

# Creating Custom Content in WordPress: Taxonomies and Fields

---

**In the first part of this two-part series<sup>[1]</sup>, we explored the three different types of custom content you can create with WordPress and looked at the uses of each of them.**

These are:

- Custom posts
- Custom taxonomies
- Custom fields

That post also looked at custom post types in more detail, explaining how to create them and display them on your site.

In this second and final part of the series I'll move on to custom taxonomies and custom fields: How to create each of them either using a plugin or by coding your own, and how to display them on your site's front-end. Let's start with custom taxonomies.

## Creating a Custom Taxonomy

As with custom post types, you can either use a plugin to create a custom taxonomy or code your own in your theme or (even better) write a plugin. The plugins you can use to do this are the same as for custom post types:

- **CustomPress<sup>[2]</sup>** gives you an interface for creating custom post types, taxonomies and custom fields and is very user-friendly. If you want to create multiple types of custom content, this saves you installing more than one plugin.
- **Custom Post Type UI<sup>[3]</sup>** is the most popular free plugin for adding custom post types and taxonomies on the WordPress plugin repository. It lets you add post types and taxonomies, but not custom fields. The interface isn't

quite as user-friendly as with CustomPress, with more technical terminology for you to get your head round, but it does make the process easier.

If you want to know more about plugins to help you create custom content, see our review of the best CMS plugins<sup>[4]</sup>.

But if you're confident enough to code your own custom taxonomies, here's how you do it.

## Coding a Custom Taxonomy

Coding a custom taxonomy isn't too different from coding a custom post type: WordPress gives you the `register_taxonomy()`<sup>[5]</sup> function to do this which you also attach to the `init` hook.

You can either add your code to your theme's functions file or you can create a separate plugin. It's better to create a plugin as then if you change your theme in future you won't lose your taxonomy. So that's what we'll do here.

Start by creating an empty file called `taxonomies.php` and saving it to the `plugins` folder in your `wp-content` directory.

Now open your file in a code editor and add the following to it:

```
<?php
/*Plugin Name: Create Product Category Taxonomy
Description: This plugin registers the 'product category' taxonomy and applies
it to the 'product' post type.
Version: 1.0
License: GPLv2
?>
```

This tells WordPress what your plugin is called and provides a description which you'll see when you activate the plugin later.

Now below the line which reads `*/` and above the `?>` line, add this code:

```
function wpmudev_register_taxonomy() {  
    add_action( 'init', 'wpmudev_register_taxonomy' );  
}
```

This creates an empty function which will contain the code for your taxonomy, and attaches it to the `init` hook. Attaching it to a hook like this makes sure that WordPress runs your function at the right time: without it, nothing will happen.

Now, inside the curly braces, add this:

```
// set up labels  
$labels = array(  
    'name' => 'Product Categories',  
    'singular_name' => 'Product Category',  
    'search_items' => 'Search Product Categories',  
    'all_items' => 'All Product Categories',  
    'edit_item' => 'Edit Product Category',  
    'update_item' => 'Update Product Category',  
    'add_new_item' => 'Add New Product Category',  
    'new_item_name' => 'New Product Category',  
    'menu_name' => 'Product Categories'  
);  
  
// register taxonomy  
register_taxonomy( 'productcat', 'product', array(  
    'hierarchical' => true,  
    'labels' => $labels,  
    'query_var' => true,  
    'show_admin_column' => true  
));
```

Let's work through that code to understand better what it does.

First, it creates a set of labels for your taxonomy which will be used in the admin screens, on links, buttons and page titles. Next it uses the `register_taxonomy()`

function to register your taxonomy, with these parameters:

- The name of the taxonomy, `productcat` – you need this for your taxonomy to work.
- The post type your taxonomy will apply to. Here I've specified this as `product`, but you could use your taxonomy with more than one post type (including built in post types like posts, pages and attachments) by including them all in an array.
- `hierarchical`: set this to `true` for your taxonomy to behave like categories and be hierarchical. If you want it to not be hierarchical, like tags, set it to `false`.
- `labels`: here you're telling WordPress to use the labels already defined
- `query_var`: setting this to `true` gives you more flexibility if you want to write a custom query<sup>[6]</sup> to find products with a given taxonomy term.
- `show_admin_column`: setting this to `true` means that when you display a list of your products in the WordPress admin, the taxonomy will have a column for you to see which taxonomy terms apply to which products. I'll show you what this looks like shortly.

Now save your file. The entire plugin will look like this:

```
<?php

/*Plugin Name: Create Product Category Taxonomy

Description: This plugin registers the 'product category' taxonomy and applies
it to the 'product' post type.

Version: 1.0

License: GPLv2

// register two taxonomies to go with the post type

function wpmudev_register_taxonomy() {

// set up labels

$labels = array(

'name' => 'Product Categories',

'singular_name' => 'Product Category',

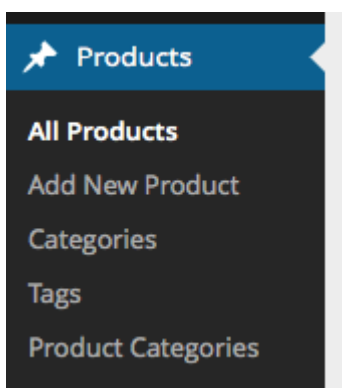
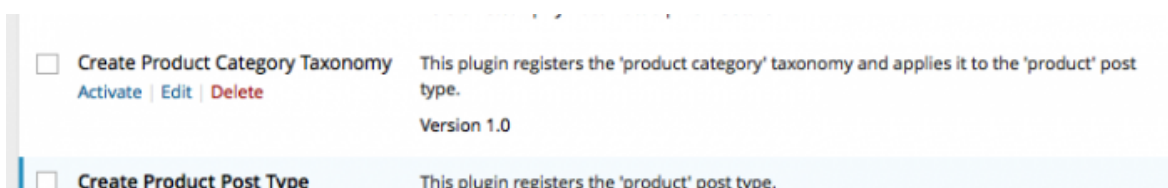
'search_items' => 'Search Product Categories',

'all_items' => 'All Product Categories',

'edit_item' => 'Edit Product Category',
```

```
'update_item' => 'Update Product Category',  
'add_new_item' => 'Add New Product Category',  
'new_item_name' => 'New Product Category',  
'menu_name' => 'Product Categories'  
  
// register taxonomy  
register_taxonomy( 'productcat', 'product', array(  
    'hierarchical' => true,  
    'labels' => $labels,  
    'query_var' => true,  
    'show_admin_column' => true  
));  
add_action( 'init', 'wpmudev_register_taxonomy' );  
?>
```

Now that your plugin is complete you need to activate it. In the WordPress admin, go to **Plugins -> Installed Plugins** to see your plugin listed:



Click the **Activate** Link to activate your plugin. You'll now see that when you click on **Products** in your admin menu, the new taxonomy will be displayed:

Now all you need to do is add some product categories in the same way as you would add normal categories. You'll then be able to select one or more of them

every time you create a product.

*Note: Your new taxonomy is not a category, instead it's a taxonomy in the same way as 'category' is. To use it you'll have to add terms. Your taxonomy's terms (i.e. the product categories you add) are similar to categories themselves, which are terms in the 'category' taxonomy.*

## Displaying Taxonomy Term Archives on Your Site

Once you've added some terms to your taxonomy, you'll need to display a list of products with each term on your site.

### Refreshing Permalinks

WordPress won't display the archives for your taxonomy term unless you refresh the permalinks settings.

1. In the WordPress admin, go to **Settings > Permalinks**.
2. Check that the 'post name' radio button is selected.
3. Click on the **Save Changes** button.

Now WordPress will be able to generate the correct URLs for your taxonomy terms.

*Note: you only need to do this once after creating the taxonomy, not each time you add a term.*

### Adding Taxonomy Term Archives to the Navigation Menu

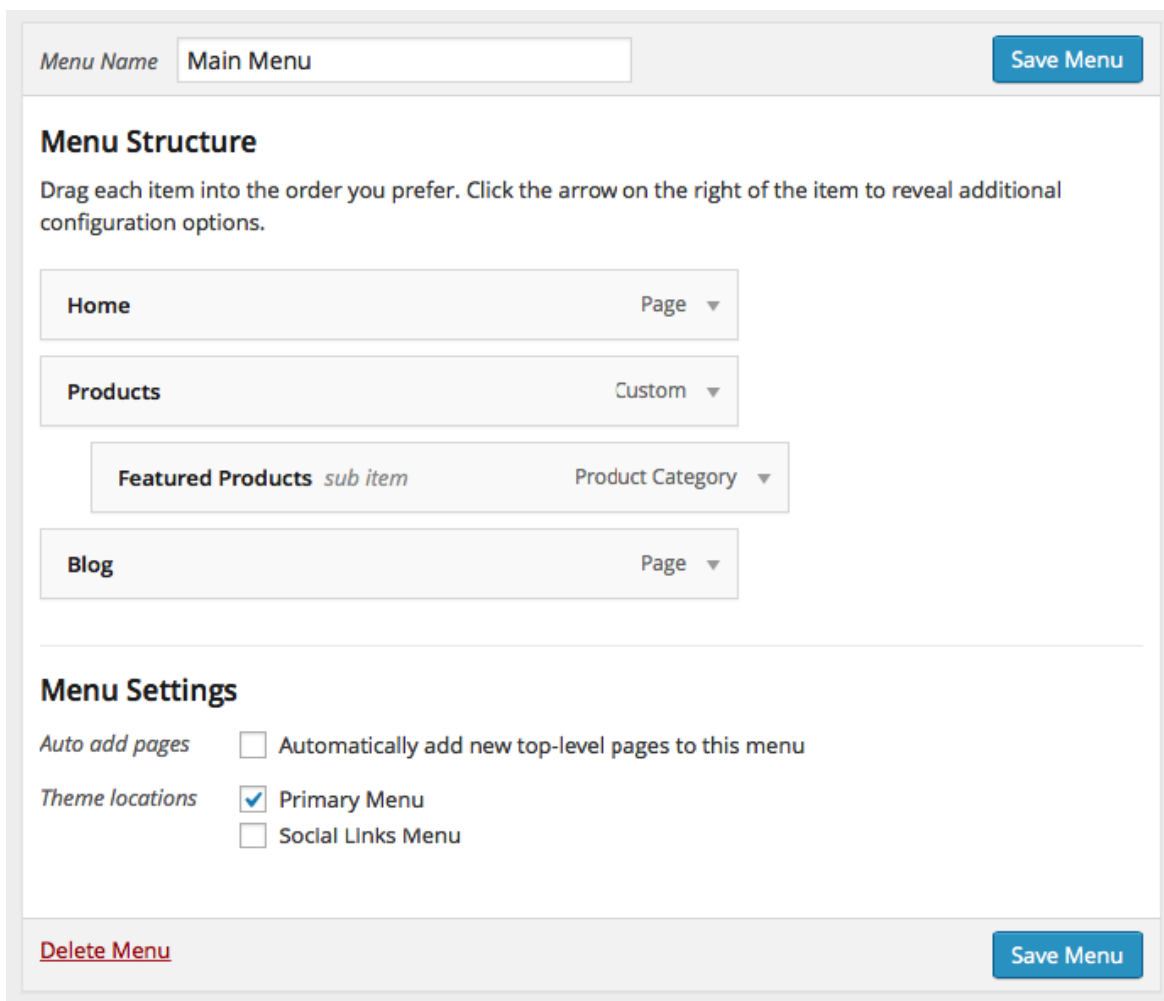
Adding archive pages for your taxonomy to the navigation menu is easier than adding post type archives (as we did in the last part of this series). Follow these steps:

1. In the WordPress admin, go to **Appearance > Menus**.
2. On the left of the screen you'll see a list of content types you can add to your menu. One of these will be 'Product Categories' (if it isn't visible, click on the **Screen Options** tab at the top right of the screen then tick the checkbox for

product categories).

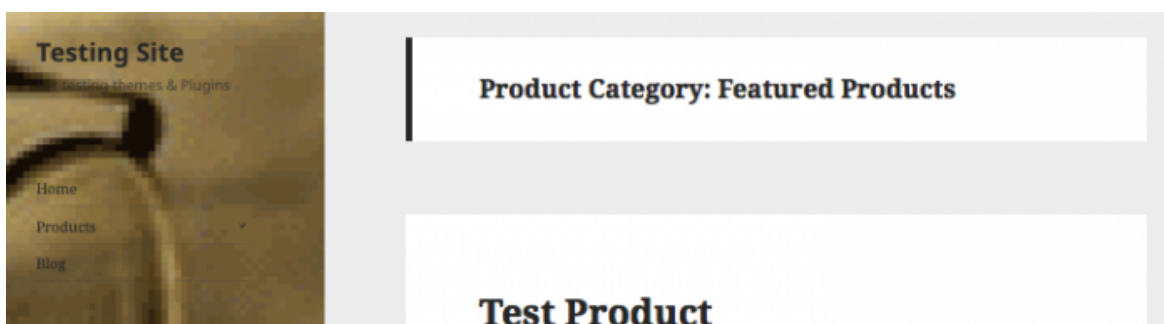
3. Click on **Product Categories** to reveal all of the terms you've added.
4. Add each of them to the menu and drag them into the place you want them.
5. Click on **Save Menu** to save your changes. Don't miss this step!

I've added one product category and inserted it in the menu under the 'Products' link I created in the first part of this series:



The screenshot shows the WordPress Menu Editor for the 'Main Menu'. At the top, there's a 'Menu Name' field with 'Main Menu' and a 'Save Menu' button. Below is the 'Menu Structure' section with instructions: 'Drag each item into the order you prefer. Click the arrow on the right of the item to reveal additional configuration options.' The menu items are: 'Home' (Page), 'Products' (Custom), 'Featured Products' (sub item, Product Category), and 'Blog' (Page). Below the menu structure is the 'Menu Settings' section with options: 'Auto add pages' (unchecked), 'Automatically add new top-level pages to this menu' (unchecked), 'Theme locations' (checked for 'Primary Menu', unchecked for 'Social Links Menu'). At the bottom, there's a 'Delete Menu' link and another 'Save Menu' button.

Now when you go to your site's front end and click on the links to your taxonomy term archives you'll see a list of products with that term:





This displays my products but might not display them in the way I want to.

## Using Template Files for Term Archives

If you want to display the archives for your taxonomy terms differently from the way your site displays other archives, you can create a special template file in your theme, or even a set of template files with one for each term.

When WordPress displays the archive page for a taxonomy term, it looks through the files in your theme using the template hierarchy<sup>[7]</sup>, and uses the first one it comes across:

1. A file for displaying the archive for a specific taxonomy term: for my featured products term, this would be called `taxonomy-productcat-featured-products.php` (a bit of a mouthful, I know!). Note that you use the name of the taxonomy followed by the slug of the taxonomy term to create the file name.
2. A file for displaying taxonomy term archives for all terms in a taxonomy, called `taxonomy-productcat.php`.
3. A file for displaying archives for all custom taxonomies, called `taxonomy.php`.
4. The file for displaying all kinds of archive, called `archive.php`
5. The generic file for all content, called `index.php`.

WordPress will work through this list and use the first one it finds. In my site I'm running the twenty fifteen theme which has a file called `archive.php`, so it uses that.



If you wanted to display your taxonomy archives differently from other archives (for example to take out excerpts and add featured images), you could create a copy of the `archive.php` file in your theme, call it `taxonomy.php` and edit it to display your archive in the way you wanted. But even if you don't do this, the good news is that WordPress will always find a suitable template file to display your term archives.

Now let's move on to the final custom content type: custom fields.

## Creating Custom Fields

You can create custom fields in one of two ways: using the default interface provided by WordPress or by using a plugin. Using a plugin can be more user-friendly and offer you some more powerful options for editing and displaying your custom fields, but isn't always necessary.

## Sign up for more WordPress wisdom

Among the best plugins for creating custom fields are:

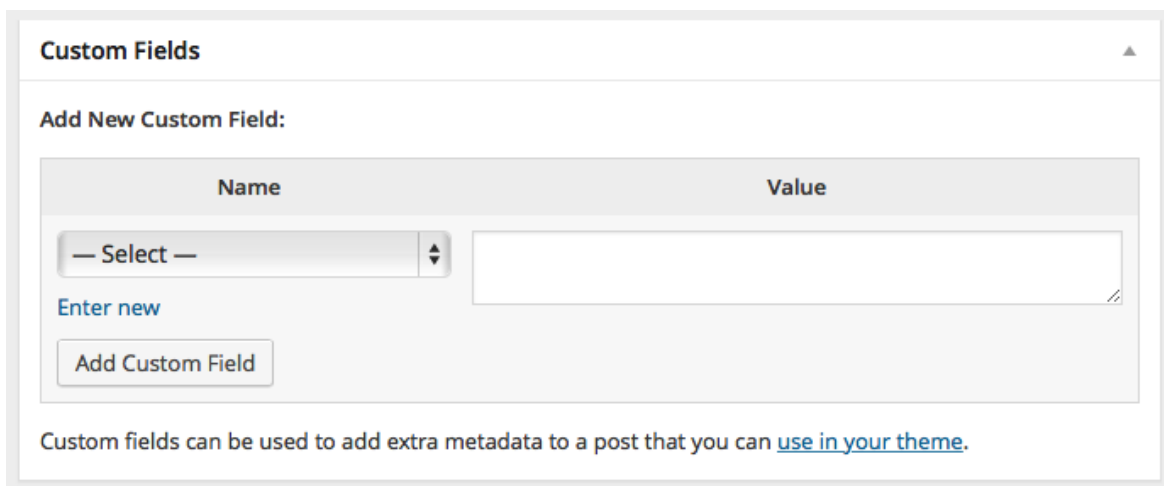
- **CustomPress<sup>[8]</sup>**: As well as letting you create custom post types and taxonomies, you can also use this plugin to create custom fields.
- **Advanced Custom Fields<sup>[9]</sup>**: This is the most popular plugin for custom fields and the basic version is free on the WordPress plugin repository. It offers some powerful features for editing and creating bespoke custom fields.

If you don't want to use a plugin however, you can create custom fields using the WordPress admin. Here's how:

## Creating Custom Fields Using the Default Admin Screens

If you go to the post editing screen for one of your posts (or you're creating a new post), you might see a metabox for custom fields below the main editing pane. If this isn't visible, you can display it by opening the **Screen Options** tab at the top right of the screen. Click on the 'Custom Fields' tick box, close the tab and the

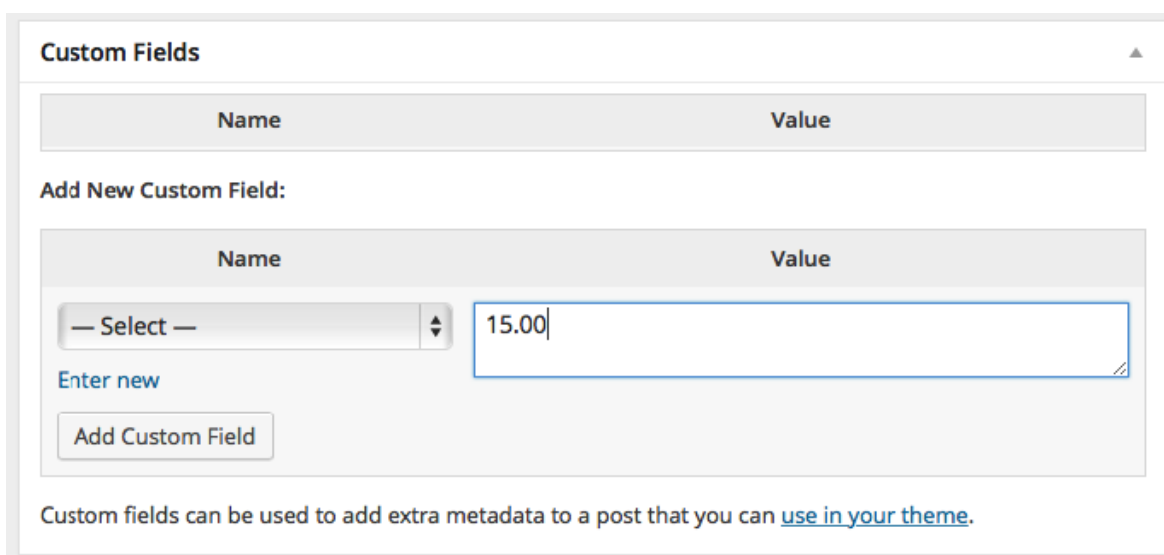
metabox will appear below your content:



The screenshot shows the 'Custom Fields' metabox in WordPress. It has a title bar 'Custom Fields' with an upward arrow. Below it is the section 'Add New Custom Field:'. Inside this section is a form with two columns: 'Name' and 'Value'. Under 'Name', there is a dropdown menu currently showing '— Select —' and a small 'Enter new' link below it. Under 'Value', there is a large text input field. At the bottom of the form is a button labeled 'Add Custom Field'. Below the form, there is a note: 'Custom fields can be used to add extra metadata to a post that you can [use in your theme](#).'

To add a custom field, you either select an existing key from the dropdown box on the left hand side, or add a new one by clicking the **Enter New** link and typing it in. Then you type in the value for your custom field.

I've created a custom field for a product I've added whose key is 'Price' and whose value is '15.00':



This screenshot shows the 'Custom Fields' metabox after a custom field has been added. The 'Add New Custom Field:' section is still visible, but the 'Name' dropdown now shows 'Price' (though it's partially obscured by the '15.00' text in the value field). The 'Value' field now contains the text '15.00'. The 'Add Custom Field' button is still present. The note at the bottom remains the same: 'Custom fields can be used to add extra metadata to a post that you can [use in your theme](#).'

Once you've added the key and value, you must click the **Add Custom Field** button to save your custom field: it won't just save when you hit the **Publish** button for your post.

You can add as many custom fields to your posts as you need to store your data. Don't forget to save your post after you've saved your custom fields.

So that's how you add a custom field using the default admin interface. Now let's take a look at how you display your custom fields on your site's front end.

## Displaying Custom Fields on Your Site

By default, WordPress won't display the values you've entered in your custom fields unless your theme is set up to do this. You might find that your theme comes with custom field support and outputs all your custom fields on the single page for each post, but the chances are it won't.

So you'll need to add a template tag<sup>[10]</sup> to do this.

*A template tag is a type of function which you insert into a theme file to tell WordPress to display some data.*

WordPress gives you a choice of template tags for displaying your custom fields:

- `the_meta()`<sup>[11]</sup> displays all of the custom fields for a post. You use it in the loop and it will output all custom fields' keys and values without giving you much control.
- `get_post_meta()`<sup>[12]</sup> gives you more control. It has parameters you can use to specify which custom fields you want to get and whether you just want to retrieve one value if your post has multiple custom fields with the same key. It doesn't actually output your custom fields though, just fetches them: to display your custom fields you have to use `echo get_post_meta()`.
- `get_post_custom_values()`<sup>[13]</sup> lets you fetch all the values for custom fields with the same key, which you specify. Again it needs to have `echo` before it to output anything.

You normally add one of these to the template file in your theme which displays single posts or archives, inside the loop. Alternatively if you wanted to list custom fields for a number of posts elsewhere in the site, you could use `get_post_meta()` outside the loop.

For now we won't worry about that – instead we'll look at how you do this in the loop.

*If you haven't come across the loop before, it's the block of code in your theme*

*template files which fetches the title and content of each post from the database and displays it.*

The first step is to open your `single.php` file in a code editor and find the loop. It will start with a line of code that looks something like this:

```
while ( have_posts() ) : the_post();
```

Below that you'll find template tags such as `the_title()` and `the_content()`. You need to add your custom fields between these: I'm going to add mine immediately after the title.

*Note: If you're working with twenty fifteen, you'll find that it doesn't have the loop cooked in the `single.php` file: instead it's in an include file called `content.php`. To edit the display for the product post type, create a copy of `content.php`, name it `content-product.php`, and edit that.*

To display all custom fields for the post, insert this code:

Save your template file.

Now when I visit my site's front end, I can see the custom field I added to my product:



By default this is shown in a bulleted list: you can style this list however you want using CSS.

But what if I want bit more control over the way my custom field is shown? I can use `get_post_meta()`.

Open your `single.php` file again and find the code you just added. Replace it with this:

```
$price = get_post_meta( get_the_ID(), 'Price', true );  
if( ! empty( $price ) ) {  
    echo '<p>Price: $' . $price . '</p>';  
}
```

Let's take a look what this does:

1. First it gets the custom field for this post with the key 'Price', and stores it as a variable called `$price`.
2. It checks that the custom field isn't empty
3. If not, it outputs the price inside a paragraph tag with some text before it.

Now save your file and take a look at your page. Mine looks like this:



As you can see, that's given me more control. Not only can I specify which custom field I want to display, I can add text before it, put it inside a paragraph element if I want to, and if I have more custom fields to show, I can repeat similar code in any order I like.

*Note: As I'm using the Twenty Fifteen theme, for this example I didn't directly edit the file in my theme but instead I created a child theme and made new files. This means that when I update Twenty Fifteen, the new files I've added won't be lost. If you're going to do the same, I suggest you follow this great guide to child themes<sup>[14]</sup>.*

## Summary

If you've been following both parts of this series, you'll know how useful and powerful custom content can be in WordPress. Each of custom post types, custom taxonomies and custom fields has its own uses and methods for getting the most from the content type.

In this second part, you've learned how to register a custom taxonomy and display its archive pages on your site; and how to create custom fields and display those on your site too.

**Do you use custom taxonomies and custom fields in your WordPress site? What do you find them most useful for? Have your say in the comments.**

## Sign up for more WordPress wisdom

1. <http://premium.wpmudev.org/blog/creating-content-custom-post-types/>
2. <http://premium.wpmudev.org/project/custompress/>
3. <https://wordpress.org/plugins/custom-post-type-ui/>
4. <http://premium.wpmudev.org/blog/top-wordpress-cms-plugins-reviewed/>
5. [http://codex.wordpress.org/Function\\_Reference/register\\_taxonomy](http://codex.wordpress.org/Function_Reference/register_taxonomy)
6. <http://premium.wpmudev.org/blog/creating-custom-queries-wordpress/>
7. [http://codex.wordpress.org/Template\\_Hierarchy](http://codex.wordpress.org/Template_Hierarchy)
8. <http://premium.wpmudev.org/project/custompress/>
9. <https://wordpress.org/plugins/advanced-custom-fields/>

10. [http://codex.wordpress.org/Template\\_Tags](http://codex.wordpress.org/Template_Tags)
11. [http://codex.wordpress.org/Function\\_Reference/the\\_meta](http://codex.wordpress.org/Function_Reference/the_meta)
12. [http://codex.wordpress.org/Function\\_Reference/get\\_post\\_meta](http://codex.wordpress.org/Function_Reference/get_post_meta)
13. [http://codex.wordpress.org/Function\\_Reference/get\\_post\\_custom\\_values](http://codex.wordpress.org/Function_Reference/get_post_custom_values)
14. <http://premium.wpmudev.org/blog/how-to-create-wordpress-child-theme/>