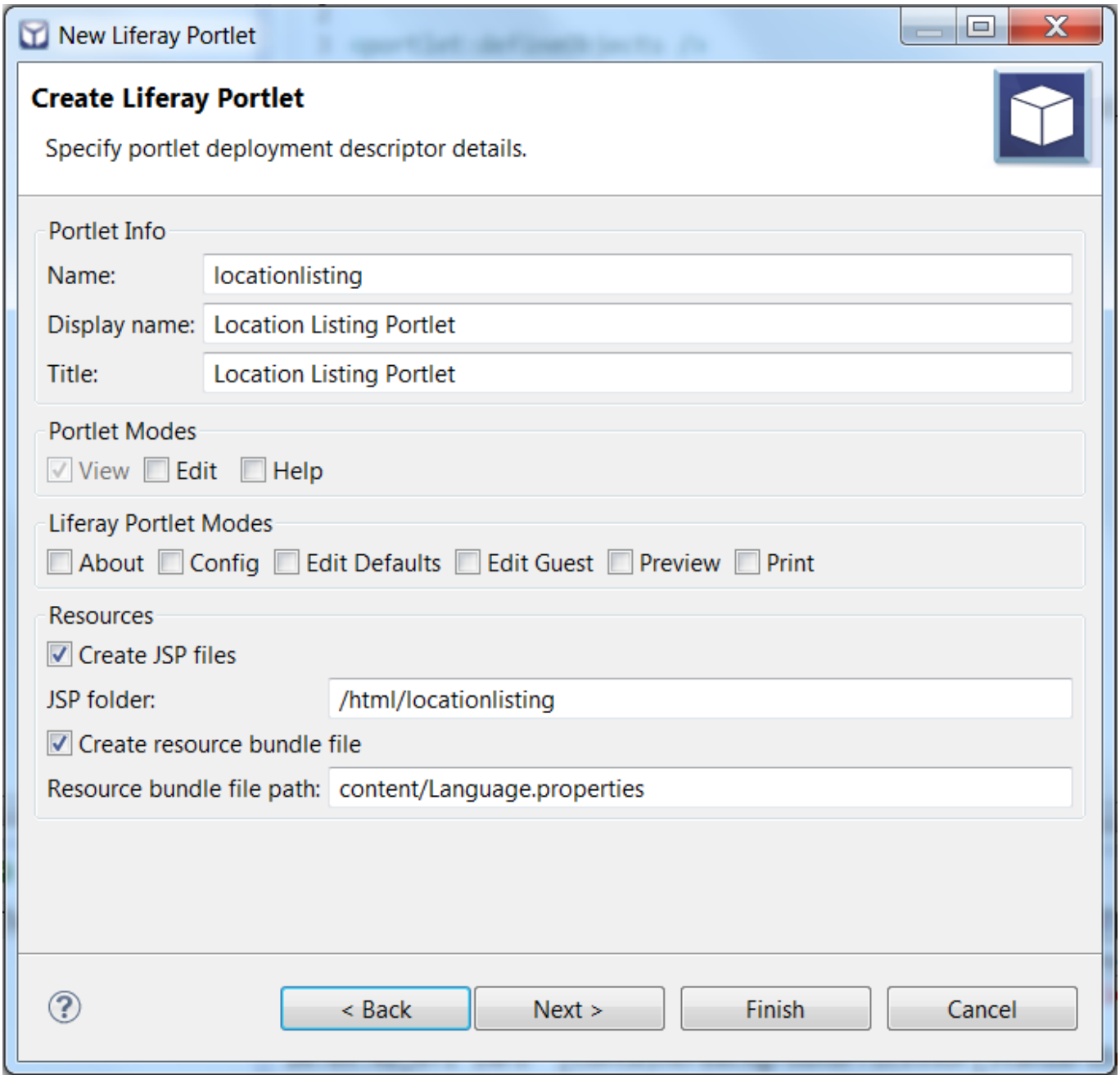

Figure 2.8: Liferay IDE's portlet creation wizard let's you specify the deployment descriptors for your portlets.

4. This window lets you specify portlet deployment descriptor details that are specific to Liferay. You can set the file paths of your portlet's custom icon, main CSS file, and main JavaScript file. You can also specify a CSS class wrapper. Next, you can also choose the category for your portlet (it's categorized under *Sample* by default), and choose whether or not to add it to the *Control Panel* of your Liferay Portal. Accept the default, leaving the *Add to Control Panel* box unflagged.

5. The last step is to specify modifiers, interfaces, and method stubs to generate in the Portlet class. Accept the defaults and click *Finish*.

Use the following steps to create the Event Listing portlet:

1. Right-click your event-listing-portlet project → *New* → *Liferay Portlet*. Specify *EventListingPortlet* as the name of the portlet class, enter *com.nosester.portlet.eventlisting* as its Java package, and select *com.liferay.util.bridges.mvc.MVCPortlet* as it's superclass.

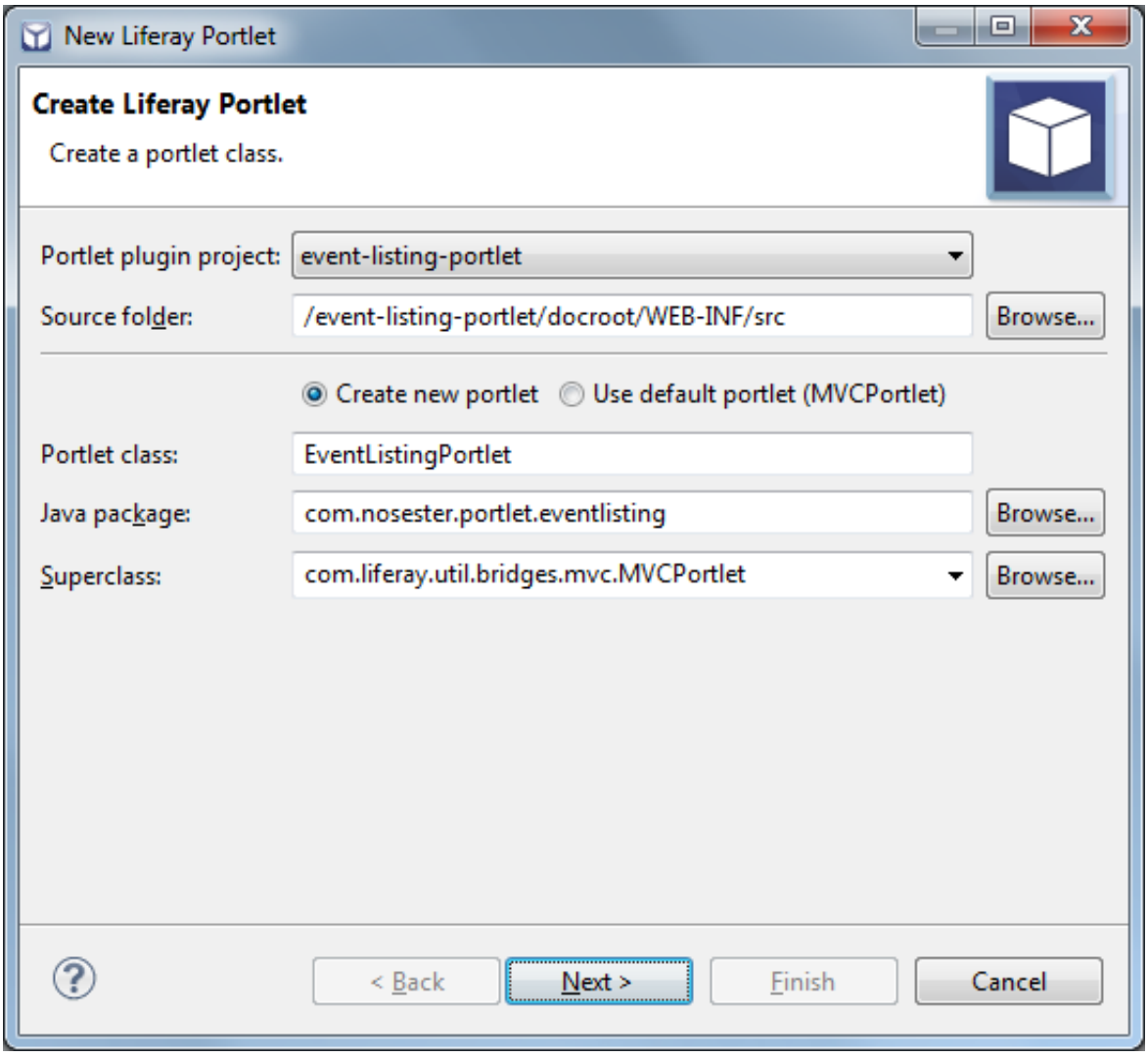

Figure 2.9: Creating portlet classes is simple with Liferay IDE's portlet creation wizard.

Here are the portlet class values to specify for the example Event Listing portlet: - **Portlet plugin project:** *event-listing-portlet* - **Source folder:** */event-listing-portlet/docroot/WEB-INF/src* - **Portlet class:** *EventListingPortlet* - **Java package:** *com.nosester.portlet.eventlisting* - **Superclass:** *com.liferay.util.bridges.mvc.MVCPortlet*

Click *Next*.

2. In this window we'll specify the portlet's deployment descriptor details.

   Here are the portlet deployment descriptor details to specify for the Event Listing portlet: - **Name:** *eventlisting* - **Display name:** *Event Listing Portlet* - **Title:** *Event Listing Portlet* - **JSP folder:** *html/eventlisting*

   Click *Next*.

3. This window lets you specify portlet deployment descriptor details that are specific to Liferay. You can set the file paths of your portlet's custom icon, main CSS file, and main JavaScript file. You can also specify a CSS class wrapper. In the *Liferay Display* section, you can choose the category for your portlet (it's categorized under *Sample* by default), and choose whether or not to add it to the *Control Panel* of your Liferay Portal. Accept the default, leaving the *Add to Control Panel* box unflagged and click *Next*.

4. The last step in creating your portlet with the wizard is to specify modifiers, interfaces, and method stubs to generate in the Portlet class. Accept the defaults and click *Finish*.

By default, new portlets use the MVCPortlet framework, a light framework that hides part of the complexity of portlets and makes the most common operations easier. The default MVCPortlet project uses separate JSPs for each portlet mode: each of the registered portlet modes has a corresponding JSP with the same name as the mode. For example, `edit.jsp` is for edit mode and `help.jsp` is for help mode.

Let's redeploy the plugin project to make our portlet plugins available in the portal. In the *Servers* tab, simply right click the *event-listing-portlet* project, then click *Redeploy*.

Now you've created and deployed the *Location Listing* portlet and the *Event Listing* portlet from the same project. Eventually, when the Location Listing portlet is complete it will allow users to enter viable event locations.

Next, we'll show you how our Service Builder tool helps you generate your model, persistence, and service layers.

Using the Service Builder Graphical Editor

Loose coupling is a great principle to use when developing your applications. By keeping all of your code for fetching data self contained in a service layer, separate from the business logic of your application, you can more easily swap out your entire service layer without disrupting the functionality of your application.

Service Builder is a model-driven code generation tool that lets you define custom object models called entities. Service Builder reads the contents of a file you create called `service.xml` and automatically creates your application's model, persistence, and service layers, freeing you to focus on the higher level aspects of your application's code.

Why should you use Service Builder? Because it lets different portlets access the same data and application logic, creating an underlying framework that supports a portal environment. If your database access code is buried in a single application's code, it can't readily be shared with other applications, and your efforts will be duplicated with each application you write. Service Builder puts the generated code in a service `JAR` file inside of one plugin, but it can be easily shared among all portlets.

To allow you more than one way to view and edit the `service.xml file`, Service Builder gives you three modes to work in:

- Overview mode provides an easy to use graphical interface in Liferay IDE where you can add to and edit the `service.xml` file. Overview mode also gives you a *Build Services* button to generate the service layer.
- Diagram mode gives you a visualization of the relationships between service entities; it's often helpful to create your entities using diagram mode.
- Source mode displays the raw XML content of the `service.xml` file.

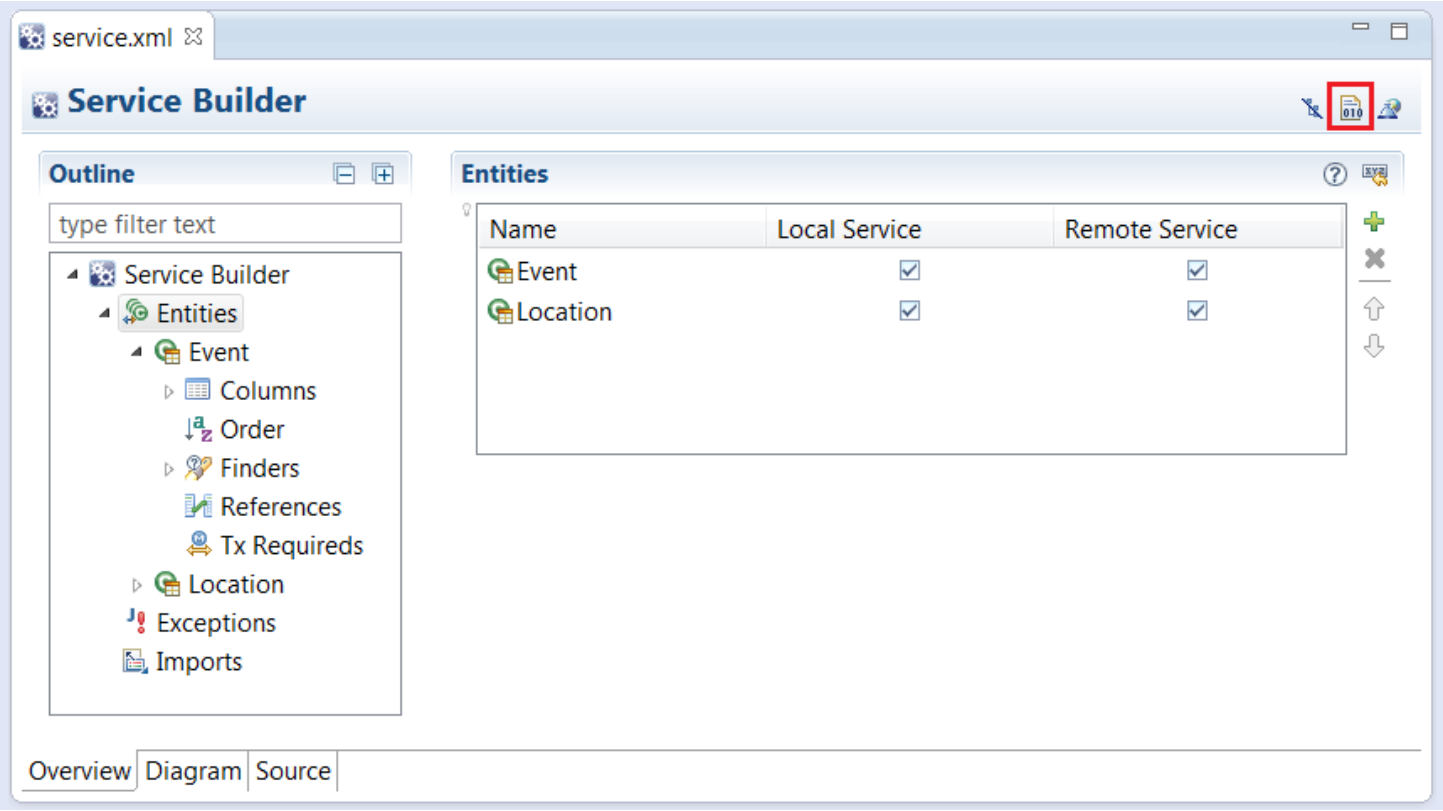

Figure 2.10: Liferay IDE provides an Overview view of `service.xml` which allows you to edit the file by drilling down through a menu and editing form fields instead of editing the XML directly. It also provides a *Build Services* button for running Service Builder.

With Liferay IDE, generating your service layer is easy. First you'll create `service.xml`, by selecting your project in the Package Explorer and then selecting *File → New → Liferay Service Builder*. Service Builder creates a `service.xml` file in your `docroot/WEB-INF/src` folder and displays the file in overview mode. If you're following along with the `event-listing-portlet`, you already have the `service.xml` file because we created service builder portlet project during setup.

Our Service Builder chapter of this guide will lead you through filling out `service.xml` to define the following:

- Global service information
- Service entities
- The attributes for each service entity
- Relationships between service entities
- Ordering of service entities instances

- Service entity finder methods

In the Service Builder chapter of this guide, we'll show you how our two custom portlets, the Events Listing Portlet and the Location Listing Portlet, can be developed more efficiently and modularly by using Service Builder. We'll describe the contents of `service.xml` in detail, and get you started using Service Builder to develop your custom applications using our code generation tool. And if *code generator* is a bad word to you, let us assure you that Liferay always gives you full control over all your code, including code generated by Service Builder.

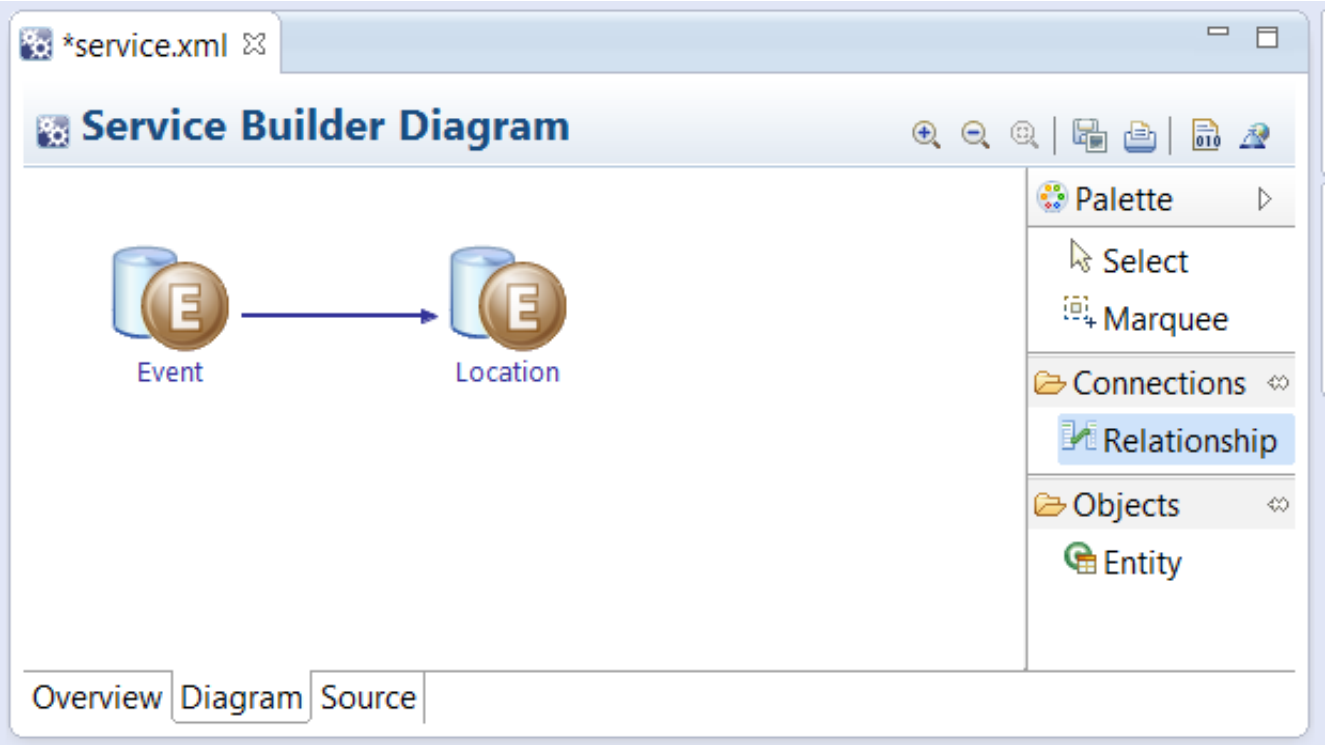

Figure 2.11: Liferay IDE provides a diagram view of `service.xml` which provides a visual aid to understanding the relationships between service entities.

Once you've generated your `service.xml`, you can build services. When viewing `service.xml` in overview mode, there's a button available at the top right hand corner of the window. The button looks like a document with the numerical sequence *010* in front of it. Alternatively, right click on your project and select *Liferay → Build Services*. Once the process is finished, you'll see a plethora of new Java classes under`docroot/WEB-INF/src` in your project that Service Builder generated for you. Now you can get out there and write your business logic, but make sure to check out the Service Builder chapter of this guide for a thorough description of its capabilities.

Now you know how to create projects and plugins from scratch and you know about Service Builder's amazing time-saving capabilities. Let's learn how to import existing projects into Liferay IDE.

Importing Existing Liferay Projects from a Plugins SDK

Do you want to import one or more Liferay projects into your Liferay IDE workspace from a Liferay Plugins SDK? Liferay IDE makes it easy. Don't worry if the projects already contain `.project` or `.classpath` files, the process we'll show you will still import them into your workspace.

Note: This section assumes that you've created projects with the Plugins SDK and are familiar with the directory structure used by the Plugins SDK. If you need to, check out the *Plugins SDK* section of this chapter; it comes right after this section.

First, let's look at the steps for importing a single Liferay project from a Plugins SDK project into your workspace. For these steps, we'll assume you haven't yet configured your Plugins SDK in Liferay IDE:

1. In Liferay IDE, go to *File → New → Project… → Liferay → Liferay Project from Existing Source*.

   You can invoke the same wizard from the Liferay shortcut bar; just click the *New* button and select *Liferay Project from Existing Source*.

   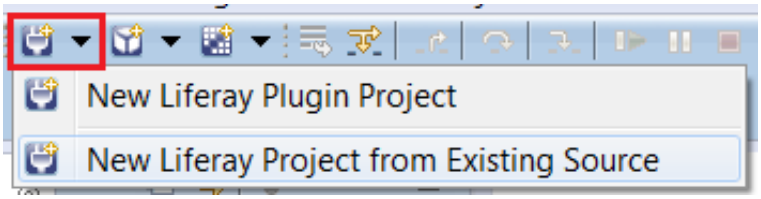

   Figure 2.12: Instead of clicking *File → New* to create a new Liferay project from an existing source, you can click the button shown above from Liferay IDE's shortcut bar.

2. In the *New Liferay Project* window, click the *Browse* button and navigate to the project folder of the plugin you'd like to import. It should be a subfolder of one of the SDK's plugin type folders (e.g., portlets, hooks, themes, etc) or you'll get an error message stating that your Liferay project location is invalid.

   On selecting the plugin project folder, the *Liferay plugin type* and *Liferay plugin SDK version* values are updated. If your Plugins SDK is outdated or you entered an incorrect project type, its field gets marked with an error.

3. Select the *Liferay target runtime* for the plugin project. If you don't have a Liferay Portal Runtime, use the *New…* button to create one now. For more detailed instructions, see the section *Liferay Portal Runtime and Server Setup*, found earlier in this chapter.

4. Click *Finish* to complete the import.

Any time you import a project into Liferay IDE, you can verify that it was successfully configured as a Liferay IDE project by using the process outlined in the section *Verifying Successful Project Import*, found later in this chapter.

Next, let's import multiple projects from a Liferay Plugins SDK you've already set up in Liferay IDE. You can use these steps:

1. In Liferay IDE, go to *File → Import… → Liferay → Liferay Projects from Plugins SDK*.

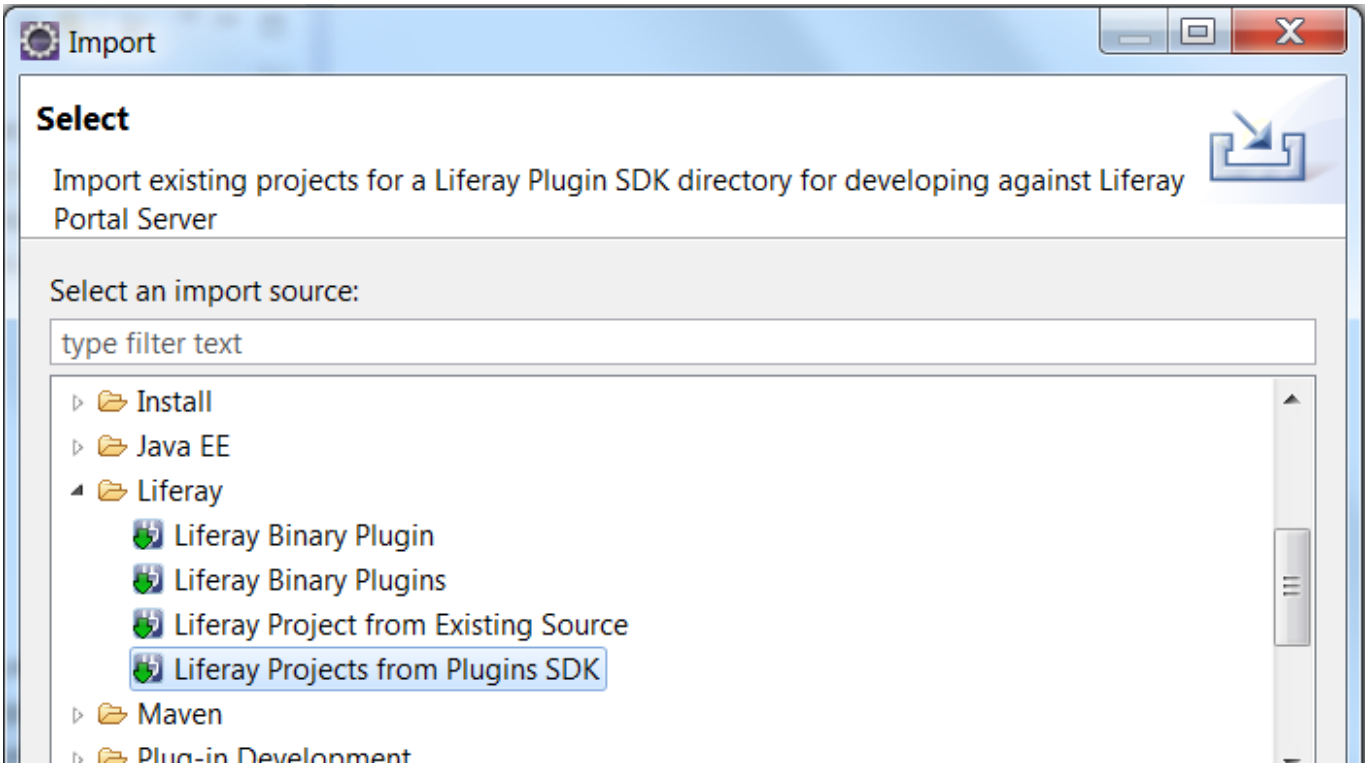

Figure 2.13: To import projects from a Plugins SDK, choose *Liferay Projects from Plugins SDK* from the Import menu.

2. In the *Import Liferay Projects* window, use the combo box to select the *Liferay Plugins SDK* from which you're importing plugins.

Note: If your SDK isn't configured in Liferay IDE (i.e., it's not in the dropdown list of the *Import Projects* window), use the *Configure* link to add one. To configure a Plugins SDK from the Installed SDKs window, just click *Add* and then browse to the Plugins SDK's root directory.

---

3. Once you select your Plugins SDK in the combo box, the *Liferay Plugin SDK Location* and *Liferay Plugin SDK Version* fields are automatically filled in, as long as they're valid. Invalid entries are marked with an error.

4. The list of projects that are available for import are displayed in a list. Any projects already in the workspace are disabled. Projects available for import have an empty check box; select each project you'd like to import.

5. Select the Liferay runtime for the imported projects. If you don't have a Liferay runtime, can add one now with the *New…* button.

6. Click *Finish*.

You've imported your plugins into your workspace! Next, we'll discuss a different scenario; converting existing Eclipse projects into Liferay projects.

Converting Existing Eclipse Projects into Liferay IDE Projects

The steps outlined in the previous section are for importing Liferay projects that aren't already in your Eclipse workspace. You can also import a non-Liferay project in your Eclipse workspace (i.e., you can see it in Eclipse's project explorer) and convert it to a Liferay project. Just follow the steps below.

1. Move the project into a Liferay Plugins SDK if it is not already in one.

2. In Eclipse's Project Explorer, right-click on the project and select *Liferay → Convert to Liferay plugin project*.

Note: If no convert action is available, either the project is already a Liferay IDE project or it is not faceted (i.e., Java and Dynamic Web project facets are not yet configured for it). For instructions on resolving these issues, see the section *Verifying Successful Project Import*, found later in this chapter.

3. In the *Convert Project* wizard, your project is selected and the SDK location and SDK version of your project is displayed.

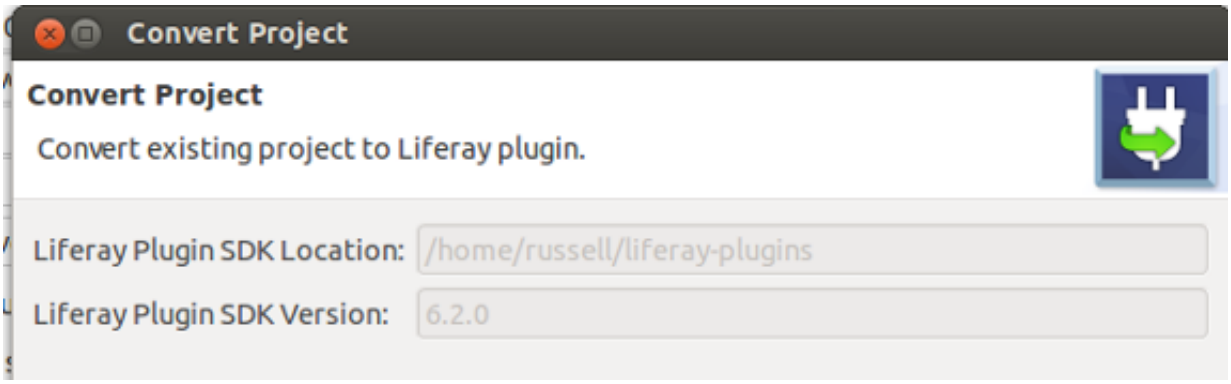

Figure 2.14: The *Convert Project* wizard detects your Plugin's SDK's the location and version.

4. Select the Liferay runtime to use for the project. If you don't have a Liferay Runtime defined, define one now by clicking *New…*.

5. Click *Finish*.

Let's verify the success of your imports and ensure that they're properly configured as Liferay IDE projects.

Verifying Successful Project Import

After importing projects into Liferay IDE, you'll want to verify that they imported successfully, and that they're properly configured as Liferay IDE projects. Here's how you verify that your imports were successful:

1. Once the project is imported, you should see a new project inside Eclipse and it should have an "L" overlay image; the "L" is for Liferay!

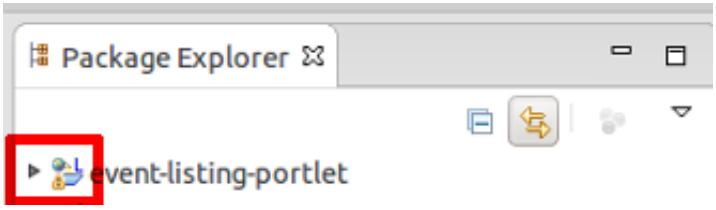

Figure 2.15: Look for an "L" overlay image to verify that the import succeeded.

2. Check the project's target runtime and facets to make sure it's configured as a *Liferay IDE* project:

2.1. In the *Package Explorer*, right click → *Properties* → *Targeted Runtimes*.

2.2. In the *Properties* window, click *Project Facets* and make sure that the Liferay plugin facets are properly configured.

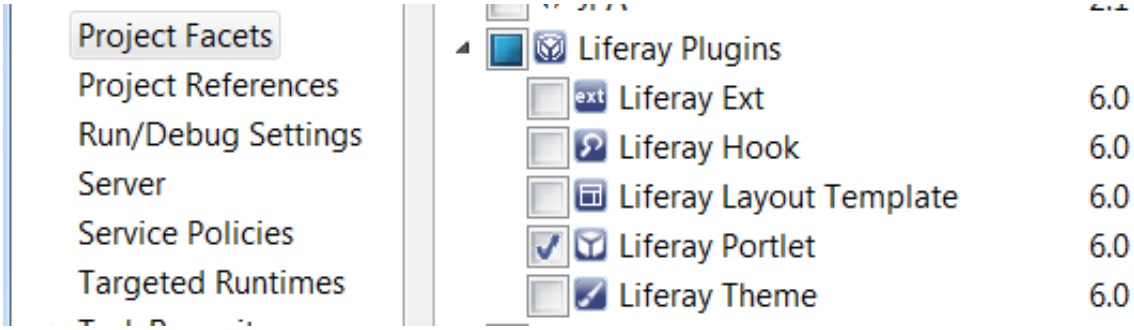

Figure 2.16: Make sure that your project's Liferay plugin facets are properly configured.

Great! You've confirmed that your import was successful; you can now make revisions to your configured Liferay IDE project. Next, let's explore Liferay IDE's Remote Server Adapter feature.

Using Liferay IDE's Remote Server Adapter

The *Remote Server Adapter* is a feature that lets you deploy your Liferay projects to a remote Liferay Portal server; it first became available in Liferay IDE 1.6.2. Let's talk about when to use the Remote Server Adapter, then we'll cover setting it up and using it in more detail.

Your remote Liferay Portal instance needs to satisfy two requirements to use a Remote Server Adapter:

- It is version 6.1 or later.
- It has the Remote IDE Connector application installed from Liferay Marketplace. The Remote IDE

Connector contains the `server-manager-web` plugin for Liferay that provides an API for Liferay IDE's Remote Server Adapter to use for all its remote operations.

The Remote Server Adapter lets developers deploy local projects to a remote development server for testing purposes–this is its primary use case. If you're using Liferay IDE and want to deploy projects to a remote server, just make sure you have access to a remote server with the Remote IDE Connector application installed. It's possible to install the Remote IDE Connector application on a production server, but it creates an unnecessary security risk, so we don't recommend it. Clients shouldn't update, or hot-fix, remotely deployed plugins with the adapter; the portal system administrator should use normal mechanisms to apply plugin updates and fixes.

To start deploying projects to a remote server, you'll need to download and install the following resources on your local development machine:

- Download Liferay IDE from Liferay's downloads page or download Liferay Developer Studio from the Customer Portal.
- Download Liferay Portal CE or EE to your local development machine.

You'll need to download Liferay Portal CE or EE to your remote (test) server as well.

Our demonstration uses the Remote Server Adapter on Liferay Portal bundled with Apache Tomcat, but you can use the adapter with Liferay Portal running on any application server Liferay Portal supports. Install Liferay Portal locally to compile the plugins you develop. Install Liferay Portal on your remote test server to host the plugins you'll deploy to it.

Important: Keep a record of your portal administrator login credentials (e.g., username/password) for your remote Liferay server; you'll need them to configure your connection from Liferay IDE to the remote Liferay server.

Let's start by configuring the Remote Server Adapter.

Configuring the Remote Server Adapter

You can use Liferay IDE's Remote Server wizard to configure the Remote Server Adapter and install the Remote IDE Connector to your remote Liferay instance. Alternatively, you can install the Remote IDE Connector to your remote Liferay instance before configuring Liferay IDE's Remote Server Adapter. To configure the Remote Server Adapter, use the following steps:

1. Start your remote Liferay Portal instance and verify that you can log in as an administrator.

2. Launch Liferay IDE and open the new server wizard by clicking *File → New → Other*; select *Server* in the

Server category and click *Next*. Select *Remote Liferay Server (Liferay 6.2)* in the Liferay, Inc. category.

3. Enter the IP address of the machine with the remote Liferay Portal instance into the *Server's host name* field. For the *Server name*, enter *Liferay@\[IP address]*, then click *Next*.

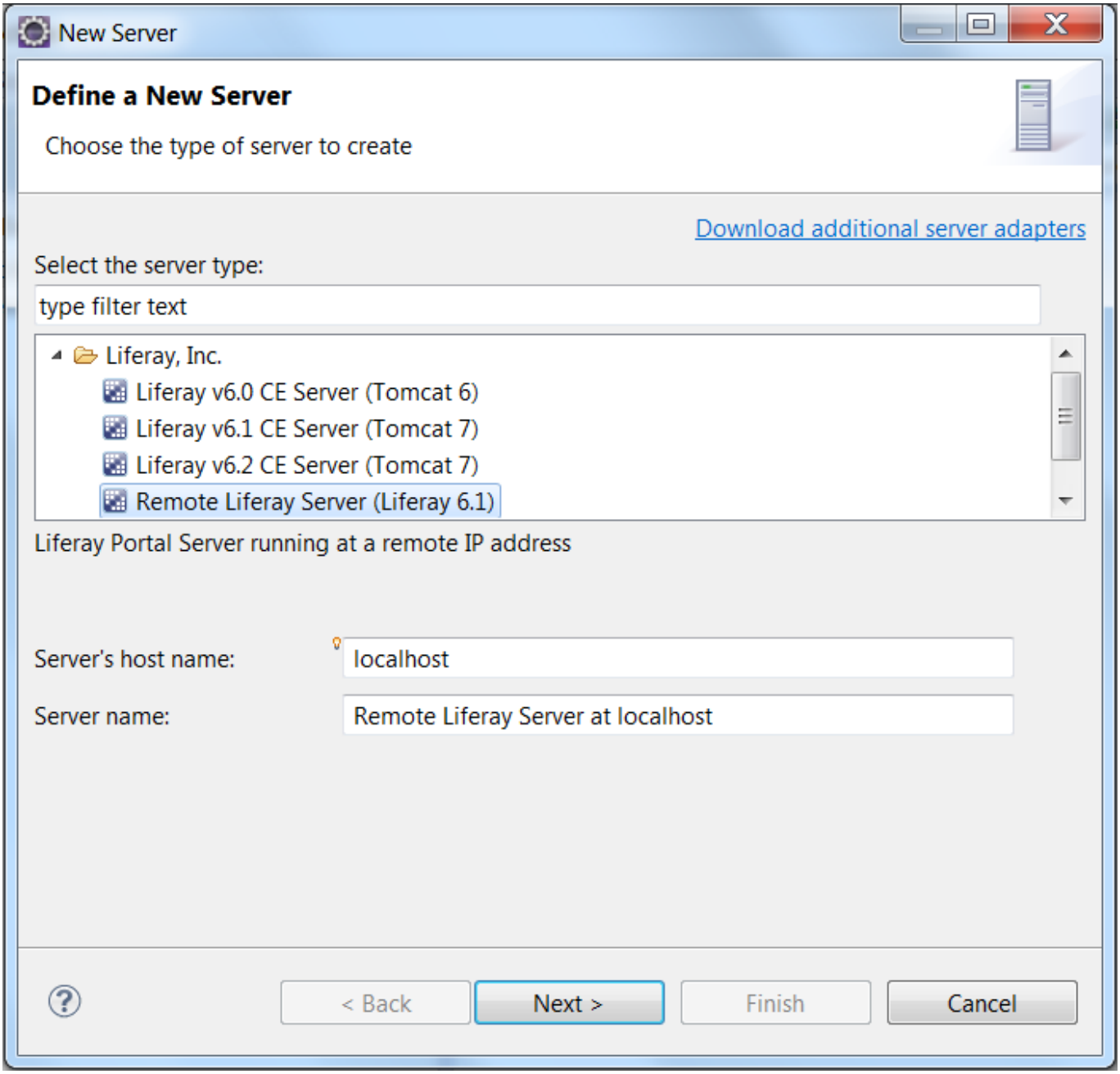

Figure 2.17: Configure the remote Liferay server's information.

4. The New Server wizard's next page directs you to define the Liferay Portal runtime stub. Doing so allows projects created for your remote server to use the runtime stub for satisfying JAR dependencies needed to compile various Liferay projects. Select the *Liferay bundle type* based on the version of your local Liferay bundle, browse to the *Liferay bundle directory*, and click *Next*.

5. On the next page of the wizard, configure your connection to your remote Liferay instance:

    - **Hostname:** Enter the IP address of your remote Liferay Portal instance's machine.
    - **HTTP Port:** Enter the port it runs on (default: 8080).
    - **Username** and **Password:** Enter your administrator credentials for the remote Liferay Portal instance.

Leave the *Liferay Portal Context Path* and *Server Manager Context Path* set to the defaults unless these values were changed for your remote Liferay Portal instance.

6. Your remote Liferay Portal instance needs the Remote IDE Connector application installed; otherwise, Liferay IDE can't connect to it. If you haven't installed Liferay IDE Connector yet, click the *Remote IDE Connector* link in the wizard. If you already downloaded the Remote IDE Connector application and installed it to your remote portal, skip to the next step and validate your connection.

   Browse Liferay Marketplace for the Remote IDE Connector application. When you've found it, click *Free* to purchase it. Follow the on-screen prompts.

   Once you've purchased the application, navigate to the *Purchased* page of the Control Panel's Marketplace interface.

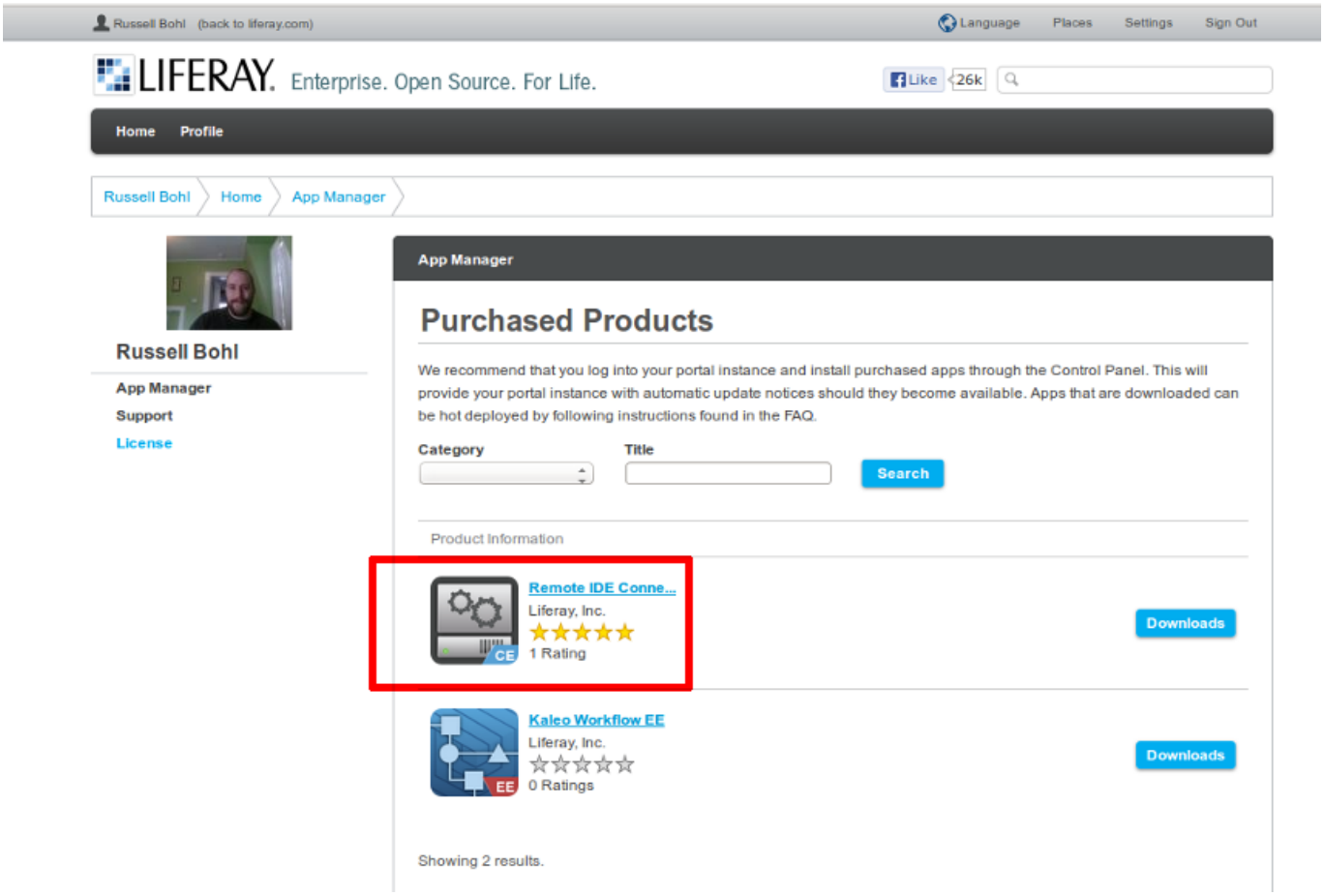

   Figure 2.18: Click *Purchased* in the Marketplace section of the Control Panel to download and install the Remote IDE Connector application that you purchased.

   Find your application in the list of purchased products. Then click on the buttons to download and install the application. Once it's been installed on your remote portal, return to the Remote Liferay Server configuration wizard in Liferay IDE.

7. Click the *Validate Connection* button; if no warnings or errors appear. your connection works! If you get any warning or error messages in the configuration wizard, check your connection settings.

8. Once Liferay IDE is connected to your remote Liferay Portal instance, click *Finish* in the Remote Liferay Server configuration wizard.

After you click *Finish*, the new remote server appears in Liferay IDE's *Servers* tab. This tab appears in the bottom left corner of the Eclipse window if you're using the Liferay perspective. If you entered your connection settings correctly, Eclipse connects to your remote server and displays the remote Liferay Portal instance's logs in the console. If your remote server is in debug mode, the Eclipse Java debugger is attached to the remote process automatically.

9. You can change the remote server settings at any time. Double-click on your remote server instance in the *Servers* tab to open the configuration editor, where you can modify the settings.

Now that your remote Liferay Portal server is configured, let's test the remote server adapter!

Using the Remote Server Adapter

Once your remote Liferay Portal server is correctly configured and your local Liferay IDE is connected to it, you can begin publishing projects to it and using it as you would a local Liferay Portal server.

Here's how to publish plugin projects to your remote server in Liferay IDE:

1. Right click on the server and choose *Add and Remove…*.

Note: Make sure you have available projects configured in Liferay IDE. If not, you'll get an error message indicating there are no available resources to add or remove from the server.

2. Select the Liferay projects to publish to your remote server; click *Add* to add them to your remote server, then click *Finish*. Deployment begins immediately. If publication to the remote server was successful, your console displays a message that the plugin was successfully deployed!

3. As you make changes to your plugin project, republish them so they take effect on the remote server. To set your remote server's publication behavior, double click your remote server in the *Servers* tab. You can choose to automatically publish resources after changes are made, automatically publish after a build event, or never to publish automatically. To manually invoke the publishing operation after having modified project files, right click on the server in the Servers view and select *Publish*.

Next, let's examine the structure of Liferay's Plugins SDK. We'll also learn how to use it independently of Liferay IDE.

Average (0 Votes)

Previous - Working with...

Table of Contents »
Working with Liferay's Developer Tools »
Developing Apps with Liferay IDE

Next - Leveraging the Plugins SDK

ACCESS DENIED