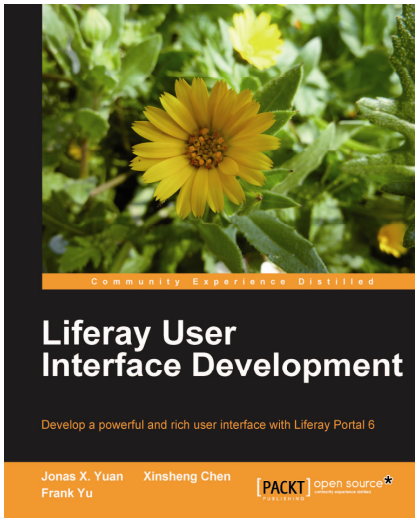
[PACKT] open source
community experience distilled
PUBLISHING

Liferay User Interface Development

Jonas X. Yuan
Xinsheng Chen
Frank Yu



Chapter No. 5
"Advanced Theme"

In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter NO.5 "Advanced Theme"

A synopsis of the book's content

Information on where to buy this book

About the Authors

Dr. Jonas X. Yuan is an expert on Liferay Portal and Content Management Systems (CMS). As an open source community contributor, he had published three Liferay books from 2008 to 2010. He is also an expert on Liferay integration with Ad Server OpenX, different search engines, enterprise contents including videos, audios, images, documents, and web contents, and other technologies such as BPM Intalio and Business Intelligence Pentaho, LDAP, and SSO. He holds a Ph.D. in Computer Science from the University of Zurich, where he focused on Integrity Control in Federated Database Systems. He earned his M.S. and B.S. degrees from China, where he conducted research on expert systems for predicting landslides. Previously, he has worked as a Project Manager and a Technical Architect in Web GIS (Geographic Information System). He is experienced in Systems Development Lifecycle (SDLC) and has deep, hands on skills in J2EE technologies. He has developed a BPEL (Business Process Execution Language) Engine called BPELPower from scratch in NASA data center. As the chief architect, Dr. Yuan led and successfully launched several large scale Liferay/Alfresco projects which are used by millions of users each month.

He has worked on the following books: Liferay Portal Enterprise Intranets, 2008, ISBN 13: 978-1-847192-72-1; Liferay Portal 5.2 Systems Development, 2009, ISBN 13: 978-1-847194-70-1; and Liferay Portal 6 Enterprise Intranets, 2010, ISBN 13: 978-1-849510-38-7.

I would like to thank all my team members at Liferay, specially Raymond Auge, Brian Chan, Bryan Cheung, Jorge Ferrer, Michael Young, Jerry Niu, Ed Shin, Craig Kaneko, Brian Kim, Bruno Farache, Thiago Moreira, Amos Fong, Scott Lee, David Truong, Alexander Chow, Mika Koivisto, Julio Camarero, Douglas Wong, Ryan Park, Eric Min, John Langkusch, Marco Abamonga, Ryan Park, Eric Min, John Langkusch, Marco Abamonga, Michael Han, Samuel Kong, Nate Cavanaugh, Arcko Duan, Richard Sezov, Joshua Asbury, Shuyang Zhou, and Juan Fernández for providing all the support and valuable information. Much thanks to all the friends in the Liferay community.

I sincerely thank and appreciate Leena Purkait and Hyacintha D'Souza, Project Coordinator and Development Editor respectively, at Packt Publishing, for criticizing and fixing my writing style. Thanks to Priya Mukherji and the entire team at Packt Publishing; it was really joyful to work with them.

Last but not least, I would like to thank my parents and my wife, Linda, for their love, understanding and encouragement. My special thanks to my wonderful and understanding kid Joshua.

Xinsheng Chen is an architect for Liferay portal projects, a computer game developer, and a software testing engineer. He holds an MS degree in Computer Science from California State University, San Bernardino. His focus was on online banking applications. He also has a bachelor's degree from Wuhan University, China. Mr. Chen was a QA engineer at VMware, Inc. He later led a team in developing four educational computer games for the Escambia County School District, Florida. He worked on Geographical Information Systems (GIS). Mr. Chen has rich experience in J2EE technologies. He has extensive experience in content management systems (CMS), including Alfreso. He is an expert in web portal technologies. Mr. Chen has hands-on experience in eight Liferay portal projects.

I would sincerely thank Leena Purkait, the Project Coordinator and Hyacintha D'Souza, the Development Editor at Packt Publishing. Thank you for reviewing my chapters. I appreciate your invaluable advice; it has helped me improve the quality of my writing. Also, I want to thank Priya Mukherji, Eleanor Duffy, and the team at Packt Publishing. It has been a happy experience working together with you!

I would also thank Dr. Jonas X. Yuan and Frank Yu for their friendship and encouragement along the way.

Frank Yu is a senior architect with 10 years of portal experience and co-founder of ForgeLife LLC, a San Francisco-based firm specializing in Liferay and CMS consulting and custom development. He manages multiple onshore and offshore teams to build portal and CMS applications. His team leadership, technical know-how, detail-oriented attitude, and result-driven approaches are highly valued by his clients.

Frank has extensive software engineering experience in Vignette-based and Liferay-based portal design, development, architecture, and project management, particularly in healthcare portal application development. He has hands-on experience of Liferay training, JSR 168, and JSR 286 portlets, portal themes and skins, the customization of portal frameworks, CMS, architecture, performance tuning, the administration of dozens of Amazon Elastic Compute Cloud (EC2) instances, and so on. He also has broad knowledge of system integration with multiple backends and major RDBMS and Java application servers. Previously, he worked on different portal products or applications at CIGNEX, InterComponentWare, Pay By Touch, and McKesson, a Fortune-14 healthcare company. Frank holds an M.S. degree in Computer & Information Sciences. He received his B.S. degree from Nanjing University, China.

I thank my co-authors, each of whom played a valuable role in ensuring that we were able to achieve coverage of a wide range of Liferay user interface feature set. It has been a great pleasure working with them.

I greatly appreciate the help from Milen Dyankov for reviewing my chapters. Thanks to Leena Purkait and Hyacintha D'Souza, the Production Coordinator and Development Editor respectively, at PACKT Publishing, for criticizing and editing my writing. Thanks also to Priya Mukherji and the entire team at PACKT Publishing.

I'm grateful for the many colleagues who have helped me over the years in different areas. Particularly, I would like to thank Alok Mathur, the CTO at Medicity, who hired me and introduced me to the portal technologies at McKesson in 2000.

I also thank my clients, my partners, and my team members across U.S. and China. It is they who make my projects successful and enjoyable. I thank the entire Liferay community, without them neither the project nor this book would be what it is today.

Last but not least, I would like to thank my wife, Yuting, for her continuing love and support, and for understanding that there is no separation between working time and spare time in the Liferay consulting world. My special thanks to my two wonderful daughters Marissa and Selina, who make me a proud father every single day.

Liferay User Interface Development

Liferay employs a specialized theming system, which allows you to change the look and feel of the user interfaces. As a developer, by using the right tools to create and manipulate themes with Liferay Portal, you can get your site to look any way you want it to. However, the Liferay theming system can be difficult to get started with. This practical guide provides you with a well organized manual for working with Liferay.

Liferay User Interface Development is a pioneer in explaining Liferay's powerful theming system by leading you through examples to get you to create your own themes as quickly as possible. It focuses on how portal pages are created and styled and also discusses some simple configuration and customization to change the look and feel of a portal page. Its explicit instructions are accompanied by plenty of source code. With the open source nature of Liferay, you will find a user-friendly environment to design themes with the latest user interface technologies.

Liferay User Interface Development unlocks the potential of using Liferay as a framework to develop a rich user interface.

The book starts off with how you should go about structuring a Liferay Portal web page. It identifies the components of a portal page: theme, layout, and portlets. This hands-on tutorial explains themes, portlets, and Alloy UI, which is the latest output from the Alloy Project of Liferay, in an easy-to-understand way. It covers all aspects of a theme from its inception and rendering through its consumption by an end user, with in-depth discussion.

By the end of this book, you will clearly understand themes, layouts, and the Alloy API. Most importantly you will obtain the skills to write a theme and layout templates, apply them to a portal, and also control the portlet UI through different mechanisms.

This clear, concise, and practical tutorial will ensure that you have developed skills to become a competent Liferay themer. The detailed text is accompanied with source code that allows you to play with the examples, update the code, and add custom features.

A practical guide to customizing the look and feel of Liferay-based portal applications

What This Book Covers

Chapter 1, Customizing your Liferay Portal, discussed Liferay architecture and framework, different kinds of plugins, and related development strategies.

Chapter 2, Basic Theme Development, addressed basic theme development, page structure, Plugins SDK, Liferay IDE, and so on. It also covered how to update each individual file in the subfolder, including CSS, images, JavaScript, and templates, and how to package, deploy, and test the themes.

Chapter 3, Layout Templates, addressed layout templates in detail. It introduced the basic concepts of Liferay Portal layout template, and how the theme, layout, and portlets work together to generate a portal page, how to create your own layout template, and page rendering code flow, and so on.

Chapter 4, Styling Pages, gave you specific and detailed answers on how to style your pages. It reviewed some Liferay terminologies, how to set up an organization, UI and usability features in Liferay Portal 6, internationalization (i18n) and localization (L10n) at different levels, UI customization of portal page, portlets, how to Add Application pop-up panel, use the Control Panel, and so on.

Chapter 5, Advanced Theme, provided details about what can be done for advanced themes. It covered how to change the value of the theme.parent property for theme creation, and addressed how to add color schemes, how to use Configurable settings in a theme and pre-defined theme settings, how to embed portlets in a theme, and other topics like theme upgrade, creating a FreeMarker-template theme, browser compatibility, Liferay IDE, and other development tools.

Chapter 6, Portlet User Interface, focused on how to customize portlet user interface. Of course, velocity templates will be useful. This chapter addressed multiple portlets support, portlet deployment, portlet and Layout, portlet content and portlet template, portlet chrome customization, Normal view and maximized view of a portlet, AJAX for portlet UI, Portable Document Format (PDF) reports and Excel reports, Vaadin portlets, common tags in portlets, portlet UI customization using hook.

Chapter 7, Velocity Templates, walked through velocity templates with you. This chapter introduced velocity template language, Velocity template, Velocimacro Velocity portlet, Five basic Velocity templates for a theme, Velocity templates and site performance, theme customization through Velocity templates, Velocity template for Web Content portlet, and freeMarker template.

Chapter 8, Alloy User Interface, focused on what's Alloy UI and how to use it. This chapter addressed the story of Alloy UI, What Alloy UI consists of, what Liferay wants to achieve with Alloy UI, Alloy form tags, node and nodelist, Ajax in Alloy UI, Alloy Plugin Widgets, and other Alloy UI features.

Chapter 9, UI Taglib gave specific details about UI tag libraries—what are they and how to use them in customization. Particularly, this chapter addressed important UI taglib, such as asset tag and categories, search container, custom attributes, tab, toggle, calendar, breadcrumb, navigation, panel, social activity, social bookmarks, discussion, ratings, diff, flags, icon, input, and many other useful UI tags.

Chapter 10, User Interface in Production, showed us how to use jQuery UI, Workflow capabilities, custom attributes capabilities in plugins; how to leverage friendly URL routing and mapping; how to use social UI, such as Open Social, Social Activities, and Social Equity; and how to add CAPTCHA or reCAPTCHA, to hook portal core UI and to deploy themes in production.

5
Advanced Theme

A typical portal page consists of a theme, a layout template, and one or more portlets. In *Chapter 2*, *Basic Theme Development*, we covered the basic concepts in themes and how to create a simple theme. In *Chapter 3*, *Layout Templates*, we covered the concepts in layout templates and how to create your own layouts. In *Chapter 4*, *Styling Pages*, we discussed the page styling and some **User Interface** (**UI**) customization at different levels.

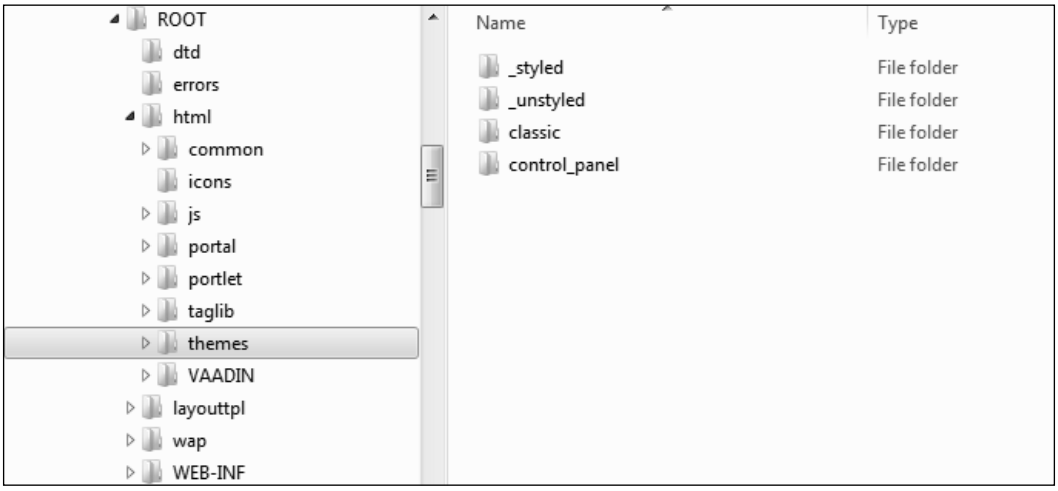
In this chapter, we will explore some aspects of a theme in depth and elaborate on the following theme topics:

- Changing the value of the `theme.parent` property for theme creation
- Adding color schemes
- Configurable settings in a theme
- Pre-defined theme settings
- Embedding portlets in a theme
- Theme upgrade
- Creating a FreeMarker-template theme
- Brower compatibility
- Liferay IDE and other development tools

We will have some lab activities of adding color schemes to a theme and upgrading a theme to Liferay Portal 6.

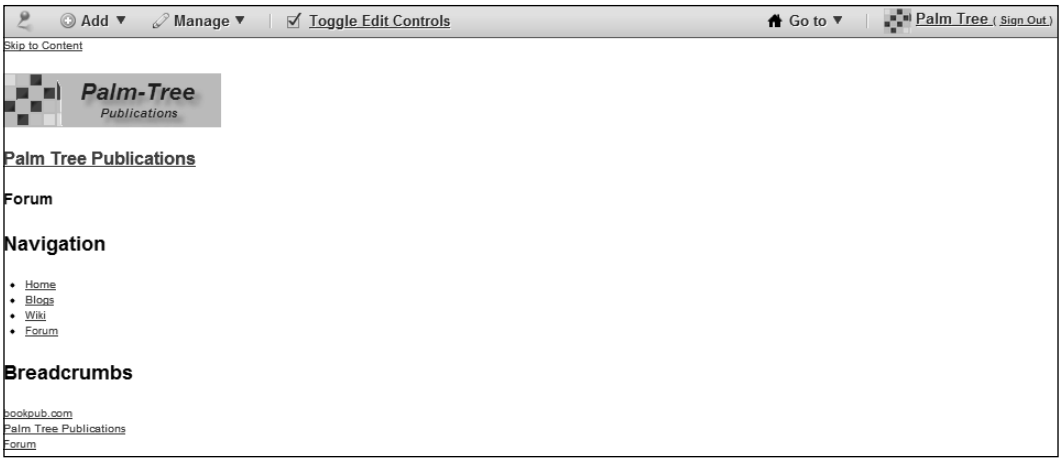
Changing theme.parent property in theme

Liferay provides four different theme implementations out-of-box in the `${PORTAL_ROOT_HOME}/html/themes/` folder as shown in the following screenshot:



When you create a new theme in the `themes/` directory of the Plugins SDK, the Ant script will copy some files to the new theme from one of these three folders, `_styled/`, `_unstyled/`, and `classic/`, which can be specified in the `${PLUGINS_SDK_HOME}/themes/build-common-theme.xml` file.

For example, you can go to the `${PLUGINS_SDK_HOME}/themes/` directory and run `create.bat palmtree "Palm-Tree Publications theme"` on Windows or `./create.sh palmtree "Palm-Tree Publications theme"` on Linux, respectively. Or you can use Liferay IDE to create same `New Liferay Theme Plugin Project`. This will create a theme folder named `palmtree-theme` in the Liferay Plugins SDK environment. When you run `ant` to build and deploy the theme and then apply the newly created theme to your page, you will find out that the look and feel of the page is messed up, as shown in the following screenshot:



If you are familiar with theme development in Liferay Portal 5.2, you may wonder what has happened to the theme created in Liferay Portal 6 Plugins SDK because you would expect a fully working theme with two simple commands, `create` and `ant`. This is because the following `build.xml` file in your theme specifies `_styled` as the value for the `theme.parent` property:

```
<?xml version="1.0"?>
<project name="palmtree-theme" basedir="." default="deploy">
  <import file="../build-common-theme.xml" />
  <property name="theme.parent" value="_styled" />
</project>
```

This means that when your newly created theme is built, it will copy all the files from the `_styled/` folder in the `${PORTAL_ROOT_HOME}/html/themes/` directory to the `docroot/` folder of your theme. The default `_styled/` folder does not have enough files to create a completely working theme and that is why you see a messed-up page when the theme is applied to a page. The reason why this default `_styled/` folder does not include enough files is that some Liferay users prefer to have minimal set of files to start with.

You can modify the `build.xml` file for your theme in the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/` folder by changing value of the `theme.parent` property from `_styled` to `classic`, if you prefer to use the Classic theme as the basis for your theme modification.

```
<property name="theme.parent" value="classic" />
```

Now you will see that your page looks exactly the same as that with the Classic theme after you build and apply your theme to the page (refer to the following screenshot):

Adding color schemes to a theme

When you create a theme in the Plugins SDK environment, the newly created theme by default supports one implementation and does not automatically have color schemes as variations of the theme. This fits well if you would like to have a consistent look and feel, especially in terms of the color display, across all the pages that the theme is applied to.

In your portal application, you might have different sites with slightly different look and feel for different groups of users such as physicians and patients. You might also need to display different pages such as public pages with one set of colors and the private pages with a different set of colors. However, you don't want to create several different themes for reasons such as easy maintenance. In this case, you might consider creating different color schemes in your theme.

Color schemes are specified using a **Cascading Style Sheets** (**CSS**) class name, with which you are able to not only change the colors, but also choose different background images, border colors, and so on.

In the previous section, we created a PalmTree Publication theme, which takes the Classic theme as its parent theme. Now we can follow the mentioned steps to add color schemes to this theme:

1. Copy the `${PORTAL_ROOT_HOME}/webapps/palmtree-theme/WEB-INF/liferay-look-and-feel.xml` file to the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/WEB-INF/` folder.
2. Open the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/WEB-INF/liferay-look-and-feel.xml` file in your text editor.

3. Change `<theme id="palmtree" name="PalmTree Publication Theme" />` as shown in the highlighted lines here, and save the change:

```
<look-and-feel>
  <compatibility>
    <version>6.0.5+</version>
  </compatibility>
  <theme id="palmtree" name="Palm Tree Publications Theme">
    <color-scheme id="01" name="Blue">
      <css-class>blue</css-class>
      <color-scheme-images-path>${images-path}/color_schemes/
blue</color-scheme-images-path>
    </color-scheme>
    <color-scheme id="02" name="Green">
      <css-class>green</css-class>
    </color-scheme>
    <color-scheme id="03" name="Orange">
      <css-class>orange</css-class>
    </color-scheme>
  </theme>
</look-and-feel>
```

4. Go to the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/palmtree-theme/_diffs/` folder and create a `css/` subfolder.
5. Copy both `custom.css` and `custom_common.css` from the `${PORTAL_ROOT_HOME}/html/themes/classic/_diffs/css/` folder to the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/_diffs/css/` folder. This is to let the default styling handle the first color scheme `blue`.
6. Create a `color_schemes/` folder under the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/palmtree-theme/_diffs/css/` folder.
7. Place one `.css` file for each of your additional color schemes. In this case, we are going to create two additional color schemes: `green` and `orange`.
8. To make the explanation simpler, copy both the `green.css` and `orange.css` files from the `${PORTAL_ROOT_HOME}/html/themes/classic/_diffs/css/color_schemes/` folder to the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/_diffs/css/color_schemes/` folder.
9. Copy all images from the `${PORTAL_ROOT_HOME}/html/themes/classic/_diffs/images/` folder to the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/_diffs/images/` folder.

10. Add the following lines in your `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/_diffs/css/custom.css` file:

```
@import url(color_schemes/green.css);
@import url(color_schemes/orange.css);
```

If you open either the `green.css` or the `orange.css` file, you will be able to identify the styling for the CSS by using a color prefix for each CSS definition. For example, in the `orange.css` file you would find that each item is defined like this:

```
orange .dockbar {
  background-color: #AFA798;
  background-image: url(../../images/color_schemes/orange/dockbar/
dockbar_bg.png);
}
.orange .dockbar .menu-button-active {
  background-color: #DBAC5C;
  background-image: url(../../images/color_schemes/orange/dockbar/
button_active_bg.png);
}
```

In the `green.css` file, the style is defined like this:

```
green .dockbar {
  background-color: #A2AF98;
  background-image: url(../../images/color_schemes/green/dockbar/
dockbar_bg.png);
}
.green .dockbar .menu-button-active {
  background-color: #95DB5C;
  background-image: url(../../images/color_schemes/green/dockbar/
button_active_bg.png);
}
```

Also, notice that the related images used for the different color schemes are included in the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/_diffs/images/color_schemes/` folder.

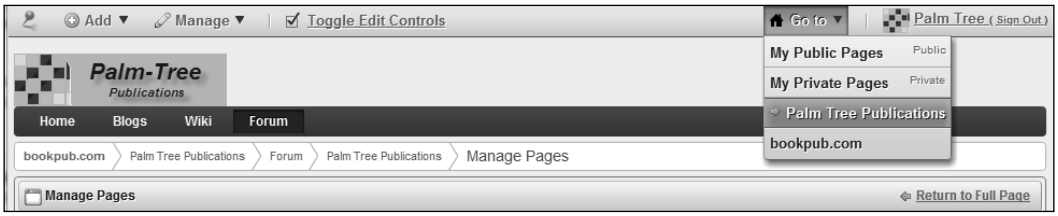
11. Re-build and deploy the PalmTree Publication theme.
12. Log in as administrator and go to the **Control Panel** to apply this theme to the PalmTree Publications Inc. organization. You will be able to see three color schemes available under this theme.
13. Select any of them to apply it to the **Public Pages**.

14. Take a screenshot of the page when each of the color schemes is applied and save it as `thumbnail.png` in the corresponding `blue`, `green`, and `orange` subfolders in the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/_diffs/images/color_scheme/` folder. Three screenshots are used to distinguish between the three color schemes in the **Control Panel** as seen in the following screenshot:

The following screenshots shows how each of the three color schemes looks when applied to a portal page:

As shown in above screenshot, color scheme blue has been applied on the Home page, by default. The following screenshot shows applying color scheme green on the current page. Of course, you would be able to apply color schemes blue or green in the entire site, if required.

Similar to color schemes blue and green, you can apply color scheme orange as well on the current page or the entire site, as shown in following screenshot:



So it works! The page background is with a hue of gray color. Now, what if we want to change the page background color to red for the green color schema?

Update the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/_diffs/css/color_schemes/green.css` file as follows. The commented outline is the original content. The next line after the comment is the new content. The `#FF0000` code is for the red color.

```
body.green, .green .portlet {
  /*background-color: #F1F3EF;*/
  background-color: #FF0000;
}
```

Re-deploy the PalmTree theme and refresh the portal page that uses the `green` color scheme. Now, you should be able to see the portal page with a red background color.

As you can see, you can use theme color schemes to create some variants of the same theme without creating multiple themes. This is useful when you have different but related user roles such as physicians, nurses, and patients and you need to build a different site for each of them. You can use the color schemes to display each of these sites in a slightly different look and feel.

Configurable theme settings

There are many use cases where you would like to change some default settings in the theme so that you can modify the values after the theme is built and deployed. Fortunately, each theme can define a set of settings to make this configurable. The settings are defined as key-value pairs in the `liferay-look-and-feel.xml` file of the theme with the following syntax:

```
<settings>
  <setting key="my-setting1" value="my-value1" />
  <setting key="my-setting2" value="my-value2" />
</settings>
```

These settings can then be accessed in the theme templates using the following code:

```
$theme.getSetting("my-setting1")
$theme.getSetting("my-setting2")
```

For example, I need to create two themes—PalmTree Publications theme and AppleTree Publications theme. They are exactly the same except for some differences in the footer content that includes copyright, terms of use, privacy policy, and so on. Instead of creating two themes packaged as separate `.war` files, we create one set of two themes that share the majority of the code including CSS, JavaScript, images, and most templates; but with one configurable setting and two different implementations of the footer Velocity files.

Here is how this can be done:

1. Open `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/WEB-INF/liferay-look-and-feel.xml` in the above sample PalmTree theme.
2. Copy the PalmTree theme section and paste it in the same file but after this section.
3. Rename the values of `id` and `name` from `palmtree` and `PalmTree Publications theme` to `appletree` and `AppleTree Publications theme` in the second section.
4. Add the following setting to the `palmtree` section before the color-scheme definition:

```
<settings>
  <setting key="theme-id" value="palmtree" />
</settings>
```

5. Add the following setting to the `appletree` section before the color-scheme definition:

```
<settings>
  <setting key="theme-id" value="appletree" />
</settings>
```

6. Find the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/WEB-INF/liferay-look-and-feel.xml` file as follows:

```
<look-and-feel>
  // ignore details
  <theme id="palmtree" name="PalmTree Publications Theme">
    <settings>
      <setting key="theme-id" value="palmtree" />
    </settings>
    <color-scheme id="01" name="Blue">
    // ignore details
    </color-scheme>
```

```
  </theme>
  <theme id="appletree" name="AppleTree Publications Theme">
    <settings>
      <setting key="theme-id" value="appletree" />
    </settings>
    <color-scheme id="01" name="Blue">
    // ignore details
    </color-scheme>
  </theme>
</look-and-feel>
```

7. Copy the `portal_normal.vm` file of the Classic theme from the `${PORTAL_ROOT_HOME}/html/themes/classic/_diffs/templates/` folder to the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/_diffs/templates/` folder.
8. Open the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/_diffs/templates/portal_normal.vm file`
9. Replace the default footer section with the following code:

```
#set ($theme_id = $theme.getSetting("theme-id"))
#if ($theme_id == "palmtree")
  #parse ("$full_templates_path/footer_palmtree.vm")
#else
  #parse ("$full_templates_path/footer_appletree.vm")
#end
```

10. Create your own version of the two footer Velocity templates in the `${PLUGINS_SDK_HOME}/themes/palmtree-theme/docroot/_diffs/templates/` folder.
11. Add related CSS definitions for your footer in your `custom.css` file.
12. Build and deploy the theme `.war` file.

Now you should be able to see both the `PalmTree` and `AppleTree` themes when you go to the Control Panel to apply either theme to your page.

Based on the theme to be used, you should also notice that your footer is different.

Of course, we can take other approaches to implement a different footer in the theme. For example, you can dynamically get the organization or community name and render the footer differently. However, the approach we explained previously can be expanded to control the UI of the other theme components such as the header, navigation, and portlets.

Portal predefined settings in theme

In the previous section, we discussed that theme engineers can add configurable custom settings in the `liferay-look-and-feel.xml` file of a theme. Liferay portal can also include some predefined out-of-the-box settings such as `portlet-setup-show-borders-default` and `bullet-style-options` in a theme to control certain default behavior of the theme.

Let us use `portlet-setup-show-borders-default` as an example to explain how Liferay portal controls the display of the portlet border at different levels.

If this predefined setting is set to `false` in your theme, Liferay portal will turn off the borders for all portlets on all pages where this theme is applied to.

```
<settings>
  <setting key="portlet-setup-show-borders-default" value="false" />
</settings>
```

By default, the value is set to `true`, which means that all portlets will display the portlet border by default.

If the predefined `portlet-setup-show-borders-default` setting is set to `true`, it can be overwritten for individual portlets using the `liferay-portlet.xml` file of a portlet as follows:

```
<liferay-portlet-app>
  // ignore details
  <portlet>
    <portlet-name>sample</portlet-name>
    <icon>/icon.png</icon>
    <use-default-template>false</use-default-template>
    <instanceable>true</instanceable>
    <header-portlet-css>/css/main.css</header-portlet-css>
    <footer-portlet-javascript>/js/main.js</footer-portlet-javascript>
    <css-class-wrapper>sample-portlet</css-class-wrapper>
  </portlet>
// ignore details
</liferay-portlet-app>
```

Set the `use-default-template` value to `true` if the portlet uses the default template to decorate and wrap content. Setting it to `false` will allow the developer to maintain the entire output content of the portlet. The default value is `true`. The most common use of this is when you want the portlet to look different from the other portlets or if you want the portlet not to have borders around the output content.

The `use-default-template` setting of each portlet, after being set to either `true` or `false`, in the `liferay-portlet.xml` file, can be further overwritten by the portlet's popup CSS setting. Users with the appropriate permissions can change it by going to the **Look and Feel** | **Portlet Configuration** | **Show Borders** checkbox of the portlet, as shown in the next screenshot:

Embedding non-instanceable portlets in theme

One common requirement in theme development is to add some portlets in different components of a theme.

For example, you might want to add the Sign In portlet in the header of your theme, the Web Content Search portlet in the navigation area, and the Site Map portlet in the footer area. All the Liferay out-of-the-box portlets can be referred in the `${PORTAL_ROOT_HOME}/WEB-INF/liferay-portlet.xml` file.

How can this be achieved?

Well, it can be quite easy or pretty tricky to embed an out-of-the-box portlet in a theme.

Embedding Dockbar and Breadcrumb portlets in a theme

As explained in *Chapter 4, Styling Pages*, the Dockbar portlet is embedded at the very top in the default Classic theme in the `portal_normal.vm` file as shown next:

```
#if($is_signed_in)
  #dockbar()
#end
```

In the same way, the **Breadcrumb** portlet can be embedded in the content area of the Classic theme in the `portal_normal.vm` file:

```
#breadcrumbs()
```

Embedding Language and Web Content Search portlets in a theme

Some other Liferay out-of-the-box portlets such as the Language and Web Content Search portlets can be embedded in a theme or a layout template easily.

For example, the **Web Content Search** portlet can be added to the far right side of the horizontal navigation area of your theme as follows in the `navigation.vm` file of your theme.

```
<div id="navbar">
  <div id="navigation" class="sort-pages modify-pages">
    <div id="custom">
      $theme.journalContentSearch()
    </div>
    // ignore details
  </div>
</div>
```

In the same way, the Language portlet can be embedded in the `portal_normal.vm` Velocity template file of your theme:

```
$theme.language()
```

Again, you need to add the necessary CSS definitions in your `custom.css` file to control the look and feel and the location of your embedded portlet(s).

Embedding Sign In portlet in the header area of a theme

Sometimes the theme design requires that the Liferay Sign In portlet be in the header area. By default, the Sign In portlet has a portlet border but we need to disable it. The previously mentioned approaches for embedding a portlet in a theme through Velocity attributes in a template do not work in this case because we need to customize the default UI of the embedded portlet.

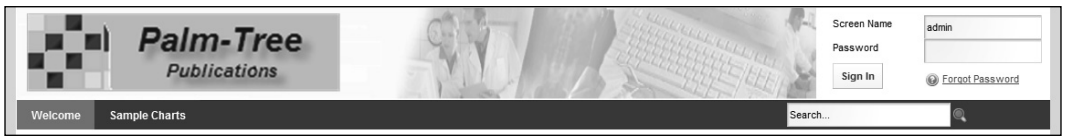
Instead, we can add the following code to the header section in the `portal_normal.vm` file in our sample `PalmTree` theme:

```
#if(!$is_signed_in)
  #set ($locPortletId = "58")
```

```
    $velocityPortletPreferences.setValue("portlet-setup-show-borders",
"false")
    #set($locRenderedPortletContent = $theme.runtime($locPortletId, "",
$velocityPortletPreferences.toString()))
    $locRenderedPortletContent
    $velocityPortletPreferences.reset()
#end
```

After the theme is re-built and re-deployed, we can see that the Sign In portlet is rendered in the header area underneath the logo without the portlet border.

The next step for us is to modify the `custom.css` file and the related files by adding CSS definition to control the location and look and feel of this Sign In portlet. The following screenshot shows the Sign In portlet in the header area and the Web Content Search portlet in the navigation area in a working theme in the production environment:



Embedding instanceable portlets in theme

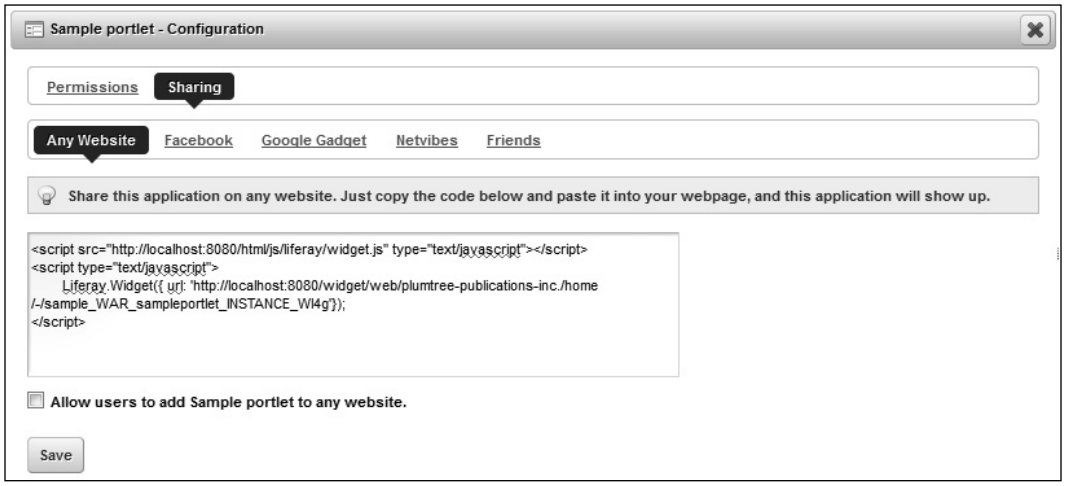
All these embedded portlets are non-instanceable as defined in the `liferay-portlet.xml` file. A **non-instanceable** portlet has only one instance on a portal page and therefore, its portlet ID is static. For **instanceable** portlets such as the Web Content Display portlet, the portlet ID is generated when an instance of the portlet is created, and therefore, can be unknown when the theme is created. This makes it more difficult to embed such portlets in a theme.

Custom portlets can also be embedded in a theme.

The first thing you need to know is the porlet ID. This can be found out by following these steps:

1. Create a custom portlet to be embedded.
2. Build and deploy the portlet.
3. Log in as portal administrator.
4. Add the portlet on a page.
5. Click on **Configuration** | **Sharing** | **Any Website**.

The portlet ID is shown in the JavaScript URL, as displayed in the following screenshot. In this case, the portlet ID is `sample_WAR_sampleportlet_INSTANCE_WI4g`.



Now you can embed this portlet in a theme by adding the following code in your theme Velocity template file:

```
$theme.runtime("sample_WAR_sampleportlet_INSTANCE_WI4g")
```

Please note that the above `$theme.runtime(portlet.id)` call creates a new portlet on the current page. As a result, it actually creates a new record in the Liferay database table. Each portlet on each page has its own portlet preference, and therefore, you would have to reset its preference when a new page with an embedded portlet in a theme is created. For more details about embedding portlets with portlet preferences, please refer to the forum post at `http://www.liferay.com/community/forums/-/message_boards/message/772138`.

Adding runtime portlets to a layout

Similar to adding a runtime portlet to a theme, you can add a runtime portlet to a layout in your custom layout template. For example, you can use the following code to add the Web Content Search portlet, which has a portlet ID of 77, to the third column in the second row of the `1_3_columns.tpl` of your custom layout template:

```
<div class="columns-3" id="content-wrapper">
  <table class="lfr-grid" id="layout-grid">
  <tr>
    <td class="lfr-column" colspan="3" id="column-1"
valign="top">
      $processor.processColumn("column-1")
    </td>
  </tr>
  <tr id="column-center">
    <td class="lfr-column thirty" id="column-2"
valign="top">
      $processor.processColumn("column-2")
    </td>
    <td class="lfr-column thirty" id="column-3"
valign="top">
      $processor.processColumn("column-3")
    </td>
    <td class="lfr-column thirty" id="column-4"
valign="top">
            $processor.processPortlet("77")
      $processor.processColumn("column-4")
    </td>
  </tr>
  </table>
</div>
```

The Web Content Search portlet will always be displayed on any page where the above custom layout template is used. The portlet can be configured, minimized or maximized but can't be removed from the page.

Theme upgrade

A common Liferay Portal project is to upgrade a portal application from an older version of Liferay Portal to Liferay Portal 6. The upgrade process typically includes the following steps:

- Installation of Liferay Portal 6
- Database upgrade
- Theme upgrade
- Content repository upgrade
- Portlet upgrade
- Upgrading the custom code, such as that in the Ext environment, in the previous version to the Ext plugin in Liferay Portal 6

In this section, we will focus on the theme upgrade process. The upgrading of the other components will be discussed in the later chapters.

Here, we will use the Natural Essence theme as an example and walk through the upgrading process of it, from Liferay 5.2 to 6. The source code of this theme is downloadable on the Community Plugins page on the Liferay site.

Let us first build the theme in the Plugins SDK of Liferay 5.2 and generate the `natural-essence-theme-5.2.0.1.war` file and then deploy it to `liferay-portal-tomcat-6.0-5.2.0`. Here is how it looks:

Now we are going to take the following steps to upgrade it to Liferay Portal 6, but we will keep the original look and feel:

1. Download `natural-essence-theme-5.2.0.1.zip` from `http://www.liferay.com/downloads/liferay-portal/community-plugins/`.
2. In the `${PLUGINS_SDK_HOME}/themes/` directory, run the following command to create a theme named *Natural Essence* in the Plugins SDK environment:

   ```
   create.bat natural-essence "Natural Essence theme"
   ```

3. Copy the `css/`, `images/`, `js/`, and `templates/` folders of the Classic theme from the `${PORTAL_ROOT_HOME}/html/themes/classic/` folder to the `${PLUGINS_SDK_HOME}/themes/natural-essence-theme/docroot/_diffs/` folder.
4. Build and deploy it to Liferay Portal 6. You will get a Natural Essence theme with the same look and feel as the Liferay Classic theme. We will start the upgrade process from here.
5. Notice that the Natural Essence theme 5.2.0.1 is unique in body, wrapper, banner, and navigation backgrounds. So we will start the code update in these four parts first. Copy the `natural-essence-theme-5.2.0.1/images/custom/` folder, which contains three image files, to the `${PLUGINS_SDK_HOME}/themes/natural-essence-theme/docroot/_diffs/images/` directory. This folder contains custom images for the Natural Essence theme.
6. Compare the `natural-essence-theme-5.2.0.1/css/custom.css` file with the `${PLUGINS_SDK_HOME}/themes/natural-essence-theme/docroot/_diffs/css/custom.css` file. Copy the related content in the former `custom.css` file to the latter `custom.css` file. Update the latter `custom.css` file as follows. The commented out code is the original code:

   ```
   body {
     /*background: #EEF0F2;
     font-size: 11px;*/
     background: #7C6F5C url(../images/custom/body_bg.jpg);
     color: #222;
     padding: 33px 0;
   }
   #wrapper {
     /*background: none;
     margin: 0 auto;
     max-width: 90%;
     min-width: 960px;
     position: relative;*/
     background: #FFF;
   ```

```
  border: 6px solid #332;
  border-top: 6px solid #332;
  margin: 0 auto;
  width: 960px;
}
#banner {
  /*background: none;
  height: auto;*/
  background: #DAD7C5 url(../images/custom/banner_bg.jpg) no-
repeat scroll left bottom;
  padding: 0 0 220px;
  position: relative;
}
#navigation {
  /*background: #414445 url(../images/navigation/bg.png) repeat-x
0 0;
  clear: both;
  margin: 0 auto 5px;
  min-height: 2.2em;
  padding: 0 5px;*/
  background: #EAE7DF url(../images/custom/navigation_bg.gif);
  height: 41px;
}
```

7. Deploy the updated Natural Essence theme and compare it with the Natural Essence 5.2.0.1 theme in look and feel.
8. Notice that the Natural Essence 5.2.0.1 theme does not have the breadcrumb of **liferay.com | Nature**. Instead of the breadcrumb, there is a navigation bar. So we remove the breadcrumb and put the navigation bar in its place. In the `${PLUGINS_SDK_HOME}/themes/natural-essence-theme/docroot/_diffs/templates/portal_normal.vm` file, remove the following section:

```
#if ($has_navigation)
  #parse ("$full_templates_path/navigation.vm")
#end
```

9. Remove the following section:

```
<nav class="site-breadcrumbs" id="breadcrumbs">
  <h1>
    <span>#language("breadcrumbs")</span>
  </h1>
  #breadcrumbs()
</nav>
```

10. Instead of the previous snippet of code, add the following code:

```
#if ($has_navigation)
  #parse ("$full_templates_path/navigation.vm")
#end
```

11. Build and deploy the theme to see the result.
12. There is a white strip above the **LIFERAY** logo, which does not look good. The logo presentation is controlled by the following code in the `${PLUGINS_SDK_HOME}/themes/natural-essence-theme/docroot/_diffs/templates/portal_normal.vm` file:

```
<header id="banner" role="banner">
  <hgroup id="heading">
    <h1 class="company-title">
      <a class="logo" href="$company_url" title="#language("go-
to") $company_name">
        <span>$company_name</span>
      </a>
    </h1>
    // ignore details
```

13. In the `custom.css` file, find the corresponding style code as follows:

```
#banner .company-title {
  float: none;
  margin: 15px 0 0;
  position: static;
}
```

14. Update the previous snippet of code as follows:

```
#banner .company-title {
  float: none;
  margin: 0 0 0;
  position: static;
}
```

15. Notice that in the present navigation bar, the page names are in white, which are not vivid. The page name font color is controlled by the following code in the `${PLUGINS_SDK_HOME}/themes/natural-essence-theme/docroot/_diffs/templates/navigation.vm` file:

```
<nav class="sort-pages modify-pages" id="navigation">
  <h1>
    <span>#language("navigation")</span>
  </h1>
    <ul>
```

```
#foreach ($nav_item in $nav_items)
  #if ($nav_item.isSelected())
    <li class="selected">
      // ignore details
```

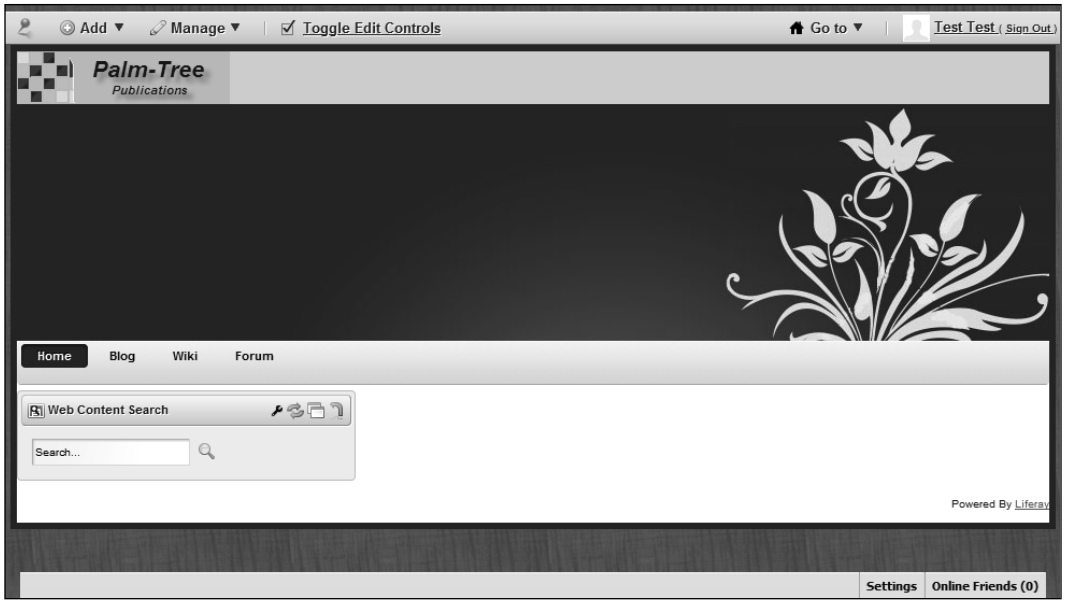
16. The corresponding style code in the `custom.css` file is as follows:

```
#navigation a {
  color: #FFF;
  font-size: 1.1em;
  font-weight: bold;
  margin: 0 1px;
  padding: 3px 15px;
  text-decoration: none;
}
```

17. Update the previous snippet of code as follows:

```
#navigation a {
  color: #000000;
  font-size: 1.1em;
  // ignore details
}
```

18. We have changed the font color into black. Here is how the Natural Essence theme looks now:



This is the Natural Essence theme 6. It looks almost exactly same as the Natural Essence theme 5.2.0.1.

The upgrade of a theme from an old version to Liferay Portal 6 is straightforward. The main difference is the introduction of Dockbar portlet, which includes some of the main administration functionalities in a very compact and short tool bar. This makes most themes in Liferay Portal 6 much simpler and easier to implement because all you need to do is focus on the logo, header, navigation, breadcrumb, portlets, and footer.

Another important part of the theme upgrade is to handle the jQuery library. As you might have known already, Liferay Portal 6 has introduced Alloy UI, which is based on **HyperText Markup Language 5** (**HTML5**), **Cascading Style Sheets Level 3** (**CSS3**), and **Yahoo User Interface** version **3** (**YUI3**). The widely used jQuery in Liferay 5.x is not packaged in Liferay Portal 6. However, you might have jQuery in your custom portlets. In order to support the jQuery functionalities in your portlets, you need to package the jQuery library in the `jquery-1.4.4.min.js` file, which can be downloaded from the jQuery site at `http://code.jquery.com/jquery-1.4.4.min.js` and the hypertext link should be changed to `http://code.jquery.com/jquery-1.4.4.min.js` as well, in your own theme. Your custom jQuery script files such as `my-custom.js` can also be packaged in your upgraded theme. This can be done by adding the following lines in the `portlet_normal.vm` file in the `${PLUGINS_SDK_HOME}/themes/{your-custom-theme}/docroot/_diffs/templates/` folder:

```
</body>
#js("$javascript_folder/jquery-1.4.4.min.js")
#js("$javascript_folder/my_custom.js")
$theme.include($bottom_include)
</html>
```

Of course, you also need to add both the mentioned JavaScript files in the `${PLUGINS_SDK_HOME}/themes/{your-custom-theme}/docroot/_diffs/js/` folder.

Creating a FreeMarker template theme

The default templates are Velocity templates in the Liferay themes. This can be verified by looking at the following highlighted line in the `${PLUGINS_SDK_HOME}/themes/build-common-theme.xml` file:

```
<if>
  <not>
    <isset property="theme.type" />
  </not>
```

```
  <then>
    <property name="theme.type" value="vm" />
  </then>
</if>
```

Liferay engineers can now create a theme with **FreeMarker** templates in Liferay portal. To create a FreeMarker theme, we need to update the theme type to FreeMarker. This can be done by following the procedure given next:

1. In the `${liferay.plugins.sdk.home}/themes/` folder, run the following command:

   ```
   create.bat|./create.sh] freemarker "FreeMarker Theme"
   ```

2. Add a `theme.type` line in the `${PLUGINS_SDK_HOME}/themes/freemarker-theme/build.xml` file as shown next:

   ```
   <property name="theme.type" value="ftl"></property>
   <property name="theme.parent" value="classic"></property>
   ```

3. Copy the `css`, `images`, `js`, and `templates` folders from the `${PORTAL_ROOT_HOME}/html/themes/classic/` directory to the `${PLUGINS_SDK_HOME}/themes/freemarker-theme/docroot/_diffs/` folder.
4. Ant-deploy the FreeMarker theme to a running Liferay environment. The `freemarker-theme-{version.number}.war` file will be deployed.
5. Copy the `${PORTAL_ROOT_HOME}/webapps/freemarker-theme/WEB-INF/liferay-look-and-feel.xml` file to the `${PLUGINS_SDK_HOME}/themes/freemarker-theme/docroot/WEB-INF/` folder.
6. Update the `${PLUGINS_SDK_HOME}/freemarker-theme/docroot/WEB-INF/liferay-look-and-feel.xml` file as follows:

   ```
   <look-and-feel>
     <compatibility>
       <version>6.0.5+</version>
     </compatibility>
     <theme id="freemarker" name="FreeMarker">
       <template-extension>ftl</template-extension>
     </theme>
   </look-and-feel>
   ```

7. Ant-deploy the FreeMarker theme again.

Now you have a Liferay theme that uses FreeMarker templates. Right now, it looks the same as the original Liferay Classic theme that uses Velocity templates.

What are the theme variables available in the FreeMarker templates? Just like the Velocity templates, you have all the theme variables necessary for creation of a theme in the FreeMarker templates. These variables are defined in the `com.liferay.portal.freemarker.FreeMarkerVariables.java` file and the `${PORTAL_ROOT_HOME}/html/themes/_unstyled/init.ftl` file. Here are some examples:

- `htmlUtil`
- `languageUtil`
- `themeDisplay`
- `company`
- `user`
- `layout`
- `scopeGroupId`
- `permissionChecker`
- `colorScheme`
- `user_id`
- `is_default_user`
- `language`
- `show_control_panel`
- `bottom_include`
- `date`
- `the_year`

Theme coding conventions

One of the best practices in Liferay theme engineering is to follow some coding conventions in the theme development. You can save time, avoid pitfalls, and make your code consistent by following these standards and conventions.

Cascading style sheet conventions

The following snippet of CSS code is taken from the `${PORTAL_ROOT_HOME}/html/themes/_styled/css/application.css` file.

```
/* ---------- Menus ---------- */

.lfr-actions.portlet-options .lfr-trigger strong span, .visible.
portlet-options .lfr-trigger strong span {
    background-image: url(../images/portlet/options.png);
}
```

You can see that the Liferay developers observe the following rules in CSS coding:

- Use a comment line to group-related selectors
- Put one blank line between the selectors and the comment
- Separate multiple selectors with a comma and a newline
- Leave one space between the selector name and the starting `"{"`
- Indent the declarations with one tab
- Put a space between the property name and its value in the declaration
- Leave no space between the URL and the opening `"("` for a background image property
- Use a relative URL path instead of an absolute one

Image folder and file conventions

Liferay uses a lot of images as part of themes. These images files are collectively put in, say, the `${PORTAL_ROOT_HOME}/html/themes/classic/images/` directory for the Classic theme. Within this directory, the image files are further grouped in different subfolders. Here are some examples:

- `add_content`: Images files for the **Add Application** functionality
- `application`: Commonly used image files for all applications
- `arrows`: All shapes of arrow images
- `aui`: Image files used by the newly developed Alloy UI library
- `blogs`: Image files used by the blog portlet and web content portlet

Only alphabetical characters (in lowercase) and underscores are allowed in the image folder names and filenames, such as the following:

- `color_schemes`
- `dockbar`
- `image_gallery/slide_show.png`
- `trees/root.png`

JavaScript coding conventions

Liferay has a **minifier filter** that is used to remove empty lines and extra spaces in the page markup file before the application server sends it to a browser. The purpose of this minifier filter is to reduce the bytes to be sent and thus improve network speed.

In some Liferay versions this minifier filter removes the comments in the JavaScript functions, while in other versions it does not. This has impact on the way we code in JavaScript. Here is an example.

```
function showMessage()
{
  // comment
  alert("Hello World!");
}
```

This function will work without the minifier filter. When the minifier filter is working, it removes the extra spaces. The function becomes as follows:

```
function showMessage(){// comment alert("Hello World!");}
```

Now the browser will complain that the `showMessage` function is not defined. The lesson here is that we should always use multiline comment signs in JavaScript coding, as seen in the following snippet of code:

```
function showMessage()
{
  /* comment */
  alert("Hello World!");
}
```

In this case, even if the function is *minified*, it will still work. This is also true for the functions defined in separate JavaScript files.

The following code works fine without the minifier filter:

```
function <portlet:namespace />validateprofile(frm)
{
  var re = /^[a-zA-Z\.-\s]+$/;
  if(!re.test(frm.<portlet:namespace />firstName.value)) {
    alert("First name contains invalid characters!");
    return false;
  }
  // ignore details
}
```

After the minifier filter is applied, the previous code becomes as follows:

```
function <portlet:namespace />validateprofile(frm){ var re = /^[a-zA-
Z\.-\s]+$/; if(!re.test(frm.<portlet:namespace />firstName.value)) { …
} ... }
```

The browser will complain about an invalid character set (in the regular expression). It will say that the `validateprofile` function is not defined. To fix this, we need to use the `RegExp` object:

```
function <portlet:namespace />validateprofile(frm)
{
  var re = new RegExp('^[a-zA-Z- \.]+$');
  if(!re.test(frm.<portlet:namespace />firstName.value)) {
    alert("First name contains invalid characters!");
    return false;
  }
}
```

It is also advisable to use a good tool for JavaScript code debugging.

Brower compatibility

Many CSS styles are displayed differently in different browsers. This is because while major modern browsers support CSS according to the **World Wide Web Consortium** (**W3C**) specifications, they implement some CSS rules in different ways.

In order for a custom theme to look the same in different browsers, we need to pay attention to three aspects in our theme composing—specifying a DOCTYPE, using CSS reset styles, and limited support of CSS3 in the Internet Explorer 6, 7, and 8.

Specifying a DOCTYPE

Modern browsers have two rendering modes—standards mode and quirks mode. In **quirks mode**, a browser renders pages written for older browsers. The quirks mode is for backward compatibility. In **standards mode**, a browser works according to the W3C specifications as closely as possible. The standards mode complies with W3C standards.

In theme coding, we use the `DOCTYPE` tag to tell a browser that a theme follows standards. After this theme is applied to a page, the browser will render this page according to W3C specifications. One example of the `DOCTYPE` tag is as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

This markup is used in the Classic theme in Liferay Portal 5.x. This XHTML 1.0 Transitional DOCTYPE will tell a browser to switch to standards mode for rendering this page.

The `DOCTYPE` should be the first tag in the document.

In the `portal_normal.vm` file for the Classic theme in Liferay Portal 6, you will find the HTML5 DOCTYPE as follows:

```
<!DOCTYPE html>
```

Using CSS reset styles

A page may still look different in two different browsers even when both the browsers are rendering the page in standards mode. For example, the `<ul>` tag gets padding in Firefox but a margin in Internet Explorer. To make the `<ul>` tag look the same in both Firefox and Internet Explorer, we use the following CSS reset style:

```
* {
  padding: 0;
  margin: 0;
}
```

The `*` is a universal selector. This style resets the padding and margins on all the elements to zero. Now it does not matter whether the `<ul>` tag has padding or a margin because both the attributes have the same value—zero. The `<ul>` tag will look the same in both, Firefox and Internet Explorer.

You will find such styles in a `base.css` file of a Liferay theme. The following is a snippet of code from that file:

```
/* ---------- Browser normalization ---------- */
body, div, dl, dt, dd, ul, ol, li, h1, h2, h3, h4, h5, h6, pre, form,
fieldset, input, textarea, p, blockquote, th, td {
  margin: 0;
  padding: 0;
}
```

This solution is called **browser normalization** in Liferay.

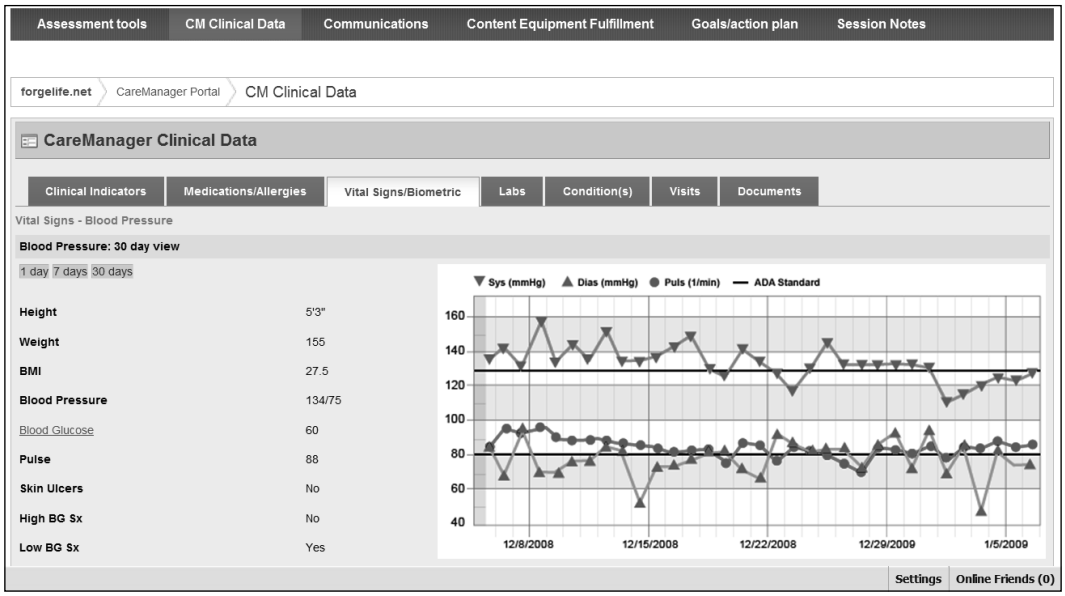
Limited support of CSS3 in Internet Explorer 6, 7, and 8

One of the nice features in CSS3 is to use the border-radius property, as seen in the code snippet that follows, in order to create rounded corners with CSS3 easily, without the need for corner images or the use of multiple `<div>` tags. This is perhaps one of the most talked about aspects of CSS3.

```
-moz-border-radius: 15px;
```

This property is supported well in Firefox, Safari, Google Chrome, and IE 9 browsers. The following screenshot, which was taken in Firefox 3.6, shows how the tabs are displayed with rounded corners without using corner images:

IE 6, 7, and 8 do not sufficiently support CSS3. Particularly, the previous property is not supported. Therefore, the same tabs controlled by the same CSS3 code appear differently (with straight corners) in IE 6, 7 and 8. Refer to the following screenshot taken in IE 7:



Dealing with browser bugs

When the previous measures still do not work or you have to deal with a browser bug, you can add browser prefix to a style to target a specific browser issue.

The following code is from the `custom.css` file of the Liferay Classic theme:

```
.ie6 #wrapper {
  width: 90%;
}
```

This style is for Internet Explorer 6 only. For a `<div>` block whose ID is `wrapper`, its width is specified as relatively 90%.

Development tools

As you can see from the `portal_normal.vm` file, the rendering of a theme is the rendering of a whole web page. Any tool suitable for web application development may be usable for theme development. Especially you may use the following tools in writing a theme.

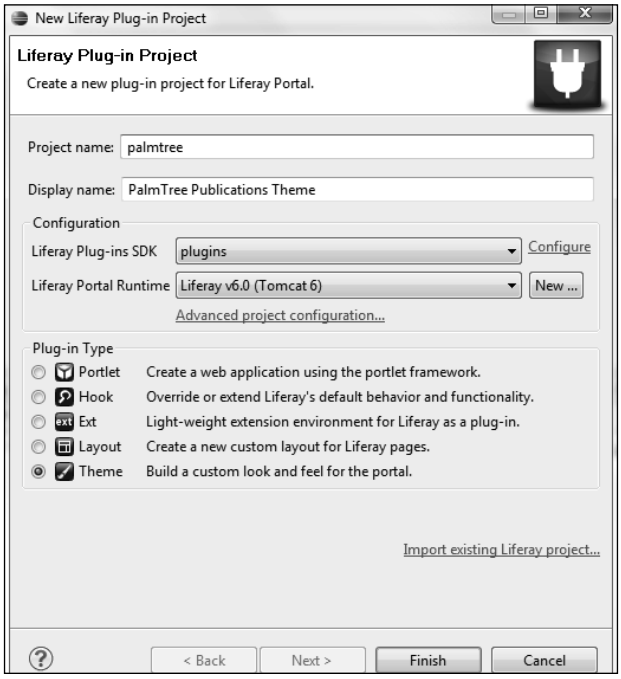
Liferay IDE in Eclipse

Eclipse is a multi-language software development environment comprising an **Integrated Development Environment** (**IDE**) and an extensible plugin system. It is written primarily in Java (and can be used to develop applications in Java) and also, by the means of various plugins, other languages.

Liferay IDE is an extension for the Eclipse platform that supports development of plugin projects for the Liferay portal platform. It is available as a set of Eclipse plugins, installable from an update site. The latest version supports five types of Liferay plugins—portlets, hooks, layout templates, themes, and EXT-style plugin. Liferay IDE requires the Eclipse Java EE developer package of versions Galileo or Helios. For more details, please visit the Liferay IDE wiki page at `http://www.liferay.com/community/wiki/-/wiki/Main/Liferay+IDE`. There are multiple wiki pages available for helping to set up the IDE and create theme plugin in the IDE.

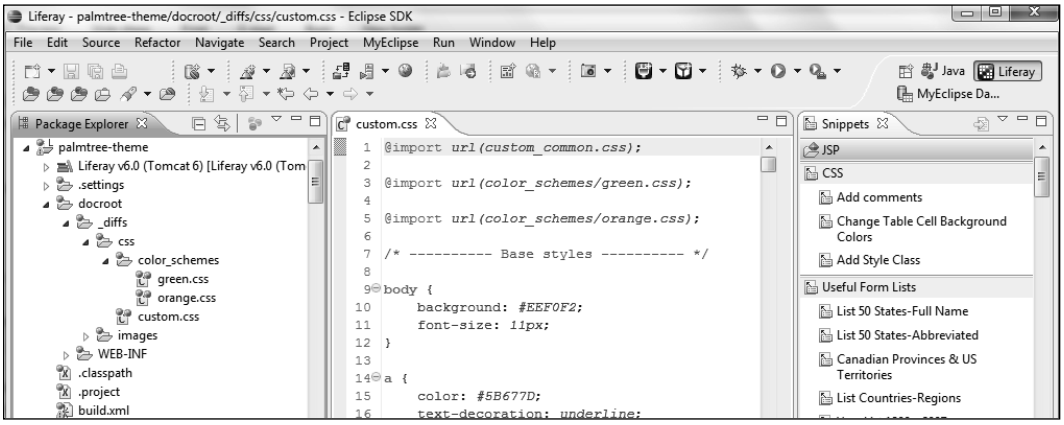
Both Liferay Portal and Liferay Plugins SDK can be integrated in this Liferay IDE running on Eclipse platform. You can include the runtime environment and set up debugging tools among other features, as you are able to do for other J2EE and web projects in Eclipse. You can also download and install Quantum DB and Subversion plugins so that you can connect to Liferay database and check in/out code with Subversion. This IDE can significantly improve your Liferay engineering efficiency.

The following screenshot shows that you can create a Liferay Plugin project in Liferay IDE:



After the theme plugin is created in the Liferay IDE, you can add all the necessary files, including CSS, images, JavaScript, and Velocity templates to your theme folders. Any code changes, whenever saved, will trigger the build process and the modified theme will be deployed in the runtime environment for verification.

The following screenshot shows a theme plugin project in Liferay IDE:



ViewDesigner Dreamweaver plugin

Web designers would be more comfortable using web designer tools such as Dreamweaver or FrontPage to design the themes rather than using text-based editors. The current approach of putting the CSS changes in the `_diffs` folder, and doing an ANT to create the theme **Web application ARchive** (**WAR**) file is not what web designers are used to.

ViewDesigner helps the web designers to easily create or modify Liferay Themes. It is an open source project licensed under **Common Development** and **Distribution License** (**CDDL**). While it currently supports only Windows, a Mac version will be available shortly. For more details, please visit the wiki page at `http://www.liferay.com/web/guest/community/wiki/-/wiki/Main/Theme+Development+using+the+ViewDesigner+Dreamweaver+Plugin`.

W3school site

You can validate your webpage HTML markup, CSS files, and XHTML files at `http://www.w3schools.com/site/site_validate.asp`. You can simply enter the URL of your page or file, and click on **Validate the page** button. Your page or file must be on a running application server or servlet container.

Firebug

Firebug usually integrates with the Firefox browser. It is a powerful web development tool. You can check the HTML markup and styles of your site in real time with Firebug. Firebug is especially useful in debugging JavaScript code. While some browser is murmuring that there is something wrong with your JavaScript, Firebug will directly tell you that the definition of a particular function is missing.

Yslow

Yslow is a Yahoo! web development tool. It is a Firefox add-on and integrates with Firebug. Yslow can analyze your web pages and give suggestions in performance improvement. You can read more about Yslow at `http://developer.yahoo.com/yslow/`.

Google Chrome

Google Chrome has the following tools to help you test your website:

- Web Inspector: It has an hierarchy view of the document object models and a JavaScript console
- Task Manager: It shows all the Google Chrome processes that are running
- JavaScript Debugger

Summary

In this chapter, we have discussed some of the advanced aspects of theme development in Liferay portal 6. In particular, we have learned that:

- A theme developer can specify the value of the `theme.parent` property to create a theme based on an existing theme
- By adding color schemes to a theme, we get variations of a single theme
- A theme designer can hide the portlet borders by changing a theme setting or through configuration in the portlet UI
- There are different ways to embed instanceable and non-instanceable portlets in a theme
- It is easier to upgrade a theme based on a theme that is already running in the target version
- FreeMarker-template-based themes are now available in Liferay
- To write a good theme, a designer should pay attention to coding conventions, browser compatibility, and choice of development tools

In the next chapter, we will investigate the topic of portlet user interfaces.

Where to buy this book

You can buy Liferay User Interface Development from the Packt Publishing website: `https://www.packtpub.com/liferay-user-interface-development/book`

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com

For More Information:
www.packtpub.com/liferay-user-interface-development/book