MetaValuables

Unleash the hidden power
of WordPress meta data

Clark Wimberly

Dedicated to my wife.

She rules.

Table of Contents

Foreword

WordPress gives us so many ways to store and access data that it's sometimes hard to keep track of them all. Such was sadly the case with meta data, which I can admit I avoided during my first few years with WordPress. When I discovered the power of meta data and custom fields, I unlocked a whole new world of functionality in my WordPress work. I could save extra information on any post I wanted. Armed with a few core functions, we can save, edit and do amazing things with meta data linked to posts, pages, comments - even users - dynamically, even from our front-end templates.

To get the most out of this course, you should have a test install of WordPress to practice these lessons on. I recommend using a fresh, local install, but you could use just about any old install you've got laying around.

For the live instruction, I'll be using a fresh install of WordPress 3.3.1 running a clean copy of _s (also known as Underscores). _s is an HTML5 starter theme I've recently fallen in love with, but feel free to use Twenty Eleven or Genesis or whatever you're comfortable with.

You'll also need a basic understanding of how WordPress works, including but not limited to: the template hierarchy, using basic functions, being comfortable in /wp-admin, and getting/saving files on your server. If you're not comfortable making a couple of template edits, this course might not be for you.

*"The truth is that you can do pretty much anything you want with **meta data**. Your imagination is your limit."*

—JUSTIN TADLOCK

CHAPTER I

What is meta data?

Meta data is extra information saved to a post, page, or custom post type.

By default, WordPress provides us with fields like the title, the excerpt, the post editor, tags, categories, post author, etc. With custom fields, we can save virtually any type of data, and with practice, call this data with great control in our template files.

The most basic access to meta data is available right out of the box: custom fields. On the Edit Post screen (or page or custom post type, if supported), you'll find a Custom Fields box with a bunch of text input pairs (one for the **meta key**, one for the **meta value**).

If you don't see this Custom Fields box, check the Screen Options tab and make sure Custom Fields is checked.

Meta data is handled with **key/value pairs**. The meta key is the name of the meta data element. You'll use the meta key when calling the data in your template. The meta value is the actual information (be it a date, an image, a mood, etc) we'll be manipulating.

Quick Note:

The meta KEY is the TITLE of the data. The meta VALUE is the data itself. Remember this!

Examples: If you wanted to save a mood on a blog post, you'd enter a key of **"mood"** with a value of **"happy"**. If you wanted to save a capacity for an event, you'd enter a key of **"event_capacity"** with a value of **"150"**.

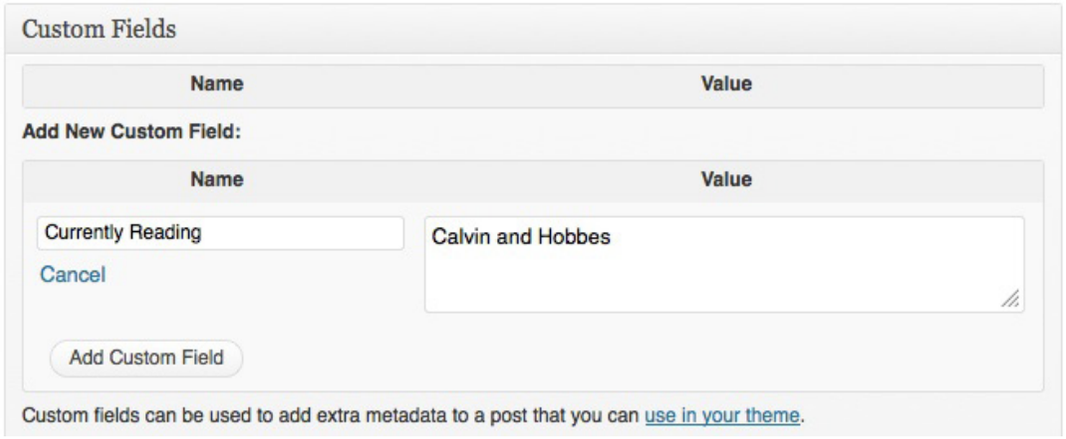
These key/value pairs are the secret to unlocking a great new world of functionality. When people talk about "using WordPress as a CMS" or when you use a fancy magazine or portfolio theme, there's likely a great bit of meta data manipulation going on. Working with meta data comfortably is one of the most important skills a budding WordPress developer will ever learn.

To get your brain flowing in the right direction, here are some examples of meta data that I'd commonly use when building a project:

- An alternate title
- A source link for an article
- Custom markup for editorial content
- Expiration dates
- Stats, specs, and dimensions
- Permission rules

When working with meta data, there are a number of ways to enter the information. Lots of folks use a custom meta box framework, like the one from fellow Austinite Bill Erickson. A custom meta box framework is a great shortcut to a robust set of text inputs, radio buttons, dropdowns and more. If you're building a client site, you'll especially want to consider using some sort of custom field framework.

For most of the examples in this course, however, we'll be just fine using the core custom field functionality. All we'll need is access to our trusty key/value pairs and everything will unfold just fine. Once you're done with the course, though, you'll want to take a peek at a framework or plugin like *More Fields*.



CHAPTER II

Hello World

To test the most basic configuration possible, let's output some simple text into the single post template.

Before we get rolling with our fancier implementations, it's always good practice to start with a basic **"Hello World"** example. If you're unfamiliar, a Hello World test is just a basic method of getting a result to render. In our case, we're going to make **"Hello World"** appear in our single.php template.

First, create a new post or find a test post to edit. On the Post Edit screen, add the meta **key/value**. For the key, let's use **"my_message"** and for the value, **"Hello World"**. Save/publish the post.

In the single post template (usually **single.php**), we're going to call our data using the function **get_post_meta()**. We'll place the function in the loop, so find the area of the template with the other parts of post data, things like **the_title()** and **the_content()**. Our new best friend, **get_post_meta()**, accepts three arguments:

```
<?php echo get_post_meta($post_id, $key, $single); ?>
```

First, you'll need to specify which post we're targeting. If you're in the loop, as is the case with **single.php**, we can use **$post->ID**. Second, we'll need the meta key we're trying to access, which is **'my_message'.** Last, we need to define if this is a single value or not, meaning we need to tell the function if we expect an individual item (such as a title, an event date, a suggested retail price) or a list (such as tracks on an album, members of the board, items in a recipe). Usually, I find myself using the function to return a single result, so I set the last parameter to **true**. To complete our Hello World, we'll fill in the values like so:

```
<?php echo get_post_meta($post->ID, 'my_message', true);
?>
```

Go refresh the post and you should see the text **"Hello world"** in the space where you placed the function. Congratulations! You've just successfully saved and called your first bit of meta data.

CHAPTER III

Doing things with meta data

Now that you understand what meta data is and how it works, we're going to review a few real-world examples that I use in nearly every site I build.

Conditionally display content

Lots of times, in addition to the actual meta data saved, you'll need to conditionally display some markup. For example, you might save an album title **"The Man on the Moon"** as meta data, but want to include some extra markup around the title to display an album icon or other styling.

To get started, create a new post including a meta key of "**album**" with a value of "**The Man on the Moon**" (or whatever floats your boat). In our template, when we call the album title, we'll want to wrap everything in a div, like so:

```
<div class="album">
Today's Jam: The Man on the Moon
</div>
```

The easiest solution is to simply wrap our **get_post_meta()** call we just learned in a <**div**>, but that might not be the best way to do things.

```
<div class="album">
Today's Jam: <?php echo get_post_meta($post->ID, 'album',
true); ?>
</div>
```

The problem with the above code is that any post **not** containing a meta key of "**album**" will still display an empty div, with a caption "**Today's Jam**" for an album that doesn't exist.

The best practice is to only display the markup if the meta data is present. To do that, we'll first attempt to store the value as a variable, then test against it:

```
<?php
$album = get_post_meta($post->ID, 'album', true);
if ($album) { ?>
<div class="album">Today's Jam: <?php echo $album ?></div>
<?php } ?>
```

First, we try to create the variable **$album**. If the meta data exists, the variable will now be set. If the post doesn't have the "**album**" field, the variable won't be created. Next, we test whether **$album** exists with a simple **IF** statement. If **$album** exists, we insert the <**div**>, the caption, and the album title itself (stored as **$album**).

All together, the previous code would render the following, or nothing at all:

```
<div class="album">
Today's Jam: The Man on the Moon
</div>
```

Conditionally displaying content or markup is something **you'll do almost every single time using meta data**. You'll often need extra markup – markup you don't want the post author to have to fuss with, markup you don't want to display if the meta data isn't present.

To master conditionally displaying content, you should get comfortable with PHP's **IF**, **ELSE**, and **IF/ELSE** statements. As you'll soon see, a few simple if statements go a long way in customizing the way your templates display data.

Making a short title

While I usually love a perfectly crafted headline, sometimes I need a shorter title for use in a sidebar or header. Think of it like an excerpt for your title. It's an easy task for a custom field.

If you'd like an example of why you'd need to shorten a title, check out the popular posts in the header of *Android and Me* [**http://androidandme.com**]. More often than not, we need to trim a couple of words from the titles to make everything neat and tidy.

To get started, find or make a post with a long title that you'd like shortened. Within it, create the meta key **"short_title"** and give it the value **'This is my short title!'** (or, of course, whatever you'd like the new title to be).

In the case of *Android and Me*, I've got a special loop that I include in the header with **get_template_part()**. For testing, you can feel free to give this a try in plain ol' **single.php**.

The actual function we're looking for is **the_title()**, which we're going to replace with a conditional check that firsts looks for a short title. If the short title exists, it will be used. If it doesn't, our regular title will be used via **the_title()**.

The code looks like this:

```
<?php
$short = get_post_meta($post->ID, 'short_title', true);

if ($short) {
echo $short;
} else {
the_title();
}
?>
```

If you've been keeping score, you should have no problems figuring out what's going on here. First, we call our meta key "**short_title**" as the variable **$short**. If we find that **$short** exists, we echo it. If not, we use the regular title, as called by **the_title()**.

This is in the same vein as our previous conditional example, except this time we actually went as far as changing what would normally be displayed. Instead of simply not displaying something, we completely changed what was there.

I can't say it enough: **making these checks and taking actions based on what meta data is available is a key part of making a dynamic website.**

Hiding the sidebar

Along with conditionally displaying the content you've saved as meta data, you can use meta data to control when your template shows, or doesn't show, things that would normally be there anyway, like the sidebar.

For example, say you had some really wide code snippet in a specific post or page, and would like to hide the sidebar for just that post.

Normally, the sidebar is called up by the function **get_sidebar()**. For this example, we're

going to seek out that function (usually in **single.php**) and wrap it in a conditional statement that first checks for the existence of a custom meta key.

First, we'll get started by entering a post and saving our custom meta key, which we'll call "**no_sidebar**". For the value, just about anything will work as we're going to simply be testing whether this key exists at all, not for a specific value. When that's the case, I usually use a value of "**true**".

Now, open **single.php** and find where **get_sidebar()** is called, (usually near the end of the file). It should look something like this:

```
<?php get_sidebar(); ?>
```

Of course, without touching it, that function would simply call the sidebar like normal. What we're going to do is wrap the whole thing in an **IF/ELSE** statement, like so:

```
<?php
$nosidebar = get_post_meta($post->ID, 'no_sidebar', true);
if ($nosidebar) {
    // do nothing
} else {
    get_sidebar();
} ?>
```

Like before, you'll see we first try to create the value **$nosidebar**, then we test to see if it exists. If we find that we've told our template we want no sidebar, we do nothing. Otherwise, we load the sidebar like normal.

Another way of doing the same thing, while only using an **IF** statement would look like this:

```
<?php
$nosidebar = get_post_meta($post->ID, 'no_sidebar', true);

if (!$nosidebar) {
    get_sidebar();
} ?>
```

Quick Note:

Using IF/ELSE statements is also a great way to set up default content as a fallback (like a generic post image or user avatar if one hasn't been set)

The difference between the first and second example is that in the first, you'd have the opportunity to place something else where the sidebar would normally be (where it says "**do nothing**", try doing something). The second example only specifies the negative side the coin, which sometimes is what you'll want.

CHAPTER IV

Query posts by meta data

Once you have a bunch of posts (or pages or custom post types) with meta data, you'll likely want to query them based on this newfound power. The good news is that querying by meta data is just as easy as targeting a tag or category.

Query posts by meta key

When you query based on meta data, you can target posts with a certain meta key, or if you need to get more specific, by meta key AND value. For instance, if you'd like to list all posts with a **mood** attached, you'd query for any post with the meta key **"mood"**. If you

wanted to list only posts where you were **happy**, you'd query for posts with the meta key "**mood**" with a meta value of "**happy**".

We'll get started by querying just by **meta key**, sticking with "**mood**" that we used earlier. For this example, we'll look to add a list of **moody** posts to our sidebar, usually in **sidebar.php**. We'll target **any** post with a mood, regardless of happiness.

```
<?php
$moody = new WP_Query('meta_key=mood&posts_per_page=5');

if ($moody) {echo '<ul>';}
while ($moody->have_posts()) : $moody->the_post(); ?>

<li><a href="<?php the_permalink(); ?>"><?php the_title();
?></a></li>

<?php endwhile; ?>
<?php if ($moody) { echo '</ul>';} ?>
```

The above code will create a new **WP_Query** with our special parameters applied, indicating that we only want posts with the meta key "**mood**". Our query will also only return 5 posts max, since we defined **post_per_page**.

After that, we've got a basic loop where you can do what you want. Here, I make an unordered list, then fill it with list items consisting of the post titles wrapped in a link.

Congratulations! You've just queried some posts by meta key, but something tells me you're not going to be satisfied with targeting **only** by key...

Quick Note:

Adding new, custom loops around your template is an absolute MUST when building dynamic themes.

Query posts by meta value

Building on our previous example, let's say you wanted to query for posts that were only made when you were **happy**. That's going to take a query that looks for a key and value pair, or any post with a key of "**mood**" with a value of "**happy**".

The query is very similar to the one we just placed in sidebar.php, with one important difference: we added the **meta_value** parameter to our **WP_Query**:

```
<?php
$moody = new WP_Query('meta_key=mood&meta_
value=happy&posts_per_page=5');

if ($moody) {echo '<ul>';}
while ($moody->have_posts()) : $moody->the_post(); ?>

<li><a href="<?php the_permalink(); ?>"><?php the_title();
?></a></li>

<?php endwhile; ?>
<?php if ($moody) { echo '</ul>';} ?>
```

Now when the query runs, it'll look beyond the fact that the key alone exists, and can make sure to grab only the posts that you're looking for, (in this case **happy** times).

Comparing meta values with a query

When querying based on meta data, you can do more than just see if the data **matches** – you can compare to rules in your query. Is the saved meta data more or less than my specified data?

Let's say you have a custom post type of "**Movie**" and each film has the date released attached as a meta key "**release_year**". Let's also say you're looking for a way to query all films made before **1990**.

If you don't run a movie site, you can pretend this is product prices, event dates or any other time you might need to compare meta data attached to a post. Ignoring the loop and surrounding markup, which you can find in the examples above, our new query would look like this:

```
$query = new WP_Query( array( 'post_type' => 'movie',
'meta_query' => array(
    array(
    'key' => 'release_year',
    'value' => '1990',
    'compare' => '<=',
    'type' => 'numeric' )
    ) ) );
```

We only want posts with the type of "**movie**", only when a release year has been specified and only when that release year is less **than or equal to 1990**.

You'll notice that we've added an extra array for the parameter "**meta_query**". Inside it are arguments like **key**, **value**, **compare**, and **type**. Key and value are pretty self explanatory, and **meta_compare** accepts a handful of comparison arguments, meaning you can use values such as:

- =
- !=
- >
- >=
- <
- <=
- LIKE
- NOT LIKE
- IN
- NOT IN
- BETWEEN
- NOT BETWEEN

The default value is '=', which means when we don't bother to define a compare – the query assumes we want a simple match (which most of the time is just fine). Finally, we've set a **type** of "**numeric**" since we're comparing numbers.

CHAPTER V

Working with user meta

Just like a post or page, you can easily add meta to a user.

For our first example, we'll look at a method of checking whether or not a user has **seen** a special message (like a security warning, site notification, etc). Next up, we'll dive a bit deeper, building a system for letting users add posts to their **favorites**, which we can accomplish with simple user meta.

Display a reminder to a user

When running a site with a lot of users, you'll often run into times when you need to announce something to the whole group. If the notification is important enough, you might even want to display a nag, or a message that won't go away until the user takes action.

For our example, let's say you want to display a message in your header (**header.php**) reminding all users to call their mothers. When displaying the message, we'll want to be courteous and make sure to only display it to users that **have not** already seen and dismissed it. To do so, we'll first check for the user meta key that would indicate that the user has already seen the message.

```
<?php
global $current_user;
get_currentuserinfo();

$seenit = get_user_meta($current_user->ID, 'mother_
reminder', true);

if ($seenit) {
// do nothing
} else {
echo 'Don\'t forget to call your mother!';
} ?>
```

We start by checking the current user data. The function **get_currentuserinfo()** places the current user info into the variable **$current_user**, which we can then use to check for user meta. As it stands now, this message will **always** be shown, since we've not yet given the user a way to hide it. Can you guess how we'll do it?

We'll give the user a way to set the key **"mother_reminder"**, which will cause our **IF** statement to return **true**, and thus we'll know to not show anything to the user. We can do that with a simple link and a bit of logic, as seen on the following page.

```
<?php
global $current_user;
get_currentuserinfo();

if ($current_user->ID AND isset($_GET[dismiss])) {
update_user_meta($current_user->ID, 'mother_reminder',
true);
}


$seenit = get_user_meta($current_user->ID, 'mother_
reminder', true);
if ($seenit) {
// do nothing
} else {
echo 'Don\'t forget to call your mother! <a
href="?dismiss=true">Dismiss</a>';
} ?>
```

We've added a "**Dismiss**" link to the reminder, which will reload the page along with a special parameter, "**?dismiss=true**". Near the top, we add some handling for that parameter. If the current user has an **ID** (aka is logged in) and the URL has our special dismiss parameter, we use **update_user_meta()** to add our "**mother_reminder**" meta key.

Quick Note:

When you check for a user's ID, you also need to check the logged in status (if the user isn't logged in, there won't be an ID available).

Our simple user reminder is now complete. We first check if the user is logged in, then we check if the user dismissed the alert **just now**, then we check if the user has ever dismissed the alert (including **just now**), then we either display or repress the reminder. If we display the reminder, we also display a way to dismiss it.

Of course there are a number of plugins for managing user reminders and notifications, but I thought this was a good example of some basic user meta functionality. I use user meta for saving things like Twitter handles, API returns, point values – the sky's the limit!

Creating a simple Favorites system

Favorites functionality comes in two parts. First, the action where a user actually marks something as a favorite, and second, calling up those favorites and displaying them to a user.

To save a post as a favorite, we'll use something similar to the previous example. In our single template (**single.php**), we'll display a link with a special parameter that saves the post **ID** to a user meta field. First, the link:

```
<a href="?favorite=<?php echo $post->ID ?>">Favorite</a>
```

The link, of course, will reload the page with the post's **ID** added on as a parameter, which we can "**listen**" for in our code, like so, (see code on the following page):

```
<?php
if (isset($_GET['favorite'])) {

global $current_user;
get_currentuserinfo();

if ($current_user->ID){
add_user_meta($current_user->ID, 'favorite', $_
GET['favorite'], false);
}

} ?>
```

When we detect the "**favorite**" parameter in the URL, we'll check to see if the user is logged in, then we'll use **add_user_meta()** to tack on the post **ID** as a meta value with the meta key "**favorite**". We'll also set $unique to "**false**", which means the user will be able to add post after post as favorites, and each will be stored as a new piece of meta data.

Quick Note:

You can use add_user_meta() and update_user_meta() almost interchangeably. If you try to update a meta key that doesn't exist, it will be created.

Once a user has racked up a couple favorites, we'll need to make a way to display them, which we can accomplish with a simple loop, maybe utilized in **author.php** or wherever else you are doing user-centric things.

```
<?php
$favorites = get_user_meta($current_user->ID, 'favorite',
false);

if ($favorites) {
echo '<ul>';

foreach ($favorites as $favorite) {
echo '<li><a href="'.get_permalink($favorite).'">'.get_the_
title($favorite).'</a></li>';
 }

echo '</ul>';
} ?>
```

We start by calling all meta with the key of "**favorite**", which is created as logged in users click the link we made in the previous step. If we have some favorites, we run through them as an unordered list using a **foreach** loop.

As we loop, the value **$favorite** is the post **ID** from each link clicked by the user. We simply make a new list item along with a link and title for each meta key we found when calling **get_user_meta()**.

At this point, the user has no way to remove favorites, but that's outside the scope of this quick example. If you're interested in extending this exercise, take a look at **delete_user_meta()**, it works exactly like you'd expect. You pass it a user **ID**, a meta key, and a value, then **POOF**, the meta data is gone.

Another point to further explore would be a conditional check when you render the "**favorite**" link. You can first check if the favorite exists using **get_user_meta()**, then you can mark the link as if it has already been used (like showing a filled in button or heart).

CHAPTER VI

Go big: Create post meta automatically, then query it

Aside from entering it when editing a post, there are ways to create and update post meta from our templates. The functions **update_post_meta()** *and* **add_post_meta()** *are two ultra-handy tools any dynamic themer is going to want in his or her toolbox.*

We'll look at two examples of automatically setting post meta. First, we'll make a custom shortcode that will automatically update post meta when it runs. Next, we'll make a function that checks the comments on a post, and if they're past a certain threshold, will mark that post as **"hot"**, using, you guessed it, post meta!

Update post meta via shortcode

When you have a blog with authors, one thing you might want to do to make life easier on everyone is create some custom shortcodes. For this example, we're going to use a custom [**quote**] shortcode, which comes with two arguments:

```
[quote text='This is where the quote goes.' name='Clark
Wimberly']
```

When an author uses this shortcode, we want two things to happen. First, of course, we want to display the quote and quote source. Next, and here's where we get tricky, we want to automatically mark any post (with meta data) that uses this shortcode.

Why would we want to do that? For the sake of this example, let's say you want a special section of posts that include quotes. Any time an author includes a quote using the shortcode, we'll automatically be able to query that post. No custom tagging, no categories – just a sneaky way to pick off these posts, automatically.

First up, let's take a look at the plain shortcode, which should be pretty straight forward. You'll place this in functions.php, or wherever else you store your shortcode functions.

```
function sc_blockquote($atts, $content = null) {
   extract(shortcode_atts(array(
        "text" => 'This is a quote',
        "name" => 'Quote Source'
   ), $atts));

return '<blockquote>'.$text.'<cite>'.$name.'</cite></
blockquote>';}

add_shortcode('quote', 'sc_blockquote');
```

First, we name our function, **sc_blockquote**. Next up, we extract the parameters, (along with providing defaults, in case the shortcode author didn't define them). To finish, we return the content, in this case a simple <**blockquote**> tag with an added <**cite**> for the quote source. To make sure this function actually fires, we hook our new shortcode up with **add_shortcode**().

At this point, nothing too tricky has occurred. We've simply created a run of the mill shortcode. Our special sauce will come in the form of post meta data, which we'll need to carefully attach at the time this shortcode runs.

To be smart and save resources, we'll only want to use an **add_post_meta()** call if the meta data **doesn't already exist**. To check, we'll first see if the meta data is available using **get_post_meta()**, like so:

```
function sc_blockquote($atts, $content = null) {
   extract(shortcode_atts(array(
        "text" => 'This is a quote',
        "name" => ''
   ), $atts));

global $post;
$quote = get_post_meta($post->ID, "_quote", true);

if (!$quote) { add_post_meta($post->ID, "_quote", true); }

return '<blockquote>'.$text.'<cite>'.$name.'</cite></
blockquote>';}

add_shortcode('quote', 'sc_blockquote');
```

What we've done here is actually quite clever. Any time this shortcode runs, it'll first check if this post has meta data with a meta key of "**_quote**". If that meta key already exists, our job is done, and we don't bother adding it. If $quote doesn't exist, it means our **get_post_meta()** check failed, and we need to create the meta ourselves.

The effect is all posts using the [**quote**] shortcode now have matching meta data which we can target in a query. On *Android and Me*, we use a similar method any time an author writes about a specific app or game. We've got a [**market**] shortcode that provides download links and stats, and any time it fires I'm able to attach the app name to the post as meta.

That way, when I later need a list of posts where Google Maps is mentioned, I'm able to query posts by meta key "**_market_package**" with a meta value of "**com.google.maps**".

To query based on this newly-created meta data, you'll revisit our handy friend **WP_Query**:

```
$query = new WP_Query( array( 'meta_key' => '_quote') );
```

Quick Note:

If you start a meta key with an underscore, like _quotes or _market, it won't show up on the Edit Post screen, which is sometimes useful for sensitive or complicated data.

This simple query will return only posts with a [**quote**] inside the post body, something that used to be hard to locate. If you're still needing a bit of help constructing a full query and loop, here is the whole process together, (see code on the following page):

```
<?php
$query = new WP_Query( array( 'meta_key' => '_quote') );

if ($query) {echo '<ul>';}
while ($query->have_posts()) : $query->the_post(); ?>

<li><a href="<?php the_permalink(); ?>"><?php the_title();
?></a></li>

<?php endwhile; ?>

<?php if ($query) { echo '</ul>';} ?>
```

It's very much the same process we mastered in our previous chapters, so I hope you're starting to notice a pattern. **Create meta, test meta, take action, REPEAT.** Once you wrap your head around meta data and how it works, the possibilities are endless.

Mark a post "hot" if it has X comments

Another handy thing you can automate with post meta is the selection of hotly discussed or heavily viewed content. For this example, we'll automatically add meta data to any post that has more than 20 comments. Just for illustrative purposes, let's hop into **single.php** and drop the following code inside the loop:

```
$num_comments = get_comments_number();

if ($num_comments >= 20) {

$hotpost = get_post_meta($post->ID, "_hotpost", true);
if (!$hotpost) { add_post_meta($post->ID, "_hotpost", true);
}
}
```

First, we get the number of comments using **get_comments_number()**. If there are more than 20 comments, we'll want to add meta data with the key "**_hotpost**". Again, to be

efficient, we'll only want to fire **add_post_meta()** if we absolutely have to. We check first using **get_post_meta()**, and if we find it doesn't exist, we're then safe to add the meta key. The next time this fires, our meta key is in place and we won't bother updating it.

The result is that any post with more than 20 comments will automatically have meta data attached to it. We can then easily target these posts in a query looking for the meta key "**_hotpost"**, like so:

```
$query = new WP_Query( array( 'meta_key' => '_hotpost') );
```

While WordPress recently added a way to order your query by comment count, I've found some pitfalls in that plan. If you have a bunch of old posts with a ton of comments, like contests, they'll **always** be at the top of the list, something you might not want if you're trying to run a site with a timely appearance. Querying your **"hot"** posts is a great way to get only popular posts but in a much more desirable order (newest first, by default).

Or course, this is just another example of setting meta data then querying against it – the real magic is in where you go from here. You could easily rig this up to save meta data based on pageviews, or post length (for something like **#longreads**), or a number of social shares, (with a service like **SharedCount.com**).

The ability to automatically highlight your own best content will take you great lengths when trying to craft a dynamic template filled with stuff that people actually want to see. While hand-curating every single piece of content is surely nice, having some automated fallbacks can be a godsend when running a busy site.

CHAPTER VII

Wrapping Up

I hope this guide has shown you how powerful meta data can be. Writing a succinct summary of everything it enables would be almost impossible, as the only limits you'll find are your own skill and imagination (which you'll find rapidly shifting as you practice).

Some of my most successful projects have dozens of pieces of meta data on each post, page and user (maybe more). Once you step outside the post box and into the land of meta, you'll find your site doing all sorts of things you never thought possible.

Keep practicing!

Bonus Section: Using a custom meta box library

As I mentioned near the start of this lesson, there are **lots** of ways to input and manage meta data. While you're more than capable of using the meta **key/value** pairs we've been

using so far, lots of folks like to use a custom meta box library, like the one developed by **Bill Erickson, Jared Atchison, and Andrew Norcross.**

Instead of being stuck with plain key/value pairs, using the custom meta box library (or CMB, for short) lets you manage meta data with rich text inputs, drop downs, checkboxes and radio buttons, etc.

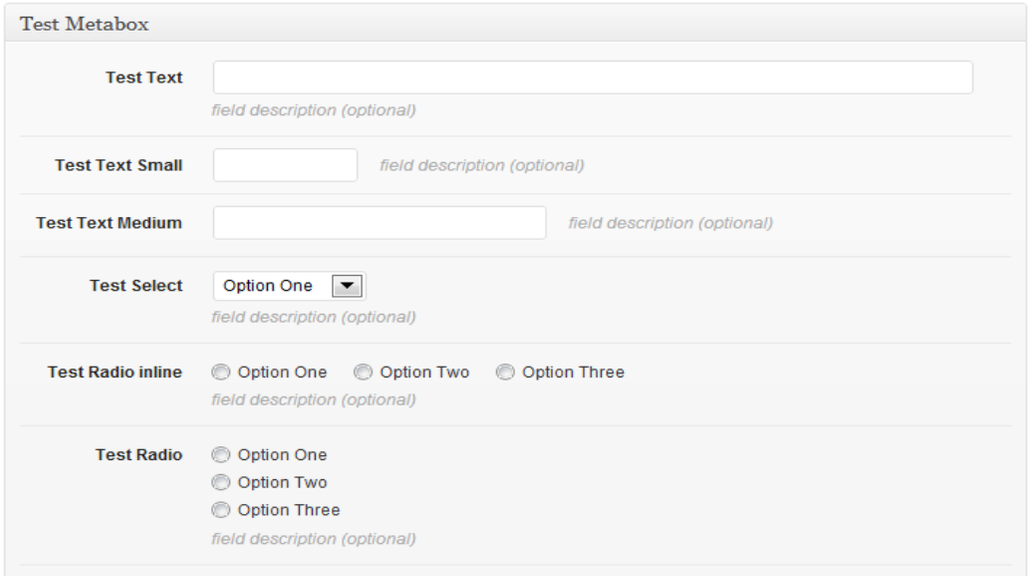
Using the library is simple – just download the files, then include the sample functions file to see your CMB in action. The library is hosted on GitHub, located at **[https://github.com/jaredatch/Custom-Metaboxes-and-Fields-for-WordPress].**

To see a sample implementation, simply include the **example_functions.php** in your **functions.php**, like so:

```
include( get_template_directory() . '/inc/metabox/example_
functions.php');
```

Once you include the file, navigate to the **Edit Page** screen in **/wp-admin**. You'll see a whole slew of new fields, including image uploaders, date pickers and more. These custom fields make handling the input of custom data a snap.

If you're making a site for a client, using some sort of **author-friendly** meta box library is a must. Most authors won't be able to correctly enter key/value pairs, and setting up the fields ahead of time is a great way to manage and restrict the way authors enter data.



CHAPTER VIII

Function Reference

The functions listed below are your new best friends. Get familiar with their parameters and you'll be unstoppable.

get_post_meta($post_id, $key, $single);

Parameters

$post_id
(integer) (required) The ID of the post from which you want the data. Use **$post->ID** to get a post's ID.
Default: None

$key
(string) (required) A string containing the name of the meta value you want.
Default: None

$single
(boolean) (optional) If set to true then the function will return a single result, as a string. If false, or not set, then the function returns an array of the custom fields. This is not intuitive. For example, if you fetch a serialized array with this method you want $single to be true to actually get an unserialized array back. If you pass in false, or leave it out, you will have an array of one, and the value at index 0 will be the serialized string.
Default: false

add_post_meta($post_id, $meta_key, $meta_value, $unique);

Parameters

$post_id
(integer) (required) The ID of the post to which you will add a custom field.
Default: None

$meta_key
(string) (required) The key of the custom field you will add.
Default: None

$meta_value
(mixed) (required) The value of the custom field you will add. An array will be serialized into a string.
Default: None

$unique
(boolean) (optional) Whether or not you want the key to be unique. When set to true, this will ensure that there is not already a custom field attached to the post with $meta_key as its key, and, if such a field already exists, the key will not be added.
Default: false

update_post_meta($post_id, $meta_key, $meta_value, $prev_value);

Parameters

$post_id
(integer) (required) The ID of the post which contains the field you will edit.
Default: None

$meta_key
(string) (required) The key of the custom field you will edit.
Default: None

$meta_value
(mixed) (required) The new value of the custom field. A passed array will be serialized into a string.
Default: None

$prev_value
(mixed) (optional) The old value of the custom field you wish to change. This is to differentiate between several fields with the same key. If omitted, and there are multiple rows for this post and meta key, all meta values will be updated.
Default: Empty

delete_post_meta($post_id, $meta_key, $meta_value);

Parameters

$post_id
(integer) (required) The ID of the post from which you will delete a field.
Default: None

$meta_key
(string) (required) The key of the field you will delete.
Default: None

$meta_value
(mixed) (optional) The value of the field you will delete. This is used to differentiate between several fields with the same key. If left blank, all fields with the given key will be deleted.
Default: Empty

get_user_meta($user_id, $key, $single);

Parameters

$user_id
(integer) (required) The ID of the user whose data should be retrieved.
Default: None

$key
(string) (required) The metakey value to be returned.
'metakey' The meta_key in the wp_usermeta table for the meta_value to be returned.
Default: None

$single
(boolean) (optional) If true return value of meta data field, if false return an array.
Default: false

add_user_meta($user_id, $meta_key, $meta_value, $unique);

Parameters

$user_id
(integer) (required) user ID
Default: None

$meta_key
(string) (required) Metadata name.
Default: None

$meta_value
(mixed) (required) Metadata value.
Default: None

$unique
(boolean) (optional) Whether the same key should not be added.
Default: false

update_user_meta($user_id, $meta_key, $meta_value, $prev_value);

Parameters

$user_id
(integer) (required) User ID.
Default: None

$meta_key
(string) (required) The meta_key in the wp_usermeta table for the meta_value to be updated.
Default: None

$meta_value
(mixed) (required) The new desired value of the meta_key, which must be different from the existing value. Arrays and objects will be automatically serialized. Note that using objects may cause this bug to popup.
Default: None

$prev_value
(mixed) (optional) Previous value to check before removing.
Default: None

delete_user_meta($user_id, $meta_key, $meta_value);

Parameters

$user_id
(integer) (required) user ID
Default: None

$meta_key
(string) (required) Metadata name.
Default: None

$meta_value
(mixed) (optional) Metadata value.
Default: None

add_shortcode($tag, $func);

Parameters

$tag
(string) (required) Shortcode tag to be searched in post content
Default: None

$func
(callable) (required) Hook to run when shortcode is found
Default: None

Index

Clark has been building websites with WordPress for the better part of a decade. He's been in Austin for half that time and recently discovered speaking, which has perfectly merged his love for the two.

CLARK WIMBERLY

Author of this Awesome e-Book