

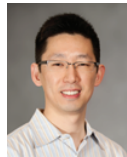
CONNECT WITH OUR FEATURED SPEAKERS



BRIAN CHAN
 Founder & Chief
 Software Architect



BRYAN CHEUNG
 Founder & Chief
 Executive Officer



BRIAN KIM
 Founder & Chief
 Operating Officer



PAUL HINZ
 Chief Marketing
 Officer



NATE CAVANAUGH
 Director of
 UI Engineering











JORGE FERRER
 Senior Software
 Architect



RAYMOND AUGÉ
 Senior Software
 Architect



JAMES FALKNER
 Liferay Community
 Manager

-  **EAST COAST SYMPOSIUM**
 Greater Washington DC | May 10-11, 2011
-  **HUNGARY SYMPOSIUM**
 Budapest | May 26, 2011
-  **FRANCE SYMPOSIUM**
 Paris | June 15, 2011
-  **WEST COAST SYMPOSIUM**
 Los Angeles, CA | September 2011
-  **SPAIN SYMPOSIUM**
 Madrid | October 4-5, 2011
-  **EUROPE SYMPOSIUM**
 Frankfurt | October 18-19, 2011
-  **ITALY SYMPOSIUM**
 Roma | November 18, 2011
-  **INDIA SYMPOSIUM**
 Bangalore | TBA - Coming Soon!

*Dates are subject to change.

 **LEARN MORE ABOUT LIFERAY**
WWW.LIFERAY.COM/EVENTS/LIFERAY-SYMPOSIUMS

CONTENTS INCLUDE:

- Starting the Development
- Portlet Development with Vaadin
- Tools for Vaadin Development
- Composing the User Interface with Vaadin
- Theming Vaadin Applications
- Inter-Portlet Communication (IPC)

Mastering Portal UI Development With Vaadin and Liferay

By Sami Ekblad, James Falkner

The open source Liferay Portal has become a popular way of implementing enterprise websites. Providing an integrated platform for application development and deployment, Liferay has also become an environment for running business applications. For application development, Liferay Portal includes Vaadin as a pre-packaged framework for developing attractive, easy-to-use applications.

About this Refcard

This Refcard gives a quick overview of the user interface development with Vaadin on Liferay. It covers topics like portlet setup, configuration, inter-portlet communication (IPC), UI composition, and theming. To get a more general understanding of Liferay Portal and Vaadin framework, see the Refcards "Liferay Essentials" and "Vaadin: A Familiar Way to Build Web Apps With Java".

STARTING THE DEVELOPMENT

Strategies for Portal User Interface

Portlets are small web applications written in Java. They run in a piece of a web user interface within a portal. Portal manages the lifecycle and aggregation of portlets to a single visible web page. When designing a user interface for a portal, there are a few strategies based on UI granularity:

Strategy	Description
Small generic portlets communicating with each other	Small user interface, very generic and portal-wide functions: <ul style="list-style-type: none"> • Light-weight, custom user interface • Requires more inter-portlet communication • Fine-grained portlets for portal-wide reuse • Typically small, public applications like search boxes, shopping carts
Integrated application developed as a single portlet	Leverage Liferay as an application platform for a business application: <ul style="list-style-type: none"> • Reuse an existing application user interface or create an application that can run also as standalone application. • Complete, integrated user interface • Rich desktop-like user experience

portal there are few strategies based on UI granularity:

Naturally, it is possible to have a mixture of these and use different approaches to meet the usability requirements.

Available UI frameworks

Liferay supports a number of web frameworks for development of portlet user interfaces. Which you should use depends on your background as well as the strategy you choose for your application.

Framework	Description	Programming Languages
Alloy UI	Rich client-side JavaScript/CSS framework based on YUI Library.	JavaScript, JSP

JavaServerFaces	Server-side user interface component framework based on JSP and tag libraries.	Java, XML, JSP
Spring MVC	Action oriented Model-View-Controller framework for web pages.	Java, XML, JSP
Struts 2	Action oriented Model-View-Controller framework for web pages.	Java, XML, JSP
Vaadin	A rich Java-only component framework based on Ajax/GWT	Java
Apache Wicket	Server-side component framework based on Java and HTML.	Java, HTML

Different portlets can use different frameworks to implement the user interface.

PORTLET DEVELOPMENT WITH VAADIN

Vaadin is a server- and component-oriented user interface framework for Java web applications. Vaadin applications can be hosted as standalone web applications as well as portlets in portals like Liferay. Vaadin is a good choice for building complete applications that use Liferay as a platform.

Portlets created with Vaadin are essentially Ajax web applications that can be considered single-page applications. This means that the page is not reloaded after it is opened initially; rather, it communicates user interaction with the server through Ajax communications.

Along with the desktop-like user experience, Vaadin provides all the typical features of a web framework, such as deep-linking and back-button support.

TOOLS FOR VAADIN DEVELOPMENT

Since Liferay 6.x, there have been several tools to help you in developing portlets with Vaadin. These tools are meant to simplify

from \$99 per month

Vaadin Pro Account

Support from the Vaadin team

Pro Add-on components and tools

Bug fix guarantee, feature voting and knowledge base

Try one month for free with code: **DZONE**

vaadin.com/pro

the creation of portlets and help portal administrators maintain the system.

Liferay Plugins SDK

The Liferay Plugins SDK is a development environment that helps in the development of portlets. This development environment is command-line-based and relies on the Apache Ant (though you may also use Maven) and allows development of all types of Liferay plugins.

The Plugins SDK is both a project skeleton generator and a location where your projects are stored. You can download the Plugins SDK from <http://liferay.com/downloads/liferay-portal/additional-files>.

To get started using the Plugins SDK, refer to the Refcard "Liferay Essentials: A Definitive Guide for Enterprise Portal Development" at <http://refcardz.dzone.com/refcardz/essential-liferay-leading-open>.

Liferay IDE

Liferay IDE is an extension for the Eclipse IDE that adds support for the development of plug-in projects for the Liferay Portal platform. Since version 1.2, the Liferay IDE has supported Vaadin by offering wizards for creating portlet plugin projects. Up-to-date information about Liferay IDE can be found at <http://www.liferay.com/community/wiki/-/wiki/Main/Liferay+IDE>.

The Vaadin Plugin for Eclipse can also be used with the Liferay IDE to give developers the ability to easily create Vaadin+Liferay projects and visually compose Vaadin components and portlets for use within Liferay.

Vaadin Control Panel for Liferay

The Vaadin Control Panel for Liferay gives portal administrators an interface to maintain the portal-wide Vaadin resources. You can use it to:

- Check and update the Vaadin libraries in portal
- Recompile the Vaadin widgetset when installing new Vaadin Add-ons.

You can access the Control Panel in Liferay after logging in as an administrator at **Manage > Control Panel > Portal > Vaadin**.

The latest version of the control panel is available at <http://vaadin.com/addon/vaadin-control-panel-for-liferay>.

COMPOSING THE USER INTERFACE WITH VAADIN

With Vaadin, the user interface is built from user interface components. They are server-side Java classes that implement a single UI control such as a button, select, or a layout.

With layout components, you can compose larger components that hierarchically build up the application UI.

Vaadin Application

A Vaadin application is defined in a class that extends the `com.vaadin.Application`. This is the class that you should define as the 'application' init-param in portlet.xml as described in later sections.

A new instance of this class is created when a new user comes to portal view where the portlet resides.

Here is the code for a minimal Vaadin application:

```
package org.vaadin.sample;

import com.vaadin.Application;
import com.vaadin.ui.Label;
import com.vaadin.ui.Window;

public class MyApplication extends Application {
    @Override
    public void init() {
        Window w = new Window();
        w.addComponent(new Label("Hello Liferay!"));
        setMainWindow(w);
    }
}
```

Vaadin UI Components

Vaadin Framework includes over 60 stock components. You can find a rapidly growing number of open-source and commercial add-on components at <http://vaadin.com/directory>.

Furthermore, you can extend Vaadin by creating new components with the Google Web Toolkit (GWT). GWT is an open-source Java-to-JavaScript compiler that allows you to build client-side features without JavaScript. See additional information at <http://code.google.com/webtoolkit/>.

You can find all the components in the Java package `com.vaadin.ui`. Add-ons may use their own package naming, but it is typical that they start with `org.vaadin`.

TIP: You can test and try different Vaadin components online at <http://demo.vaadin.com/sampler>. All the demos include source code and documentation.

User Interface Layout

Start by creating a main **Window** for your application and putting the initial content in there. The user interface structure is a hierarchy of nested layouts and components. Here is an example of a simple user interface hierarchy:

```
MyApplication
  Window
    VerticalLayout
      TextField
      TextField
      Button
```

The above UI could be created in Java as follows:

```
Window w = new Window("Subscribe Newsletter");
setMainWindow(w);
w.setContent(new VerticalLayout());

TextField name = new TextField("Name");
TextField email = new TextField("Email");
Button subscribeBtn = new Button("Subscribe");

w.addComponent(name);
w.addComponent(email);
w.addComponent(subscribeBtn);
```

TIP: You should avoid creating too deeply nested layout structures. In particular, older browsers can become slow. Instead, use the `CustomLayout`, `GridLayout`, or some lightweight layouts like the `CSSLayout`.

User Interface Events

Vaadin is an event-based framework. You can receive user-triggered events in your application by registering a listener for it. Here is an example for `Button.ClickEvent`:

```
subscribeBtn.addListener(new Button.ClickListener() {
    @Override
    public void buttonClick(ClickEvent event) {
        // ...
    }
});
```

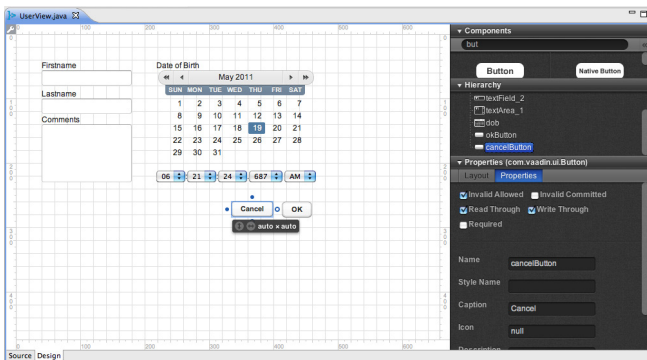
Event listeners are executed in the server side synchronously. You can fetch data and update the user interface by adding and removing components.

TIP: Good practice for event listeners is to only call your Java control code and let them do the UI updates. This is better object-oriented design, and it enhances readability of your Java code.

Vaadin Visual Editor

The visual editor is part of the Vaadin Plugin for Eclipse and is available at <http://vaadin.com/eclipse>. It includes a WYSIWYG editor for defining a CustomComponent; that is, UI composites in Vaadin.

The visual editor generates the Java code that you can continue to modify and extend.



The visual editor runs inside the Eclipse IDE, giving developers a quick way of creating user interface without writing the code itself.

To activate the visual editor, create a new component using the Eclipse wizard: **New > Vaadin CustomComponent (composite)**, open the file with the right editor **Open with > Vaadin Editor**, and choose **Design** tab.

Every time you save the file in the Design mode, the Visual Designer generates the Java code that makes up the UI.

Note: The Visual editor works with the "reindeer" theme, but you can change the theme in your **Application** class by calling the `setTheme` method. For example:

```
myApplication.setTheme("liferay").
```

THEMING VAADIN APPLICATIONS

Vaadin is designed to support parallel work of application developers and graphic designers by strongly separating the graphical elements from the functionality.

All Vaadin applications have an associated theme. Themes are essentially a collection of CSS and images that define the look and feel of the Vaadin's user interface components.

The following Vaadin themes are included in Liferay by default:

Theme	Description
base	Base theme for creating your own customized theme. Handles most of the cross-browser issues.
liferay	A Liferay 6 look-a-like theme. Use this to create applications that match the Liferay 6 default styles.
reindeer	Reindeer is default look and feel of Vaadin. It provides minimalistic, but stylish look for business applications.
runo	More colorful and rounded theme for web applications.

Structure of a Vaadin Theme

Vaadin themes are located in the themes folder of the portal. They are a collection of CSS and images that give the Vaadin components their look and feel.

The theme folder must contain the **styles.css** stylesheet, and custom layouts must be placed in the layouts sub-folder. Other contents may be named freely.

A typical Vaadin theme follows the structure under the theme folder:

Theme	Description
styles.css	The CSS for the whole theme.
layouts/	Directory for CustomLayout definition files.
<component>/	CSS definitions for a single UI component. Only used to split the CSS for easier maintenance. These are compiled into styles.css as is a single CSS file.
<component>/img/	Static image resources for the component.

Typically, you start to develop your theme by inheriting some existing -theme:

```
@import url(../liferay/styles.css);
```

After that, you can apply the CSS rules that override the original theme without completely rewriting a theme.

To activate the theme in your portlet, add the following to the `init` method of your application:

```
public void init() {
    setTheme("mytheme");
    // ...
}
```

CSS Classnames in Vaadin

To maximize the use of theme inheritance and to help customize components, the CSS class selectors in Vaadin are defined the following scheme **.v-<component|item>**. All style names are lowercase.

As an example, the following CSS rules change the color of all captions and adds borders to all **TextFields**:

```
.v-caption {color: red;}
.v-textfield {border: 1px solid red;}
```

The most relevant CSS class names are:

Class Name	Description
.v-app	The top-level DIV container for the whole application.
.v-window	Container for the application window.
.v-<component>	Container for a specific component type. Note that captions are managed outside the component, by the containing Layout.

To avoid style leakage outside the Vaadin application, it is recommended that you use the most specific CSS selector when applying your own styles and limit them by using container, such as:

```
.v-app .v-caption {color: green; }
```

VAADIN PORTAL-WIDE SETTINGS

The core Vaadin Framework consists of a single jar file that includes the framework itself along with the core components. This jar along with the CSS themes and custom widgets are installed to the portal itself, and they are shared by all Vaadin-based portlets. This means that only a single version of Vaadin is supported in a portal installation.

A Liferay 6 installation includes the following Vaadin-related files and directories:

Class Name	Description
vaadin.jar (Java jar-file)	Vaadin Framework, portlet integration and core UI components.
widgetsets/<name>/ (directory)	Client-side widgets of Vaadin. JavaScript compiled with Google Web Toolkit (GWT). Must be publicly accessible.
theme/<theme name>/ (directory)	Collection of CSS and static images that define the look of the Vaadin components. Must be publicly accessible.
<add-on>-<version>.jar (Java jar-file)	An extension to Vaadin - new UI component, data-binding or a theme. Standard jar file.
gwt-user.jar, gwt-dev.jar (Java jar-files)	Google Web Toolkit libraries needed to re-compile the client-side JavaScript if new components are imported.

Depending on the application server used, these files are installed in different locations.

Tomcat 6.x	Location
Global vaadin.jar	\$(TOMCAT_DIR)/webapps/ROOT/WEB-INF/lib
Vaadin Add-ons	\$(TOMCAT_DIR)/webapps/ROOT/WEB-INF/lib
Vaadin CSS Themes	\$(TOMCAT_DIR)/webapps/ROOT/html/VAADIN/themes
Vaadin Client-side Widgetset	\$(TOMCAT_DIR)/webapps/ROOT/html/VAADIN/widgetset
GWT jar-files (only needed for compiling widgetset)	\$(TOMCAT_DIR)/webapps/ROOT/WEB-INF/vaadin/gwt

GlassFish 3.x	Location
Global vaadin.jar	\$(GLASSFISH_DOMAIN_DIR)/applications/j2ee-modules/Liferay-portal/WEB-INF/lib
Vaadin Add-ons	\$(GLASSFISH_DOMAIN_DIR)/applications/j2ee-modules/Liferay-portal/WEB-INF/lib
Vaadin CSS Themes	\$(GLASSFISH_DOMAIN_DIR)/applications/j2ee-modules/Liferay-portal/VAADIN/themes
Vaadin Client-side Widgetset	\$(GLASSFISH_DOMAIN_DIR)/applications/j2ee-modules/Liferay-portal/VAADIN/widgetset
GWT jar-files (only needed for compiling widgetset)	\$(GLASSFISH_DOMAIN_DIR)/applications/j2ee-modules/Liferay-portal/WEB-INF/vaadin/gwt/

JBoss 5.x	Location
Global vaadin.jar	\$(JBOSS_INSTANCE_DIR)/deploy/ROOT.war/WEB-INF/lib
Vaadin Add-ons	\$(JBOSS_INSTANCE_DIR)/deploy/ROOT.war/WEB-INF/lib

Vaadin CSS Themes	\$(JBOSS_INSTANCE_DIR)/deploy/ROOT.war/VAADIN/themes
Vaadin Client-side Widgetset	\$(JBOSS_INSTANCE_DIR)/deploy/ROOT.war/VAADIN/widgetsets/
GWT jar-files (only needed for compiling widgetset)	\$(JBOSS_INSTANCE_DIR)/deploy/ROOT.war/WEB-INF/vaadin/gwt

Liferay Portlet Setup

To use Vaadin in a Liferay portlet, the portlet has to be configured to use Vaadin and optional add-on libraries by creating and/or editing various configuration files.

Anatomy of a Portlet Project

Portlets (Vaadin and non-Vaadin) are built as Liferay plugins, which can be compiled and hot-deployed into a Liferay environment. In their source (uncompiled) form, there are several file and directory structures used to manage the project.

Folder	Description
WebContent/ (or docroot/)	This folder is the "root" of your Vaadin portlet application
WEB-INF	Standard WEB-INF folder for web applications. Also contains Liferay-specific descriptors such as portlet.xml, liferay-portlet.xml, and others.
WEB-INF/src	Java source code for the Vaadin Portlet
build.xml	ANT build script controlling building and deploying
liferay-display.xml	Describes the category under which the portlet should appear in the Liferay UI

liferay-plugin-package.properties	Describes properties used by Liferay's hot deploy mechanism, most notably which Vaadin dependencies to include when compiling the plugin.
liferay-portlet.xml	Describes Liferay-specific portlet enhancements (akin to portlet.xml for generic portlets). There are many settings here to customize your portlet.
portlet.xml	Standard JSR-168 or JSR-286 portlet descriptor, including settings for performing non-Vaadin portlet IPC.
web.xml	Standard Web Application descriptor. You should not need to edit this for use with Vaadin.

Liferay Plugin Package Properties

The **liferay-plugin-package.properties** file defines a number of settings for the portlet, most importantly the Vaadin Framework and Vaadin Add-on jar-files to be used.

The following example of a dependency definition:

```
name=MyVaadinPortletName
module-group-id=vaadin
module-incremental-version=1
tags=
short-description=
change-log=
page-url=http://www.liferay.com
author=Your Company, Inc.
licenses=LGPL
portal-dependency-jars=\
    vaadin.jar,\
    paperstack-0.8.1.jar,\
    console-1.0.0.jar
```

The Vaadin-related portlet dependencies are highlighted. The

vaadin.jar contains the framework itself. The other dependencies, **paperstack-0.8.1.jar** and **console-1.0.0.jar**, are Vaadin add-ons used by the portlet.

Refer to the application server setup to see where these jar dependencies should be installed to work at compiletime and runtime.

Portlet Descriptor

To wire the portlet to your Vaadin application class, configure portlet mapping in the **portlet.xml**:

```
<portlet>
  <portlet-name>MyVaadinPortlet</portlet-name>
  <display-name>MyVaadinPortlet</display-name>
  <portlet-class>
    com.vaadin.terminal.gwt.server.ApplicationPortlet2
  </portlet-class>
  <init-param>
    <name>application</name>
    <value>org.vaadin.sample.MyApplication</value>
  </init-param>
</portlet>
```

Vaadin portlets always use the same portlet class, **com.vaadin.terminal.gwt.server.ApplicationPortlet2**, and the actual application is defined as an **init-param**.

Liferay Portlet Descriptor

Liferay also requires a **liferay-portlet.xml** descriptor file that defines Liferay-specific parameters. In particular, Vaadin portlets must be defined as **"instanceable"** but not as **"ajaxable"**:

```
<liferay-portlet-app>
  <portlet>
    <!-- Matches definition in portlet.xml. -->
    <!-- Note: Must not be the same as servlet name. -->
    <portlet-name>Portlet Example portlet</portlet-name>

    <instanceable>true</instanceable>
    <ajaxable>false</ajaxable>
  </portlet>
</liferay-portlet-app>
```

This is because Vaadin portlets handle the Ajax requests internally without Liferay's Ajax mechanisms.

Liferay Portlet Display Descriptor

The **liferay-display.xml** file defines the portlet category under which portlets are located in the **Add Application** window in Liferay. Without this definition, portlets will be organized under the "Undefined" category.

The following puts the application in a new category called "Vaadin":

```
<display>
  <category name="Vaadin">
    <portlet id="MyVaadinExamplePortlet" />
  </category>
</display>
```

For more information on these and other optional descriptors, see the Chapter 11.8 of the "Book of Vaadin" at <http://vaadin.com/book> and refer to the Liferay Developer Guide at <http://liferay.com/documentation>

INTER-PORTLET COMMUNICATION (IPC)

Liferay offers different IPC mechanisms to allow portlets

communicate with each other. The following table summarizes the different IPC methods in Liferay:

Method	Description
JSR 286 Portlet Events	Standard portlet communication mechanism. Requires page reload.
JavaScript	Traditional client/server communication, using client-side JavaScript, calling other portlets running in the same page using Liferay's client-side JavaScript API: <code>Liferay.fire(eventName, data)</code> <code>Liferay.on(eventName, function, [scope])</code>
Vaadin Addon for Liferay IPC	Mechanism for sending and receiving events between Vaadin and non-Vaadin portlets.
Custom Event Bus	Direct client-side communication between portlets (e.g. using <code>OpenAjax.Hub</code>). No page refresh necessary, and no server communication is required.
Ajax Push (Reverse Ajax)	Typically used for server->client notifications (for example, in-browser chat). Long-held connections are used to push data from server to client, as needed, instead of separate communications for each message.

IPC in Vaadin Portlets

Vaadin portlets are based on Ajax communication that is most useful if the user never changes the page in the browser. In this scenario, the application talks to the server frequently and only small user interface updates are sent to the browser. This makes the best user experience.

When communicating with other portlets in a portal, the different scenarios may require different approaches to optimize the user experience.

A Vaadin portlet sending an event to a non-Vaadin portlet.

Depending on the other portlet, this typically requires a page reload. Below is an example of sending a "date" event to another portlet.

Configure an event definition in the portlet.xml:

```
<event-definition>
  <qname xmlns:vaadin="http://vaadin.com/portlet-events">vaadin:date</qname>
  <value-type>java.util.Date</value-type>
</event-definition>
```

Send a portlet event from a Vaadin application:

```
((PortletApplicationContext2) getApplication().getContext())
    .sendPortletEvent(getMainWindow(),
        new QName("http://vaadin.com/portlet-events",
            "date"),
        (Date)dateField.getValue());
```

Receive an event in a non-Vaadin portlet:

```
Receiving an event in a non-Vaadin portlet:
public class MyPortlet extends GenericPortlet

    @Override
    public void processEvent(EventRequest request,
        EventResponse response)
        throws PortletException, IOException {
        Event e = request.getEvent();
        if ("date".equals(e.getName())) {
            Date date = (Date) e.getValue();
            // ...
        }
    }
}
```

This style of IPC relies on server-side processing of the events and, therefore, requires a page reload to see the effects of the event in the non-Vaadin portlet.

Communicating with a Vaadin-based portlet using Ajax. In this

case, the full page request is not needed.

Note: This requires the Liferay IPC add-on to be installed in the application. For more details and instructions, go to: <http://vaadin.com/addon/vaadin-ipc-for-liferay>.

Receive an event in a Vaadin application/portlet:

```
LiferayIPC ipc = new LiferayIPC();
ipc.addListener("uniqueEventId", new LiferayIPCEventListener() {
    public void eventReceived(LiferayIPCEvent event) {
        // Process event here
    }
});
```

Send an event from another portlet using JavaScript:

```
LiferayIPC ipc = new LiferayIPC(); ipc.sendEvent("uniqueEventId", "payloadData");
```

Receive an event from a Vaadin application/portlet using JavaScript:

```
<script>
Liferay.on("uniqueEventId",
    function(event) {
        alert(event);
    }
);
</script>
```

Send an event from another portlet using JavaScript:

```
<script>
Liferay.fire("uniqueEventId", "someData");
</script>
```

This method is not suitable for sending a large amount of data, rather, it's for notifying the portlet that something has updated. The actual data should be shared using the database, files, or some external storage.

Note: When sending events to non-Vaadin portlets that are ajax-enabled (ajaxable set to true and render-weight < 1), be aware that if a portlet takes some time to load, it might not receive the event in the case that the event is sent before the portlet is fully initialized.

Further Information

For up-to-date and in-depth information, refer to the Liferay official documentation for Liferay at www.liferay.com/ and Vaadin documentation at vaadin.com/book/.

ABOUT THE AUTHOR



Sami Ekblad is one of the original authors of the Vaadin framework. Working in web application development since 1998, he now works as Partner Manager at Vaadin Ltd to help professional web developers to get most out of the Vaadin framework and tools. He holds B.Sc. degree in Computer Science from the University of Turku.

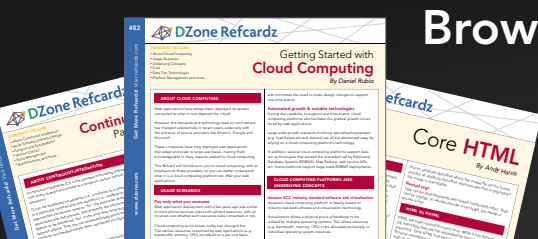


James Falkner is an open-source evangelist, community manager, and software developer working at Liferay, producers of the world's leading open source enterprise portal. In addition to Liferay, James has been active in a number of other open source products and projects, including the GlassFish Enterprise portfolio, Community/Social Equity, OpenSolaris, OASIS standards, and more. contributor and speaker at industry events such as JavaOne, JAX, and others.

RECOMMENDED BOOK



Liferay in Action is a comprehensive and authoritative guide to building portals on the Liferay 6 platform. Fully supported and authorized by Liferay, this book guides you smoothly from your first exposure to Liferay through the crucial day-to-day tasks of building and maintaining an enterprise portal that works well within your existing IT infrastructure. The book starts with the basics: setting up a development environment and creating a working portal. Then, you'll learn to build on that foundation with social features, tagging, ratings, and more. As the book progresses, you'll explore the Portlet 2.0 API, and learn how to create your own portlet applications.



Browse our collection of over 100 Free Cheat Sheets

Free PDF

Upcoming Refcardz
 NetBeans 7.0 Java Editor
 MySQL 5.5
 HTML 5 Canvas
 Android



DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
 140 Preston Executive Dr.
 Suite 100
 Cary, NC 27513
 888.678.0399
 919.678.0300
Refcardz Feedback Welcome
refcardz@dzone.com
Sponsorship Opportunities
sales@dzone.com



\$7.95