Page Layout

Overview

A template will typically have two layout files: index.php for the majority of content and error.php for custom error pages ("404 - Not Found", etc.). Both of these files are contained within the top level of a template (i.e., they cannot be placed in a sub-directory of the template).

```
/app
  /templates
    /{TemplateName}
      error.php
      index.php
```

All the HTML that defines the layout of your template is contained in a file named index.php. The index.php file becomes the core of every page that is delivered and, because of this, the file is **required**. Essentially, you make a page (like any HTML page) but place PHP code where the content of your site should go.

The error.php layout, unlike index.php is optional. When not included in a template, Joomla! will use its default system error layout to display site errors such as "404 - Page Not Found". Including error.php is recommended though as it helps give your site a more cohesive feel and experience to the user.

A Breakdown of index.php

Note: For the sake of simplicity, we've excluded some more common portions found in HUBzero templates. The portions removed were purely optional and not necessary for a template to function correctly. We suggest inspecting other templates that may be installed on your HUB for further details.

Starting at the top:

```
<?php
defined( '_HZEXEC_' ) or die( 'Restricted access' );

$this->addScript($this->baseurl . '/templates/' . $this->template . '/
js/hub.js');

// Get the user's browser and browser version
// We add this to the document root as classes for better targeting wi
```

```
th CSS
$browser = new HubzeroBrowserDetector();
$b = $browser->name();
$v = $browser->major();

// Set the page title
$this->setTitle(Config::get('sitename') . ' - ' . $this->getTitle());
?>
<!DOCTYPE html>
<!--[if lt IE 7 ]> <html dir="<?php echo  $this->direction; ?>" lang="
<?php echo  $this->language; ?>" class="ie6"> <![endif]-->
<!--[if IE 7 ]>    <html dir="<?php echo  $this->direction; ?>" lang="
<?php echo  $this->language; ?>" class="ie7"> <![endif]-->
<!--[if IE 8 ]>    <html dir="<?php echo  $this->direction; ?>" lang="
<?php echo  $this->language; ?>" class="ie8"> <![endif]-->
<!--[if IE 9 ]>    <html dir="<?php echo  $this->direction; ?>" lang="
<?php echo  $this->language; ?>" class="ie9"> <![endif]-->
<!--[if (gt IE 9)|!(IE)]><!--> <html dir="<?php echo $this->direction;
 ?>" lang="<?php echo  $this->language; ?>" class="<?php echo $b . ' '
 . $b . $v; ?>"> <!--<![endif]-->
```

The first line prevents unauthorized people from looking at your coding and potentially causing trouble. Then we grab a reference to the global site configuration. Next, we push some scripts to the document, first checking if the jquery plugin is enabled. Following that, we get the current site visitors browser and browser version. We add this to the document root as classes for better targeting with CSS. The last line of PHP takes the current page title and prepends the site's name. Thus, every page results with a title like "myHUB.org - My Page Title".

The first line of actual HTML tells the browser (and webbots) what sort of page it is. The next line says what language the site is in.

```
<head>
 <link rel="stylesheet" type="text/css" media="screen" href="<?php
 echo HubzeroDocumentAssets::getSystemStylesheet(array(
  'fontcons', 'reset', 'columns', 'notifications', 'pagination',
  'tabs', 'tags', 'comments', 'voting', 'layout'
 )); /* reset MUST come before all others except fontcons */ ?>" />
 <!-- Include the template's main CSS file -->
 <link rel="stylesheet" type="text/css" media="screen" href="<?php ech
o $this->baseurl ?>/templates/<?php echo $this->template; ?>/css/main.
css" />
 <link rel="stylesheet" type="text/css" media="print" href="<?php echo
 $this->baseurl ?>/templates/<?php echo $this->template; ?>/css/print.
```

```
css" />

 <!-- This includes metadata tags and the <title> tag -->
 <jdoc:include type="head" />

 <!--[if IE 9]>
  <link rel="stylesheet" type="text/css" media="screen" href="<?php ec
ho $this->baseurl ?>/templates/<?php echo $this->template; ?>/css/brow
ser/ie9.css" />
 <![endif]-->
 <!--[if IE 8]>
  <link rel="stylesheet" type="text/css" media="screen" href="<?php ec
ho $this->baseurl ?>/templates/<?php echo $this->template; ?>/css/brow
ser/ie8.css" />
 <![endif]-->
 <!--[if IE 7]>
  <link rel="stylesheet" type="text/css" media="screen" href="<?php ec
ho $this->baseurl ?>/templates/<?php echo $this->template; ?>/css/brow
ser/ie7.css" />
 <![endif]-->
</head>
```

The first line compiles several bootstrap CSS files into a single, minified (comments and white-space removed to lessen file size) file to reduce http requests.

The following two lines include the main stylesheet for the template and a print stylesheet that applies more suitable styles when printing.

The fifth line gets Joomla! to put the correct header information in. This includes the page title, meta information, your main.css, system JavaScript, as well as any CSS or JavaScript that was pushed to the template from an extension (component, module, or plugin). This is a bit different than Joomla! 1.5's typical behavior in that the HUBzero code is automatically finding and including main.css and some key JavaScript files from your template. This is done due to the fact that order of inclusion is important for both CSS and JavaScript. For instance, one cannot execute JavaScript code built using the MooTools framework *before* the framework has been included. It would simply fail. As such, the naming and existence of specific directories, CSS, and JavaScript files becomes quite important for a HUBzero template.

The rest creates links to a couple CSS fix style sheets for Internet Explorer (more on this in the Cascading Style Sheets chapter).

Now for the main body:

```
<body>

 <div id="header">
  <h1><a href="<?php echo $this->baseurl ?>" title="<?php echo Config:
:get('sitename'); ?>"><?php echo Config::get('sitename'); ?></a></h1>

  <ul id="toolbar" class="<?php if (!$juser->get('guest')) { echo 'log
gedin'; } else { echo 'loggedout'; } ?>">
<?php
 // Is the user logged in?
 if (!User::isGuest()) {
  // Yes. Show them a different toolbar.
  echo '<li id="logout"><a href="/logout"><span>'.Lang::txt('Logout').
'</span></a></li>';
  echo '<li id="myaccount"><a href="/members/'.User::get('id').'"><spa
n>'.Lang::txt('My Account').'</span></a></li>';
  echo '<li id="usersname">'.User::get('name').' ('.User::get('usernam
e').')</li>';
 } else {
  // No. Show them the login and register options.
  echo "ttt".'<li id="login"><a href="/login" title="'.Lang::txt('Logi
n').'">'.Lang::txt('Login').'</a></li>'."n";
  echo "ttt".'<li id="register"><a href="/register" title="'.Lang::txt
('Sign up for a free account').'">'.Lang::txt('Register').'</a></li>'.
"n";
 }
?>
  </ul>

  <!-- Include any modules for the "search" position -->
  <jdoc:include type="modules" name="search" />
 </div><!-- / #header -->

 <!-- Include any modules assigned to the "user3" position -->
 <div id="nav">
  <h2>Navigation</h2>
  <jdoc:include type="modules" name="user3" />
 </div><!-- / #nav -->

 <div id="wrap">
  <div id="content" class="<?php echo $option; ?>">
   <!-- Include the component output -->
   <jdoc:include type="component" />
  </div><!-- / #content -->

  <div id="footer">
```

```
   <!-- Include any modules assigned to the "footer" position -->
   <jdoc:include type="modules" name="footer" />
  </div><!-- / #footer -->
 </div><!-- / #wrap -->
</body>
```

First we layout the site's masthead in the block. Inside, we set the <h1> tag to the site's name, taken from the global site configuration.

Next, we move on to a toolbar that is present in the masthead of every page. This toolbar contains "login" and "register" links when not logged in and "logout" and "My Account" links when logged in. While not required, it is highly recommended that all templates include some form of this arrangement in an easy-to-find, consistent location.

Some modules that have been assigned the position "search" are then loaded in the masthead. Most HUBzero templates default to having a simple search form module appear. Again, this is not required and placement of modules is entirely up to the developer(s) but we, once again, strongly recommend that some form of a search box be included on all pages.

Then we move on to a block where navigation is loaded. It is here that our main menu will appear.

Next, we get to the primary content block. One of the first things you may notice is the use of module as a jdoc:include type. This is how we tell where in our template to output modules that have been assigned to specific positions.

It is also worth noting the small bit of PHP (<?php echo $option; ?>) in the class attribute of the content . This small bit of code outputs the name of the current component as a CSS class. So, if one were on a page of a "groups" component, the resulting HTML would be . Since all component output is contained inside the "content" div, this allows for more specific CSS targeting.

See the Modules: Loading article for more details on module positioning.

The content div contains a very important jdoc:include of type component. This is where all component output will be injected in the template. It is essential this line be included in a template for it to be able to display any content.

A Breakdown of error.php

Starting at the top:

```
<?php
defined( '_HZEXEC_' ) or die( 'Restricted access' );

// Get the user's browser and browser version
// We add this to the document root as classes for better targeting wi
th CSS
$browser = new HubzeroBrowserDetector();
$b = $browser->name();
$v = $browser->major();
?>
<!DOCTYPE html>
<!--[if lt IE 7 ]> &lthtml dir="&lt?php echo  $this->direction; ?>" la
ng="<?php echo  $this->language; ?>" class="ie6"> <![endif]-->
<!--[if IE 7 ]>    <html dir="<?php echo  $this->direction; ?>" lang="
<?php echo  $this->language; ?>" class="ie7"> <![endif]-->
<!--[if IE 8 ]>    <html dir="<?php echo  $this->direction; ?>" lang="
<?php echo  $this->language; ?>" class="ie8"> <![endif]-->
<!--[if IE 9 ]>    <html dir="<?php echo  $this->direction; ?>" lang="
<?php echo  $this->language; ?>" class="ie9"> <![endif]-->
<!--[if (gt IE 9)|!(IE)]><!--> <html dir="<?php echo $this->direction;
 ?>" lang="<?php echo  $this->language; ?>" class="<?php echo $b . ' '
 . $b . $v; ?>"> <!--<![endif]-->
```

The first line prevents unauthorized people from looking at your coding and potentially causing trouble. Then we grab a reference to the global site configuration. The first line of actual HTML tells the browser (and webbots) what sort of page it is. The next line says what language the site is in.

```
<head>
 <meta http-equiv="content-type" content="text/html; charset=utf-8" />
 <title><?php echo Config::get('sitename'); ?> - <?php echo $this->tit
le; ?> - <?php echo $this->error->message ?></title>
 <link rel="stylesheet" type="text/css" media="all" href="<?php echo $
this->baseurl ?>/templates/<?php echo $this->template; ?>/css/error.cs
s" />
</head>
```

Unlike with index.php, we do not include the <jdoc:include type="head" /> tag. Instead, we

simply set a single metadata tag to declare the character set and then set the title tag. Next, we include the error.css style sheet, which contains styling just for this layout.

Now for the main body:

```
<body>
 <div id="wrap">
  <div id="header">
   <h1><a href="<?php echo $this->baseurl ?>" title="<?php echo $confi
g->getValue('config.sitename'); ?>"><?php echo Config::get('sitename')
; ?></a></h1>
  </div>
  <div id="outline">
   <div id="errorbox" class="code-<?php echo $this->error->code ?>">
    <h2><?php echo $this->error->code ?> - <?php echo $this->error->me
ssage ?></h2>

    <p><?php echo Lang::txt('You may not be able to visit this page be
cause of:'); ?></p>

    <ol>
     <li><?php echo Lang::txt('An out-of-
date bookmark/favourite'); ?></li>
     <li><?php echo Lang::txt('A search engine that has an out-of-
date listing for this site'); ?></li>
     <li><?php echo Lang::txt('A mis-typed address'); ?></li>
     <li><?php echo Lang::txt('You have no access to this page'); ?></
li>
     <li><?php echo Lang::txt('The requested resource was not found');
 ?></li>
     <li><?php echo Lang::txt('An error has occurred while processing
your request.'); ?></li>
    </ol>

    <p><?php echo Lang::txt('If difficulties persist, please contact t
he system administrator of this site.'); ?></p>
   </div><!-- / #errorbox -->

   <form method="get" action="/search">
    <fieldset>
     <?php echo Lang::txt('Please try the'); ?> <a href="index.php" ti
tle="<?php echo Lang::txt('Go to the home page'); ?>"><?php echo Lang:
:txt('Home Page'); ?></a> <span><?php echo Lang::txt('or'); ?></span>
     <label>
      <?php echo Lang::txt('Search:'); ?>
      <input type="text" name="searchword" value="" />
```

```
     </label>
     <input type="submit" value="<?php echo Lang::txt('Go'); ?>" />
    </fieldset>
   </form>
  </div><!-- / #outline -->
<?php
  if ($this->debug) :
   echo "tt".'<div id="techinfo">'."n";
   echo $this->renderBacktrace()."n";
   echo "tt".'</div>'."n";
  endif;
?>
 </div><!-- / #wrap -->
</body>
```

As can be seen, this is relatively straight-forward. We set a title for the page, output the error message, provide some potential reasons for the error and, finally, include a search form. Note that we did not use any modules.

One portion to pay special attention to is the small bit of PHP at the end of the page. This outputs a stack trace when site debugging is turned on.

Note: It is never recommended to turn on debugging on a production site.

Loading Modules

Modules may be loaded in a template by including a Joomla! specific jdoc:include tag. This tag includes two attributes: type, which must be specified as module in this case and name, which specifies the position that you wish to load. Any modules assigned to the specified position (set via the administrative Module Manager) declared in the name attribute will have their output placed in the template (the jdoc:include is removed by the CMS afterwards).

```
<jdoc:include type="modules" name="footer" />
```

See the Modules: Loading article for further details on how to use more advanced features.