Liferay Themes: Customizing Liferay's Look & Feel

Liferay is a JSR-168 compliant enterprise portal. Starting with version 3.5.0, Liferay provides a mechanism for developers to easily customize the user interface (look and feel). This is done by creating themes. A theme is basically a user interface design aimed to make the portal more user-friendly and visually pleasing. Multiple themes can be added to the portal for users to choose from. This article explains how you can use this powerful feature to customize the look and feel of Liferay to your own design.

1 What can you do with a theme?

A theme can control the whole look and feel of the pages generated by Liferay. The portal-side of the page can be completely customized. Even the appearance of the portlets that come with Liferay can be customized using CSS, images, JavaScripts, and special templates.

1.1 Understanding the basics

The first important concept when designing a theme is the *portlet template*. Every portlet in Liferay can be divided in two areas:

1. Portlet Content: the internal area that shows the *meat* of the portal.
2. Portlet Template: the external area and can be customized for each theme.

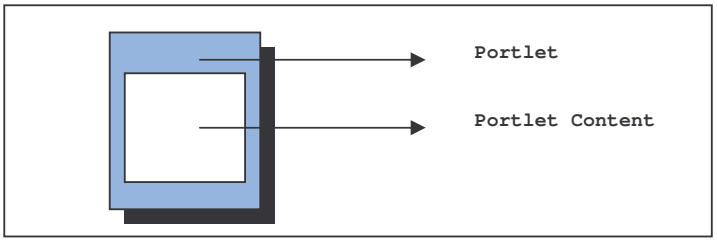


Figure 1.1.1

Using a template for the whole portal is much more flexible than just modifying a page top and bottom (which is the old way). The following pictures show several themes that have been developed for real life websites.

Figure 1.1.2

Figure 1.1.3

Figure 1.1.4

Figure 1.1.5

When portlets are displayed, by default Liferay draws a box around each of them (this behavior is configurable though the file `liferay-portlet.xml`). The appearance of this box can be completely customized by using two templates – namely `portlet_top.jsp` and `portlet_bottom.jsp`. These two templates are responsible for drawing the portlet box shown below.

The *portlet_top* template is responsible for drawing the buttons that control the portlet: maximize, minimize, move up, move down, close, restore, help (if help mode is supported) and edit preferences (if edit mode is supported). The portal tells the template which of these buttons should be shown in each situation.

Besides the portal template and the portal box, the content of the portlet can also be modified. This will only be possible for portlets which follow some conventions. The most important is to use the CSS styles defined by the JSR-168 specification. The portlets packaged along with Liferay use these styles. They also use a special type of box called *inner* every time some

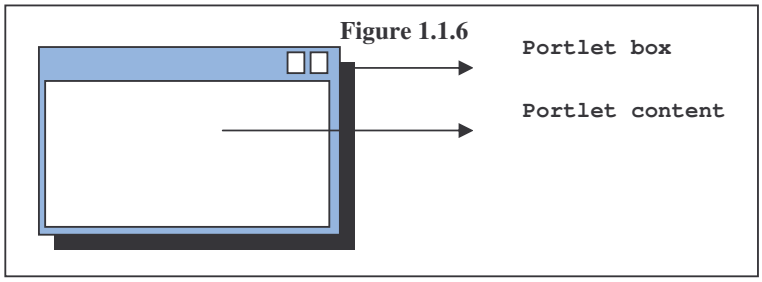


Figure 1.1.6

demarcation is needed. The look and feel of these inner boxes can be modified in a similar way as shown for the portlet box using the templates inner_top and inner_bottom.

Liferay portal installation can have many available themes. By default the desktops of all users and the websites of all groups use the default theme called 'classic', but this theme is configurable. Each user can change its own theme and the portal administrator can set a different theme for each group website. This is much more powerful than other CMS systems which use a single theme for the whole portal!

Now that you have an idea of what can be done with a theme let's take a look at how to create one.

2 Creating new theme

The process of creating a new theme starts at the design using appropriate graphical tools. In this article we will assume this task has already been done and the design is already available as a set of HTML pages. From these files you'll need to edit an XML file, create/adapt CSS files and of course develop some JSP pages by adding some presentation logic to the HTML pages.

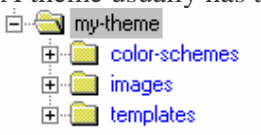
2.1 Adding new themes

Themes can be added to Liferay in two ways:

1. Add all the files and directories of your theme directly under the `/html/themes` directory.
2. Package all the files and directories of your theme into a WAR-file to be dropped into a hot deployable directory (at this time, this only works for Tomcat's `/webapps` directory.)

2.1.1 Adding theme files/directories

A theme usually has the following directory structure:



- **color-schemes**: contains files needed for each of the color schemes of the theme.
- **images**: contains the images of this theme, used/referenced by the templates.
- **templates**: contains the templates for the general view of the portlet, for popups and for the box-like decorations used in different views of the portal.

2.1.2 Defining a new theme

Before digging into these folders, the next step is to register the theme in Liferay. Edit the file `liferay-look-and-feel.xml` present in the *WEB-INF* directory and add the following XML fragment:

```
<theme id="my-theme" name="My theme">
        <root-path>/html/themes/my-theme</root-path>
        <templates-path>/html/themes/my-theme/templates</templates-path>
        <images-path>/html/themes/my-theme/images</images-path>
        <template-extension >jsp</template-extension>
        <settings>
                <setting key="my-setting" value="example-value"/>
        </settings>
        <color-scheme id="gen-color" name="DarkBlue">
                <![CDATA[
      -- Content omitted for clarity --
                ]]>
        </color-scheme>
</theme>
```

The *root-path* element must point to the root theme folder. The elements *templates-path* contains the path to the JSP templates, and *images-path* points to the subfolder that contains the images for this theme. Note that you can choose any subdirectory name you want or point to subdirectories in other directories. However, it is recommended that you keep the default convention unless you have a good reason to change it.

The template-extension element specifies the language in which the templates of the theme are written. JSPs and velocity (specifying **vm** as the template extension) are supported. Liferay comes with examples of both. In this article we will assume JSP templates, but they are very

similar.

Finally the settings element and its inner setting sub elements allow you to specify any number of custom settings of the theme (note that this is optional). These settings can be retrieved from the templates of the theme by calling the `getSetting()` method of the current theme object. For example using JSP scripting:

```
<% String value = themeDisplay.getTheme()getSetting("my-setting"); %>
```

Each theme must also have at least one color-scheme which controls the colors of the theme. The color scheme is defined using the color-scheme element in the `liferay-look-and-feel.xml`. By specifying several color schemes, it is possible to have one single theme with more than one combination of colors. An example of a combination of colors for the previous theme would be:

```
<color-scheme id="gen-color" name="DarkBlue">
    <![CDATA[
        body-bg=#666666

        layout-bg=#666666
        layout-text=#FFFFFF

        layout-tab-bg=#666666
        layout-tab-text=#FFFFFF

        layout-tab-selected-bg=#666666
        layout-tab-selected-text=#FFFFFF

        portlet-title-bg=#666666
        portlet-title-text=#FFFFFF

        portlet-menu-bg=#666666
        portlet-menu-text=#FFFFFF

        portlet-bg=#666666

        portlet-font=#FFFFFF
        portlet-font-dim=#FFFFFF

        portlet-msg-status=#FFFFFF
        portlet-msg-info=#FFFFFF
        portlet-msg-error=#FFFFFF
        portlet-msg-alert=#FFFFFF
        portlet-msg-success=#FFFFFF

        portlet-section-header=#FFFFFF
        portlet-section-header-bg=#666666

        portlet-section-subheader=#FFFFFF
        portlet-section-subheader-bg=#666666

        portlet-section-body=#FFFFFF
        portlet-section-body-bg=#666666

        portlet-section-body-hover=#000000
        portlet-section-body-hover-bg=#F4F4F4

        portlet-section-alternate=#FFFFFF
        portlet-section-alternate-bg=#333333

        portlet-section-alternate-hover=#000000
        portlet-section-alternate-hover-bg=#F4F4F4
```

```
                portlet-section-selected=#FFFFFF
                portlet-section-selected-bg=#666666

                portlet-section-selected-hover=#FFFFFF
                portlet-section-selected-hover-bg=#666666
        ]]>
    </color-scheme>
```

The values of these colors will be used by a file called `css.jsp`.

Liferay acknowledges that the format of a theme may need to change in the future and provides a mechanism to prevent errors should that happen. You can specify for which versions of Liferay your theme is designed for:

```
<look-and-feel>
 <compatibility>
      <version>3.5.0</version>
      <version>3.5.1</version>
 </compatibility>
  ...
```

After covering how to define a new theme we are ready to explain each of its components in more detail.

2.1.3 Theme templates

After the theme has been registered we can start working on the templates that will make up the Liferay look and feel.

Liferay theme's templates are created using JSPs and the struts tiles tag library. Let's start with the portal templates. There are two variants of them:

- portal_normal.jsp: Controls the look & feel of the portal template for normal pages.
- portal_pop_up.jsp: Controls the look & feel of the portal templates for pop-ups. This is used to show a portlet when its window state is equal to `LiferayWindowState.POP_UP`.

Both of these templates must use the tiles ***insert*** tag to insert the content of the current view. The simplest template for any of these variant would be similar to (`taglib` definitions and imports have been omitted for clarity):

```
<tiles:useAttribute id="tilesTitle" name="title"
   classname="java.lang.String" ignore="true" />
<tiles:useAttribute id="tilesSubNav" name="sub_nav"
   classname="java.lang.String" ignore="true" />
<tiles:useAttribute id="tilesContent" name="content"
   classname="java.lang.String" ignore="true" />
<html>
<head>
   <title><%= tilesTitle %></title>
</head>
<body>
<liferay:include page="<%= Constants.TEXT_HTML_DIR + tilesContent %>" flush="true" />
</body>
</html>
```

The template for pop-ups is usually a very simple HTML page. The previous HTML plus including the file `head.jsp` (explained later) is usually enough for most themes.
The template for normal views is probably the most important of the whole theme. It will usually contain most of the design to achieve the effects shown in the previous screenshots or in Liferay's website.

Other mandatory templates are the ones that control the look and feel of the portlet boxes:

- `portlet_top.jsp`: Draws the top part of the box of a portlet.
- `portlet_bottom.jsp`: Draws the bottom part of the box of a portlet.

These two JSP files will have access to the **`portletDisplay`** object which provides information about which portlet control buttons should be shown. It will also contain the URLs that should be used for each of the buttons. (Please see Liferay's API for more information on **`portletDisplay`**.) An example of including the edit icon querying this object is:

```
<c:if test="<%= portletDisplay.isShowEditIcon() %>">
   <a href="<%= portletDisplay.getURLEdit() %>">
         <img border="0" hspace="0" name="p_<%= portletDisplay.getId() %>_edit"
         src="<%= themeDisplay.getPathThemeImage() %>/portlet/edit.gif">
   </a>
</c:if>
```

Take a look to the `portlet_top.jsp` template of the classic theme for a more elaborate example.

The last four mandatory templates are:

- `inner_top.jsp`: Draws the top part of the decoration of a visual section inside a portlet or an administration screen.
- `inner_bottom.jsp`: Draws the top part of the decoration of a visual section inside a portlet or an administration screen.
- `css.jsp`: Generates the dynamic CSS styles based on the color-scheme information provided in `liferay-look-and-feel.xml`. The next section explains the styles that this file may contain in more detail.
- `javascript.jsp`: Declares the location of your custom JavaScript file into the <head> tag of the resulting webpage. (If your theme doesn't make use of `javascript.jsp`, simply create a blank file named `javascript.jsp`.)

The templates folder may optionally include other files too. For example the 'classic' theme uses the following included JSPs:

- `navigation.jsp`: Draws the navigation bar for users to navigate through different layouts.
- `portal_init.jsp`
- `init.jsp`

2.1.4 CSS Styles

CSS allows webpage designers to separate the presentation attributes from the content of a webpage. This is also encouraged with Liferay's theme design. A theme designer can insert CSS style definitions directly into the `css.jsp` file.

Since Liferay is JSR-168 compliant, all the JSR-168 CSS style definitions need to be defined in `css.jsp`. You're encouraged to build on top of the classic theme's `css.jsp` by copying the file to your own theme's template directory. You can easily modify the content of these JSR-168 style definitions to fit your design. (Visit http://www.jcp.org/en/jsr/detail?id=168 for more information on JSR-168.)

The css.jsp file for the classic theme also contains other style definitions. They are divided into different sections for readability. Below is a brief overview of each section:

- Global – contains all the style definitions that affect the webpage as a whole. For

example:
    - Anchor tags shouldn't have backgrounds.
    - Anchor tags, when hovered, should have their text underlined.
    - Forms should have a margin of zero.
    - Form buttons should have a border color based on what's defined in the color-scheme (mentioned earlier). Their borders should be a solid line with a width of one pixel. The font to use for the button should be Arial with a size of 10 pixels.
- Alpha, Beta, Gamma – contain legacy styles that will be phased out in future releases. These styles are currently used directly by some portlets. Please include these styles in your themes for now.
- Liferay Layout (CSS) – contains style definitions that control the appearance of the overall portal layout.
    - Layout-box is the immediate box around the entire portal.
    - Layout-outer-side-decoration and layout-inner-side-decoration are two boxes enclosing the layout-box. These two style definitions allow you to decorate outside of the portal content.
    - Layout-corner-ul allows you to style the upper left corner of the layout. For instance, you can put an image of a rounded corner for your theme.
    - Layout-corner-br allows you to style the bottom right corner of the layout.
- Column Layout – contains style definitions that control the appearance of the portlet columns. For example, how wide should the wide column be, how much space between columns, etc.
    - Layout-column-n1 styles the first narrow column.
    - Layout-column-n2 styles the second narrow column (if there is one).
    - Layout-column-w1 styles the wide column.
    - Layout-blank_n1_portlet styles the narrow column when no portlets exist in that column. For the most part, this is used as a place holder in case there's an empty column.
    - Layout-margin-div allows you to style the margin between columns.
- Portlet CSS – contains style definitions that control the appearance of each portlet.
    - Portlet-box is the immediate box around the portlet.
    - Portlet-title controls how the title is displayed.

As you can see, the names of these definitions are self explanatory for the most part. If you're unsure about any of these definitions, you can usually just add a solid border around it, refresh your page, and see what that definition controls. For example:

```
#layout-outer-side-decoration {
      background-color: none;
      margin-left: auto;
      margin-right: auto;
      width: <%= layoutBoxWidth %>px;
      border: 1px solid blue;
}
```

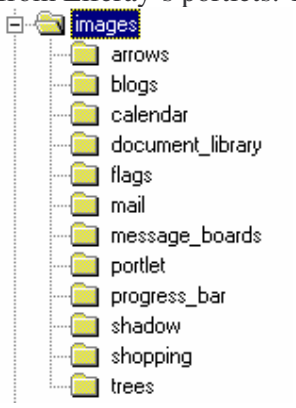
2.1.5 JavaScript

JavaScript is often used to provide your users with a more dynamic user interface. There are instances where some of these scripts need to be declared in the <head> tag instead of the <body> tag. The JSP template `javascript.jsp` allows you to declare the location of your custom JavaScript in the header (<head> tag). Here's an example of how to make a declaration in the `javascript.jsp` file:

```
<script language="JavaScript" src="<%= themeDisplay.getPathThemeRoot()
%>/js/mytheme.js"></script>
```

2.1.6 Images

Each theme has several sets of images which may be used from the templates and are also used from Liferay's portlets. The directory structure of the images folder:



As you can probably tell, some of these subfolders contain images that are needed for some standard Liferay portlets (i.e. Calendar portlet, Mail portlet). Other subfolders contain general images. For example, the tree directory contains images that can be used to display a tree diagram of a file system.

For your custom theme, you may replace these image files with your own (keeping the same filename). Be aware that if the size of a replacement image is different from the original image, the image may not display properly.

For other custom images, you can create a new subdirectory under /images to store all your custom images. You can then reference these custom images in your templates. Here's an example:

Let's say you have created a subdirectory under `/images` named `/mytheme`. You then create an image file called `myimage.gif`. In your template (even `css.jsp`), you can reference this image by inserting this line below:

```
<%= themeDisplay.getPathThemeImage() %>/mytheme/myimage.gif
```

2.2 Advanced features

2.2.1 Packaging themes as a WAR file

Liferay allows packaging a theme or a set of themes inside a WAR file. This file can then be hot-deployed to a running installation.

Let's say your theme is in the `D:\my-custom-themes` directory like shown below:

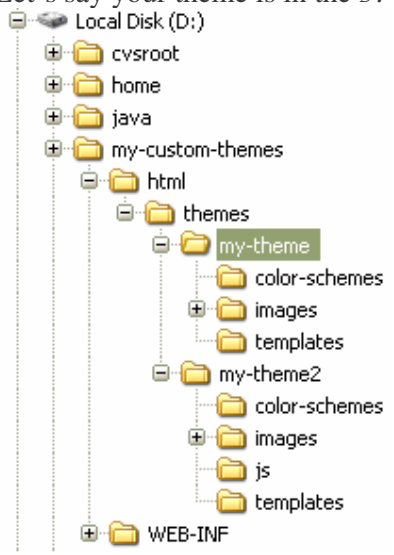


Figure 2.2.1.1

To create a WAR file, simply JAR the entire theme directory. Here's an example:

```
D:\my-custom-themes> jar cvf mythemes.war *.*
```

The WAR file should include the *WEB-INF* folder, which contain the `liferay-look-and-feel.xml` file explained earlier in this user guide. Make sure the preserved directory structure within your WAR file is reflected in the theme declaration in the `liferay-look-and-feel.xml` file (section 2.1.2).

This new `mythemes.war` file can then be hot-deployed to Tomcat by simply copying the WAR file to Tomcat's */webapps* directory. (Currently, only Tomcat supports hot deployment of Liferay themes.) Both **my-theme** and **my-theme2** are now packaged and available in the WAR file.

For another example of hot deployment, please visit
http://www.liferay.com/cms/servlet/DOCUMENTATION-DEVELOPMENT-HOT-DEPLOY

2.2.2 Portal services for theme templates

The portal offers several functionalities that can be (optionally) reused from a theme. These functionalities are implemented in several files of the directory */html/common/themes*. To use its functionality you can include the file which implements it from `portal_normal.jsp` or `portal_pop_up.jsp`.

These functionalities are:

- Managing the session time out: include the file `session_timeout_js.jsp`.
- Play a sound when a new message arrives: include the file `new_mail_alert.jsp`.

- Include general portal header information: include the file `head.jsp`. This file provides:
  - Global portal meta tags
  - Global JavaScript libraries used by several Liferay portlets
- Show an administration message when the server is going to be shutdown: include the file `top_warning.jsp`.

A regular `<jsp:include/>` tag can be used to activate any of these services. For example:

```
<jsp:include page="/html/common/themes/session_timeout_js.jsp"/>
```

2.2.3 Auto-generate images for the classic theme

Liferay also includes a tool that would read the values of the color schemes and generate complementary images for each set of color schemes. Currently, this tool only works for the classic theme that came with Liferay. You're free to develop a similar tool for your themes. *The development of this tool is beyond the scope of this tutorial. Liferay may provide a tutorial to explain this tool in the future.*

To demonstrate the auto image generation tool for the classic theme, follow the steps below:

1. Go to *{$Liferay}/portal/portal-web/docroot/html/themes/classic/color_schemes* (*{$Liferay}* is the Liferay root directory on your system.)
2. Delete all the directories within it (01, 02, 03).
3. Go to *{$Liferay}/portal/portal-web* with in your command prompt.
4. Type the command: **`ant build-color-scheme`**.
5. Validate that the */color_schemes* directory for the classic theme now has images for all the color schemes defined in the `liferay-look-and-feel.xml`.

Feel free to add your own color schemes for the classic theme in `liferay-look-and-feel.xml` and try auto-generating images again!

3 Conclusions

Liferay provides a very powerful theme system with which almost any design can be integrated into the Liferay portal. It is a mature service that has already been tested in several production portals.

Liferay is a multi-theme portal. Several themes can be registered letting each user choose the one they prefer for their desktop and letting the administrator choose which theme to use for the website of each existing group.

In the future, Liferay hope to provide more features to themes. They include (but not limited to):

- Ability to define different sets of themes per type of user or group.
- Tools to help create new themes.
- A more powerful authorization system which could allow the administrators of each group to choose its theme.