**DIGM-243: Web Authoring II**

**Week 06**

**Customizing WordPress with**

**Thumbnails, Post Types, Taxonomies & Fields**

**Post Thumbnails**

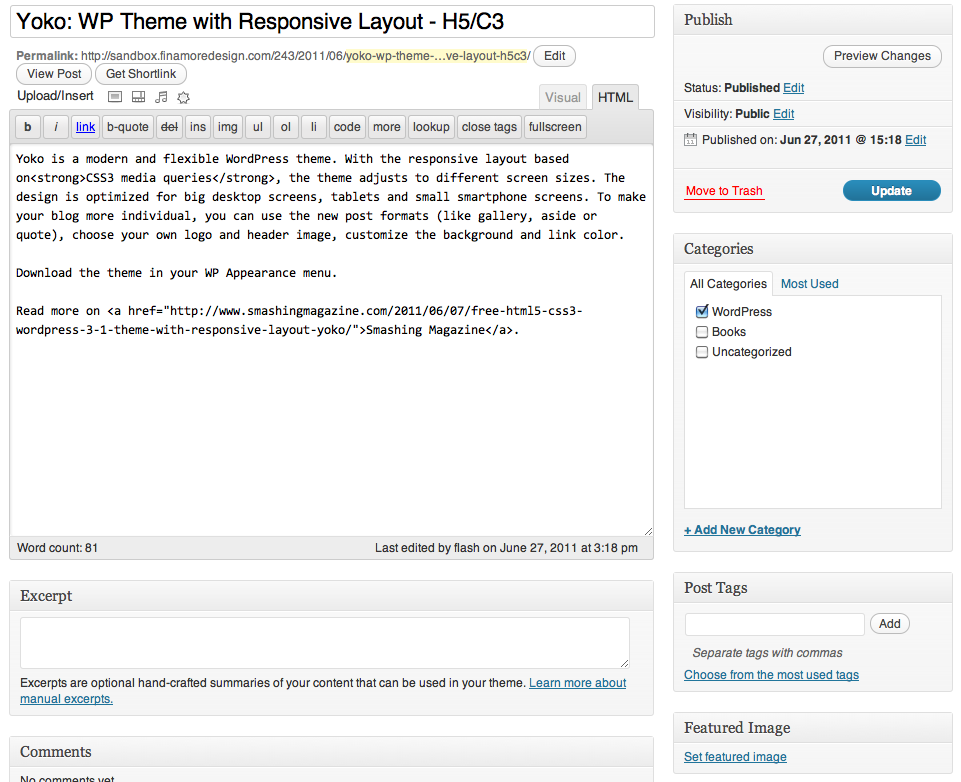
WordPress has the functionality to define a thumbnail image to each page or post. This is a special feature that needs to be activated in your theme. To add support for thumbnails to your theme you must declare that your theme supports this feature in the theme’s functions.php file:

if ( function\_exists( 'add\_theme\_support' ) ) {

add\_theme\_support( 'post-thumbnails' );

}

This code will enable the Post Thumbnail UI for both Post and Page content types. It will appear in the right side bar under Post Tags as Featured Image.



If you’d only like to add it to only one taxonomy, you modify the add\_theme\_support line as follows:

add\_theme\_support( 'post-thumbnails', array( 'post' ) );

// Adds thumbnails support for posts only

add\_theme\_support( 'post-thumbnails', array( 'page' ) );

// Adds thumbnails support for pages only

Now, you can make use of the template functions to display thumbnail images in your theme. These functions should be used in the loop. If you want to show the thumbnail you can simply use the\_post\_thumbnail() to output the Post Thumbnail, if it exists:

<?php the\_post\_thumbnail(); ?>

But what if you failed to set a Featured Image? Then nothing will show. If your theme is designed around showing thumbnails you can use an if statement testing for has\_post\_thumbnail() which returns true/false and indicates whether the current post has a manually-chosen Post Thumbnail

<?php

if ( has\_post\_thumbnail() ) {

// the current post has a thumbnail

the\_post\_thumbnail();

} else {

// the current post lacks a thumbnail

// display something else

}

?>

The default image sizes are "thumbnail", "medium", "large" and "full" (the image you uploaded). These image sizes can be configured in the admin Media panel under Settings > Media. This is how you use these default sizes with the\_post\_thumbnail():

the\_post\_thumbnail();

// without parameter -> Thumbnail

the\_post\_thumbnail('thumbnail');

// Thumbnail (default 150px x 150px max)

the\_post\_thumbnail('medium');

// Medium resolution (default 300px x 300px max)

the\_post\_thumbnail('large');

// Large resolution (default 640px x 640px max)

You could also specify the dimensions of your post thumbnails in the functions.php file. You have two options here: box-resizing and hard-cropping.

**Box resizing** shrinks an image proportionally, until it fits inside the “box” you’ve specified with your width and height parameters. For example, a 100×50 image in a 50×50 box would be resized to 50×25. The benefit here is that the entire image shows. The downside is that the image produced isn’t always the same size. Sometimes it will be width-limited, and sometimes it will be height-limited. If you’d like to limit images to a certain width, but don’t care how tall they are, you can specify your width and then specify a height of 9999 or something ridiculously large that will never be hit.

set\_post\_thumbnail\_size( 50, 50 );

// 50 pixels wide by 50 pixels tall, box resize mode

Your second option is **hard-cropping**. In this mode, the image is cropped to match the target aspect ratio, and is then shrunk to fit in the specified dimensions exactly. The benefit is that you get what you ask for. If you ask for a 50×50 thumbnail, you get a 50×50 thumbnail. The downside is that your image will be cropped (either from the sides, or from the top and bottom) to fit the target aspect ratio, and that part of the image won’t show up in the thumbnail.

set\_post\_thumbnail\_size( 50, 50, true );

// 50 pixels wide by 50 pixels tall, hard crop mode

Those are the basics. How about some advanced stuff?

What if you want to use a small 50×50 hard-cropped image for the home page, but want to use a 400 pixel-wide (unlimited height) image on the post’s permalink page? You’re in luck. You can specify additional custom sizes! Here’s the code for your functions.php

if ( function\_exists( 'add\_theme\_support' ) ) {

add\_theme\_support( 'post-thumbnails' );

set\_post\_thumbnail\_size( 50, 50, true );

// Normal post thumbnails

add\_image\_size( 'single-post-thumbnail', 400, 9999 );

// Permalink thumbnail size

}

On your home.php or index.php, depending on your theme structure (in the loop):

<?php the\_post\_thumbnail(); ?>

single.php (in the loop):

<?php the\_post\_thumbnail( 'single-post-thumbnail' ); ?>

That’s it! set\_post\_thumbnail\_size() just calls add\_image\_size( 'post-thumbnail' ) — the default Post Thumbnail “handle.” But as you can see, you can add additional ones by calling add\_image\_size( $handle, $width, $height, {$hard\_crop\_switch} );, and then you use that new size by passing the handle to the\_post\_thumbnail( $handle );

There is one caveat for this feature — it only works fully for new image uploads. We can’t yet resize images on the fly. If you call the template functions on a post that has a Post Thumbnail that was uploaded prior to your theme having declared the new sizes, you won’t be able to do hard-cropping, and the box-resize will be done in the browser. As a temporary solution, Viper007Bond has a great plugin that will go back and create missing image sizes for you: Regenerate Thumbnails <http://wordpress.org/extend/plugins/regenerate-thumbnails/>

**Styling Post Thumbnails**

Post Thumbnails are given a class "wp-post-image". They also get a class depending on the size of the thumbnail being displayed You can style the output with these CSS selectors:

img.wp-post-image {}

img.attachment-thumbnail {}

img.attachment-medium {}

img.attachment-large {}

img.attachment-full {}

You can also give Post Thumbnails their own class.

Display the Post Thumbnail with a class "alignleft":

<?php the\_post\_thumbnail('thumbnail', array('class' => 'alignleft')); ?>

**Post Types**

WordPress can hold and display many different types of content. Internally, these are all stored in the same place, in the wp\_posts table. These are differentiated by a column called *post\_type*.

There are five major types that WordPress uses by default.

1. **Post**  
   A "post" in WordPress is the main type used by the blog. Posts are normally displayed in the blog in reverse sequential order by time (newest posts first). Posts are also used for creating the feeds.
2. **Page**  
   A "page" in WordPress is like a post, but it lives outside the normal time based structure of posts. They have their own URLs directly off the main site URL. They can also use special Page Templates to display them. Pages can also be organized in a hierarchical structure, with Pages being parents to other Pages.
3. **Attachment**  
   An "attachment" is a special post that holds information about files uploaded through the Media upload system. They hold all the description and name and other information about uploaded files. For images, this is also linked to metadata information about the size of the images and thumbnails generated from the images, the location of the files, and even information obtained from EXIF data embedded in the images.
4. **Revisions**A "revision" is used to hold draft posts as well as any past revisions of existing posts or pages. These are basically identical to the main post/page that they are for, but have that post/page as their parent.
5. **Nav Menus**The "nav\_menu" type holds information about a single item in the Navigation Menu system. These are the first examples of entries in the posts table to be used for something other than an otherwise displayable content on the blog.

**Custom Post Types**

It is possible to add your own custom post type to WordPress. This is done via the register\_post\_type function. This function allows you to define the post type and how it operates within WordPress.

**Taxonomies**

A *taxonomy* is a group of terms. Example: Post Tags is a taxonomy, each tag is a term within the taxonomy. Each taxonomy and individual term has a name, a slug, and a description.

The name field defines what will be displayed on your page.

Slugs must be unique. The Slug is used in the URL. For example, setting a Category Name of "Recipes" and a Category Slug of "food," the page would display all "Recipes" posts with a URL like *yourdomain.com/blog/food/*

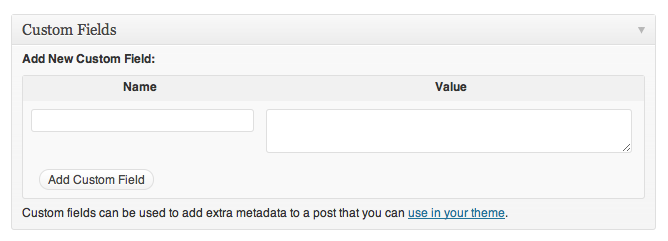
Descriptions are optional. Some themes take advantage of descriptions, others do not, so having Descriptions may or may not modify the way Categories are displayed for your site. Even if they are not used by your site's theme, you may still find them useful from an administration point of view.

In order to have our own post types and taxonomys we will need to write our own functions to add this functionality to our theme.

Remember functions only affect the specific theme. If you add functions to one theme then switch themes those function will need to be copied over in order to still work.

**Custom Fields Forever**

WordPress has the ability to allow you to assign custom meta data to a post via the built in custom fields.



This meta-data can include bits of information such as:

Mood: Somber

Genre: Minimalistic

Rating: 5 Stars

Client: Difficult

Location: East Coast

Software: Illustrator

Meta-data is handled with key/value pairs. The key is the name of the meta-data element. The value is the information that will appear in the meta-data list on each individual post that the information is associated with. Keys can be used more than once per post.

**Do It Yourself**

**Writing Post Type and Taxonomy Functions**

We will create a new post type called Work so that we can build Work pages to showcase our artwork. We will then create two custom taxonomies so that we can help the viewer understand what it is that they are looking at. The first taxonomy we will create will be hierarchical (like Categories) But we will call this Type so that we can list them by type of work. The second will non-hierarchical (like Tags) so that we can give them some subsidiary info. We will call this Clients so that you can tag work by client.

You do not have to use these two names. You do not have to use both methods. You can name them anything that you think will make better organizational sense to your users. Heck, you don't even have to use the custom page type. You may not want a blog and just end up using the Posts for your Portfolio.

To create a new taxonomy you will need to create a functions.php with the following code:

<?php

//hook into the init action and call register\_work when it fires

add\_action('init', 'register\_work', 1); // Set priority to avoid plugin conflicts

function register\_work() { // A unique name for our function

$labels = array( // Used in the WordPress admin

'name' => \_x('Work', 'post type general name'),

'singular\_name' => \_x('Work', 'post type singular name'),

'add\_new' => \_x('Add New', 'Work'),

'add\_new\_item' => \_\_('Add New Work'),

'edit\_item' => \_\_('Edit Work'),

'new\_item' => \_\_('New Work'),

'view\_item' => \_\_('View Work '),

'search\_items' => \_\_('Search Work'),

'not\_found' => \_\_('Nothing found'),

'not\_found\_in\_trash' => \_\_('Nothing found in Trash')

);

$args = array(

'labels' => $labels, // Set above

'public' => true, // Make it publicly accessible

'hierarchical' => false, // No parents and children here

'menu\_position' => 5, // Appear right below "Posts"

'has\_archive' => 'resources', // Activate the archive

'supports' => array('title','editor','comments','thumbnail','custom-fields'),

);

register\_post\_type( 'work', $args ); // Create the post type, use options above

}

//hook into the init action and call create\_work\_taxonomies when it fires

add\_action( 'init', 'create\_work\_taxonomies', 0 );

//create two taxonomies, type and client for the post type "work"

function create\_work\_taxonomies()

{

// Add new taxonomy, make it hierarchical (like categories)

$labels = array(

'name' => \_x( 'Types', 'taxonomy general name' ),

'singular\_name' => \_x( 'Type', 'taxonomy singular name' ),

'search\_items' => \_\_( 'Search Types' ),

'all\_items' => \_\_( 'All Types' ),

'parent\_item' => \_\_( 'Parent Type' ),

'parent\_item\_colon' => \_\_( 'Parent Type:' ),

'edit\_item' => \_\_( 'Edit Type' ),

'update\_item' => \_\_( 'Update Type' ),

'add\_new\_item' => \_\_( 'Add New Type' ),

'new\_item\_name' => \_\_( 'New Type Name' ),

'menu\_name' => \_\_( 'Type' ),

);

register\_taxonomy('type',array('work'), array(

'hierarchical' => true,

'labels' => $labels,

'show\_ui' => true,

'query\_var' => true,

'rewrite' => array( 'slug' => 'type' ),

));

// Add new taxonomy, NOT hierarchical (like tags)

$labels = array(

'name' => \_x( 'Clients', 'taxonomy general name' ),

'singular\_name' => \_x( 'Client', 'taxonomy singular name' ),

'search\_items' => \_\_( 'Search Clients' ),

'popular\_items' => \_\_( 'Popular Clients' ),

'all\_items' => \_\_( 'All Clients' ),

'parent\_item' => null,

'parent\_item\_colon' => null,

'edit\_item' => \_\_( 'Edit Client' ),

'update\_item' => \_\_( 'Update Client' ),

'add\_new\_item' => \_\_( 'Add New Client' ),

'new\_item\_name' => \_\_( 'New Client Name' ),

'separate\_items\_with\_commas' => \_\_( 'Separate Clients with commas' ),

'add\_or\_remove\_items' => \_\_( 'Add or Remove Clients' ),

'choose\_from\_most\_used' => \_\_( 'Choose from the most used Clients' ),

'menu\_name' => \_\_( 'Clients' ),

);

register\_taxonomy('client','work',array(

'hierarchical' => false,

'labels' => $labels,

'show\_ui' => true,

'query\_var' => true,

'rewrite' => array( 'slug' => 'client' ),

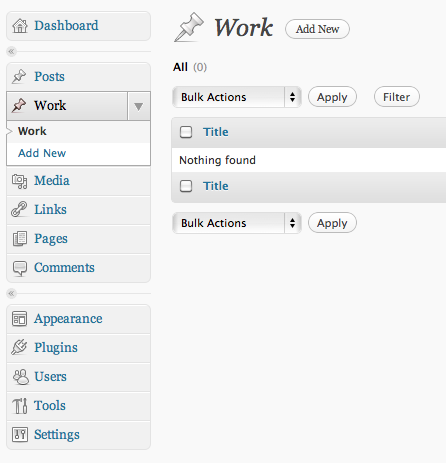
));

}

?>

The code above tells WordPress to “register” a post type called “work”. Then, we pass in our options, letting WordPress know that we want to use our own labels, that we want our post type to be publicly accessible, non-hierarchal, and that we want it to show up right below “posts” in our admin menu. Then, we activate the “archive” feature, new in WordPress 3.1. Finally, we add in “supports”: the default title field, the WordPress editor, comments, featured thumbnail, and custom fields (I’ll explain that later). More options can be found at <http://codex.wordpress.org/Function_Reference/register_post_type>

You should now see your custom "Work" type just below "Posts" in the admin sidebar.



One thing you need to remember is that the functions.php file is specific to the theme. So if you change themes these custom taxonomies will not remain part of your site. If you want them to remain part of the site everytime you change themes you need to activate these via a plugin. Luckily this is easy enough to do. Just create a new php file and name is something unique like DIGMWorkTaxonomy.php

We need to start with some

<?php

/\*

Plugin Name: DIGM Work Taxonomy

Plugin URI: http://digm.drexel.edu/

Description: This plugin creates a custom post type and taxonomies.

Version: 0.1

Author: Troy Finamore

Author URI: http://digm.drexel.edu

License: A "Slug" license name e.g. GPL2

Copyright 2011 Troy Finamore (email : troy@finamoredesign.com)

This program is free software; you can redistribute it and/or modify

it under the terms of the GNU General Public License, version 2, as

published by the Free Software Foundation.

This program is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

GNU General Public License for more details.

You should have received a copy of the GNU General Public License

along with this program; if not, write to the Free Software

Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

\*/

add\_action('init', 'register\_work', 1); // Set priority to avoid plugin conflicts

function register\_work() . . .

And follow through with the rest of the previous sample code.

Upload this .php file into your wp-content/plugins folder, then when you go to the plugins page in your admin you should be able to activate it.

**Adding Custom Fields to your Post**

Working with custom fields is a lot easier than custom post types and taxonomies. It is already built into the system. After you have written your post, scroll down to the area titled *Custom Fields*.

**Note:** As of WordPress version 3.1, some screen options on the Post & Page edit Administration Panels are hidden by default. Custom Fields are hidden by default if they have not been used before.

To create a new Custom Field called "Software", enter the text "Software" (without the quotes) in the text entry field titled Key.

The newly created Key should now be assigned a Value, which in our case is the name of the software package we used to build the work, "Adobe Illustrator". Type "Adobe Illustrator" in the Value field (again, without the quotes).

Click Add Custom Field button to save this custom information for that post.

If you used more than one software package you can create another key/value set. The Key should now feature a pull down list with the previously entered Custom Fields (you may need to refresh the screen). Choose "Software" and then enter the correct info in the value. Click Add Custom Field to complete the process.

You only need to create a new "KEY" once, after which you can assign a value to that key for every post, or not.

**Displaying Custom Fields**

With a Custom Field added to the post, it's time to display your books and mood to the world. To display the Custom Fields for each post, use the the\_meta() template tag. The tag must be put within The Loop in order to work. Many people add the\_meta() template tag to the end of their post or in their Post Meta Data Section. Here is a basic example of using the tag:

<?php the\_meta(); ?>

It might look like this in the source code:

<ul class='post-meta'>

<li><span class='post-meta-key'>Software:</span> Adobe Illustrator</li>

<li><span class='post-meta-key'>Software:</span> Adobe Flash</li>

</ul>

The template tag automatically puts the entire meta-data into a CSS style called post-meta. The key is in a span called post-meta-key so you can style it in your style sheet. All of this is showcased in an unordered list.

To add it into our sample we would simply do the following:

<header>

<h1><a href="<?php the\_permalink() ?>" rel="bookmark" title="Permanent Link to <?php the\_title\_attribute(); ?>"><?php the\_title(); ?></a></h1>

<p>Posted on <?php the\_time('F jS, Y'); ?> by <?php the\_author(); ?></p>

<p>Client: <?php echo $work\_client ?></p>

<p>Project Type: <?php echo $work\_type ?></p>

**<?php the\_meta(); ?>**

</header>

There are also many WordPress Plugins in the Official WordPress Plugin Directory that add some nice features to the job of displaying meta tags. A search for Custom Field in the Install Plugins section should help you find even more.

You can use custom fields with you Posts or pages and simply add <?php the\_meta(); ?> into your Theme files where you want this displayed. Remember, in order to add support for custom fields within aour custom post type, we registered it with the following line:

'supports' => array('title','editor','comments','thumbnail','custom-fields'),

In retrospect, I see that we added all of this into the header. I probably would have put this into the footer for each entry.

**Displaying Your Custom Content**

**Creating A Custom Post Type Template**

The WordPress template engine has a hierarchy that it follows when deciding what theme template it uses to display data associated with a post. In the case of our “work” post type, the WordPress hierarchy (as of 3.1) is as follows:

* single-{posttype}.php  
  WordPress will check the theme folder for a file named single-{posttype}.php, if it exists, it will use that file to display the content. In the case of our example where we have a posttype of "work", WordPress will look for single-work.php. For different post types, simple replace {posttype} with the name of your custom post type.
* single.php   
  If no post type specific template is found, the default single.php is used. This is what you probably saw if you did an early preview.
* index.php  
  If no single template is found, WordPress defaults to the old standby – the index.

To get started with our example, we will modify the single.php file that came with the H5 theme and create a file called single-work.php and upload it to your theme folder. The code will display our custom post types, but what about our taxonomies? Let’s add that in. Add the bolded code into single-work.php with the following:

<?php get\_header(); ?>

**<?php $work\_client = get\_the\_term\_list( $post->ID, 'client', '', ', ', '' ); $work\_type = get\_the\_term\_list( $post->ID, 'type', '', ', ', '' ); ?>**

<?php if (have\_posts()) : while (have\_posts()) : the\_post(); ?>

<section>

<article id="post-<?php the\_ID(); ?>">

<header>

<h1><a href="<?php the\_permalink() ?>" rel="bookmark" title="Permanent Link to <?php the\_title\_attribute(); ?>"><?php the\_title(); ?></a></h1>

<p>Posted on <?php the\_time('F jS, Y'); ?> by <?php the\_author(); ?></p>

**<p>Client: <?php echo $work\_client ?></p>**

**<p>Project Type: <?php echo $work\_type ?></p>**

</header>

<section>

<?php the\_content('Read more on "'.the\_title('', '', false).'" &raquo;'); ?>

</section>

<footer>

<?php wp\_link\_pages(array('before' => '<p><strong>Pages:</strong> ', 'after' => '</p>', 'next\_or\_number' => 'number')); ?>

<?php the\_tags( '<p>Tags: ', ', ', '</p>'); ?>

<p>

This entry was posted on <?php the\_time('l, F jS, Y'); ?> at <?php the\_time(); ?> and is filed under <?php the\_category(', ') ?>.

You can follow any responses to this entry through the <?php post\_comments\_feed\_link('RSS 2.0'); ?> feed.

<?php if (('open' == $post-> comment\_status) && ('open' == $post->ping\_status)) {

// both comments and pings open ?>

You can <a href="#respond">leave a response</a>, or <a href="<?php trackback\_url(); ?>" rel="trackback">trackback</a> from your own site.

<?php } elseif (!('open' == $post-> comment\_status) && ('open' == $post->ping\_status)) {

// only pings are open ?>

Responses are currently closed, but you can <a href="<?php trackback\_url(); ?> " rel="trackback">trackback</a> from your own site.

<?php } elseif (('open' == $post-> comment\_status) && !('open' == $post->ping\_status)) {

// comments are open, pings are not ?>

You can skip to the end and leave a response. Pinging is currently not allowed.

<?php } elseif (!('open' == $post-> comment\_status) && !('open' == $post->ping\_status)) {

// neither comments nor pings are open ?>

Both comments and pings are currently closed.

<?php } edit\_post\_link('Edit this entry','','.'); ?>

</p>

</footer>

</article>

<?php comments\_template(); ?>

<nav>

<p><?php previous\_post\_link(); ?> &bull; <?php next\_post\_link(); ?></p>

</nav>

</section>

<?php endwhile; else: ?>

<section>

<article>

<p>Sorry, no posts matched your criteria.</p>

</article>

</section>

<?php endif; ?>

<?php get\_sidebar(); ?>

<?php get\_footer(); ?>

**Archive template**

In the same way that posts are shown on their own single-{posttype].php page, custom post types will use archive-{posttype}.php if it's available.

So for the above example, you could create a archive-work.php file and the product posts would be shown using that template.

Note: Use the is\_post\_type\_archive() function to check if the query shows post type archive page, and the post\_type\_archive\_title() to echo the post type title.

**Querying by post type**

In the rest of the theme system, you can also create new queries to display posts from a specific post type. This is done via the "post\_type" parameter to a WP\_Query.

Example:

$args = array( 'post\_type' => 'work', 'posts\_per\_page' => 10 );

$loop = new WP\_Query( $args );

while ( $loop->have\_posts() ) : $loop->the\_post();

// display content

endwhile;

This simply loops through the latest 10 product posts.

Check out get\_the\_term\_list on the WordPress Codex to learn more about how it works.

**Taxonomy Templates**

Remember how we listed those custom taxonomies in our single page? Well if you click on one where will wordPress take you? As with custom posts, the WordPress template engine has a custom taxonomy template hierarchy that it follows to determine what template it uses to display data associated with a custom taxonomy term. The WordPress hierarchy is as follows:

* taxonomy-{taxonomy}-{term}.php  
  If the taxonomy were client, and taxonomy's slug were drexel WordPress would look for taxonomy-client-drexel.php.
* taxonomy-{taxonomy}.php  
  If the taxonomy were client, WordPress would look for taxonomy-client.php
* taxonomy.php
* archive.php
* index.php

So you should (at minimum) create taxonomy-{taxonomy}.php. You would only need to create taxonomy-{taxonomy}-{term}.php if there was a reason to display a specific taxonomy term in a specific way.

Of course there is a Custom Post Type Plugin that has all these features and more... <http://wordpress.org/extend/plugins/custom-post-type-ui/screenshots/>

**Bonus:**

**Embed Vimeo Videos to WordPress via Custom Fields**

Want to add videos from Vimeo without having to upload them onto your WP server? Well it is quite easy now.

http://en.support.wordpress.com/videos/vimeo/

---

**Homework:**

**Read:**

**Beginning WordPress 3: Make Great Websites the Easy Way**

• Chapter 12 - Custom Content Types, Taxonomies, and Fields

**Build:**

Start working on coding your theme.

Bring your files to class for review.

Be able to show your instructor what progress you have made on your lists:

* Changes you believe you will be able to make in the Dashboard.
* Changes you believe you will be able to make in CSS.
* Changes you believe you will have to make with PHP.

**Submit:**

N/A

**[Instructor NOTE: Check it see if students have made progress, if students has NOT made progress ask them why, offer assistance (during class lab time), and remind him/her that their final project is the Majority of their grade for the class]**