How to Create A Landing Page In WordPress

In this tutorial we will use the child theme as the mechanism for achieving our aim of creating a landing page but you could just as easily make changes to your theme files too.

Using a child theme is much cleaner because you are not touching any of your theme's original files and it also means that you won't have to worry about losing your changes if you ever have to update your theme to a newer version.

We will base our child theme example on the default **twentyeleven** theme which comes with every WordPress installation.

(For more info about child themes please see the relevant section in one of our previous articles here or also see the WordPress codex.)

Step 1:

To start with, create a folder somewhere on your computer called "**twentyeleven-landing**" which will contain all of our child theme's files.

Inside this folder create a new file using your favourite code editor called **style.css**. In this file paste the following code:

```
/*
Theme Name:     twentyeleven-landing
Theme URI:      http://yoursitename.com/
Description:    Child theme for the Twenty Eleven theme
Author:         Your Name
Author URI:     http://yoursitename.com/
Template:       twentyeleven
Version:        1.0
*/

@import url("../twentyeleven/style.css");
```

This is the minimum amount of code required to create a child theme.

We'll come back to this file soon to add some CSS code.

Step 2:

Now make a copy of the following original twentyeleven theme files which can be found in that theme's directory (**wp-content/themes/twentyeleven**):

single.php

content.php

header.php

footer.php

Place a copy of the above files in your newly created child theme directory and then rename them as follows:

single.php -> **single-landing.php**

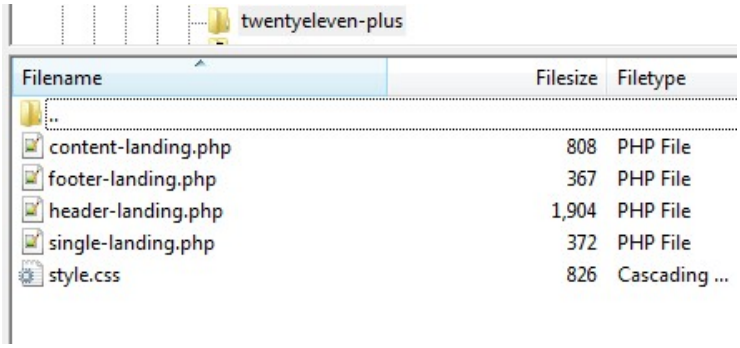
content.php -> **content-landing.php**

header.php -> **header-landing.php**

footer.php -> **footer-landing.php**

Note the naming convention we've used, ie, original filename followed by a dash and then another string – we will talk more about this later.

So you should now have a folder which contains the following files:



Step 3:

Open the **single-landing.php** file and modify the code inside which contains the original code from the single.php we copied earlier.

Make the changes to the content so it looks like the following:

```
<?php

/** * Template Name: Landing Page Template */get_header('landing'); ?>

<div id="primary"> <div id="content" role="main">         <?php while (
have_posts() ) : the_post(); ?>

                    <?php get_template_part( 'content', 'landing' ); ?>

                          <?php endwhile; // end of the loop. ?> </div><!--
#content --></div><!-- #primary -->

<?php get_footer('landing'); ?>
```

Some key areas of change have been highlighted in the figure above and are explained below:

1) The comments section has been modified to now include the template name.

Any words which you specify after the "Template Name:" string will appear in the template drop down box when you are editing the page from the WordPress page editor.

2) The first and last lines of the original code have been modified as follows:

`get_header()` was changed to **`get_header('landing')`**
`get_footer()` was changed to **`get_footer('landing')`**

You'll note that we have now inserted a parameter called `'landing'`.

When calling the get_header and get_footer functions, WordPress allows us to specify our own "header" and "footer" template files as long as the naming convention of our template files is as follows:

header-**{name}**.php and footer-**{name}**.php.

(in our case we used the word "landing" as the value for "name")

Therefore the above lines of code will call our special template files respectively as shown below:

header-landing.php

footer-landing.php

3) Amongst the other changes we have also deleted parts of the original code which were displaying the navigation links for previous/next post.

4) We also modified the line of code which contains the **get_template_part** function as follows:

```
get_template_part('content', 'single')
```

was changed to

```
get_template_part('content', 'landing');
```

You'll note that we changed the second parameter from "single" to "landing".

get_template_part is a wordpress function which allows us to specify two parameters as follows:

get_template_part($slug, $name)

The **$slug** parameter is **mandatory** and refers to a generic template.

The **$name** parameter is optional and tells WordPress to look for the specialized template file with that string in the filename which follows the $slug name.

One way to think of the **$slug** parameter is as a sort of generic name or category describing the part of a page's output that file is dealing with.

Similarly, you could also think of the $name parameter as a sub-category of the above.

As an example, the twentyeleven theme has templates which deal with a page's content.

For instance if you look at the standard twentyeleven theme, it has 11 versions of content templates as follows:

content.php, content-aside.php, content-featured.php, content-gallery.php, content-image.php, content-intro.php, content-link.php, content-page.php, content-quote.php, content-single.php, content-status.php

So in this case, the generic template or $slug value is "content".

The variations of the generic template where the $name parameter is represented are the other versions you see in the above list where $name takes values such as "aside", "featured" etc.

We are doing something similar in our twentyeleven child theme and we are creating our own "content" template file called **content-landing.php**.

If WordPress cannot find the file called content-landing.php it will fallback to using the generic content template file from the parent theme, ie, content.php.

Step 4:

Open the **content-landing.php** file and modify the code inside which contains the original code from the content.php we copied earlier.

Make the changes to the content so it looks like the following:

```
<?php/** * The template for displaying content in the single-landing.php
template * * @package WordPress * @subpackage Twenty_Eleven * @since Twenty
Eleven 1.0 */?><article id="post-<?php the_ID(); ?>" <?php post_class('page-
landing'); ?>>

	<header class="entry-header">		<h1 class="landing-
heading"><?php the_title(); ?></h1>	</header><!-- .entry-header -->

	<div class="entry-content">		<?php the_content(); ?>
	</div><!-- .entry-content -->

</article><!-- #post-<?php the_ID(); ?> -->
```

A summary of the changes made to our content-landing.php file is show below:

1) As you can see from the above we have deleted a lot of the existing code from the original content.php file.
We won't go into every line which we deleted and if you are curious you can compare **our content-landing.php** file with the original twentyeleven **content.php** file in your own time.

2) We also modified the first highlighted line of code which contains the **post_class** function as follows:

`post_class()` was changed to `post_class('page-landing')`

post_class() is a WordPress function which adds CSS classes to html elements.

It gives theme and plugin developers greater options for CSS styling. By default this function's parameter is optional, but in our case we are passing the parameter of our new classname called '**page-landing**'. This will allow us to apply styling to the <article> tag later on.

3) We also made changes to the line which prints the <h1> content.

The original line looked like this:

```
<h1 class="entry-title"><a href="<?php the_permalink(); ?>" title="<?php
printf( esc_attr__( 'Permalink to %s', 'twentyeleven' ),
the_title_attribute( 'echo=0' ) ); ?>" rel="bookmark"><?php the_title();
?></a></h1>
```

In our case we didn't need anchor tags and or permalinks because all we want is to display our headline and be able to style it. Therefore our version of the above line of code is as follows:

```
<h1 class="landing-heading"><?php the_title(); ?></h1>
```

As you can see we've changed the class name to our own unique class called "landing-heading" which we will use to style our headline later.

We've also removed the unwanted anchor tag stuff and simply left the code which prints the title (or headline) of our page.

Step 5:

Next we will need to modify the code inside the **header-landing.php** file which contains the original code from the header.php we copied earlier.

Inside this file we will remove all code between and including the **<header>** tags because it is related to site banner/header-image, site heading/description, search form, navigation etc and these components will not be required for our landing page style template.

(To examine the final contents of the **header-landing.php** file see our zip file download for this tutorial)

Step 6:

Modify the code inside the **footer-landing.php** file which contains the original code from the footer.php we copied earlier.

In this file we removed components which were not required such as footer sidebar related stuff and also the code which prints the WordPress credits.

(To examine the final contents of the **footer-landing.php** file see our zip file download for this tutorial)

Step 7:

At this point we need to do some styling by adding some css code to our child theme's style.css file.

We want to add some color to headline and also set its font size and other related parameters.

We also want to reduce any unnecessary padding or white space from the top of the page to the headline.

Thus our style.css file will now look like this:

```
/*
Theme Name:     twentyeleven-landing
Theme URI:      http://yoursitename.com/
Description:    Child theme for the Twenty Eleven theme
Author:         Your Name
Author URI:     http://yoursitename.com/
Template:       twentyeleven
Version:        1.0
*/

@import url("../twentyeleven/style.css");

.landing-heading {
clear: both;
color: #035492;
font-size: 48px;
font-weight: bold;
line-height: 1.3em;
padding-bottom: .3em;
padding-top: 15px;
text-align: center;
}

.singular.page .page-landing {
padding: 0.5em 0 0;
}
```

So now that we have completed our child theme we can ftp our folder which we created in step 1 and which contains our child theme's files to our WordPress host's "**/wp-content/themes/**" directory.

We can then activate our child theme by going to the **Appearance->Themes** menu from the WP administration page and clicking the Activate link as shown below:

After activating the new child theme, you can now build pages which use the newly created template.

To do this you simply select the new page template from your WordPress page editor by clicking the template dropdown box as shown in the figure below and then saving your page.

Although this tutorial has demonstrated relatively basic changes to a page's display, we hope it will give you the confidence to further explore the many possibilities you have at your finger tips when developing within WordPress.