

Taming Content

**FREE
Article!**



php[architect]

ALSO INSIDE

Community Corner:
December 2014

Education Station:
Get Your PHP Sound
in the Cloud with
SoundCloud

Laravel Tips:
Queues

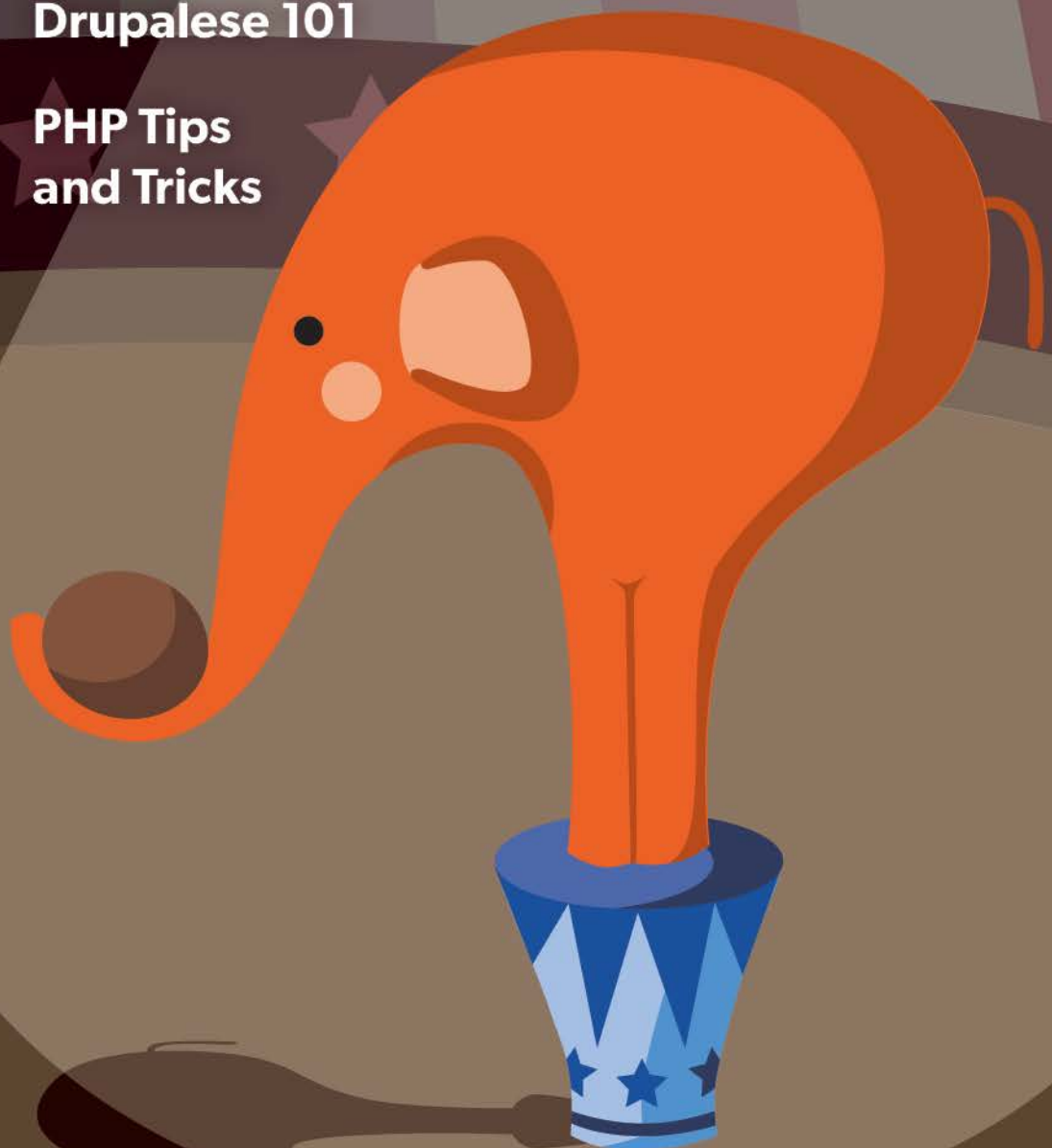
finally{ }:
Another Year
Draws to a Close

**Advanced Sites Deserve Advanced
Custom Fields**

**ProcessWire: Flexibility, Power, and a
Generous Dose of Pure Fun**

Drupalese 101

**PHP Tips
and Tricks**



Advanced Sites Deserve Advanced Custom Fields

Steve Grunwell

Building WordPress meta boxes has gotten easier over time, but it can still be a daunting task for a new theme or plugin developer. Fortunately, Elliot Condon's powerful (and free!) Advanced Custom Fields plugin makes creating complex meta boxes, repeaters, and more a cinch. We'll take a look at the plugin, walk through some practical examples of building carousels, alternate page headlines, and more, and then discuss the future of the plugin with the upcoming ACF 5 release.

DisplayInfo()

Other Software:

- WordPress 3.5+
- Advanced Custom Fields (available in the WordPress.org plugin repository)

Related URLs:

- Advanced Custom Fields - <http://www.advancedcustomfields.com>
- Elliot Condon - <http://www.elliotcondon.com>
- Repeater Field add-on - <http://www.advancedcustomfields.com/add-ons/repeater-field>
- Options Page add-on - <http://www.advancedcustomfields.com/add-ons/options-page>
- Gallery Field - <http://www.advancedcustomfields.com/add-ons/gallery-field>
- Flexible Content Field - <http://www.advancedcustomfields.com/add-ons/flexible-content-field>
- ACF Add-ons - <http://www.advancedcustomfields.com/add-ons>
- Creating a new field type - <http://www.advancedcustomfields.com/resources/creating-a-new-field-type>
- WP add_meta_box - http://codex.wordpress.org/Function_Reference/add_meta_box
- Relevanssi - <http://www.relevanssi.com>

Introduction

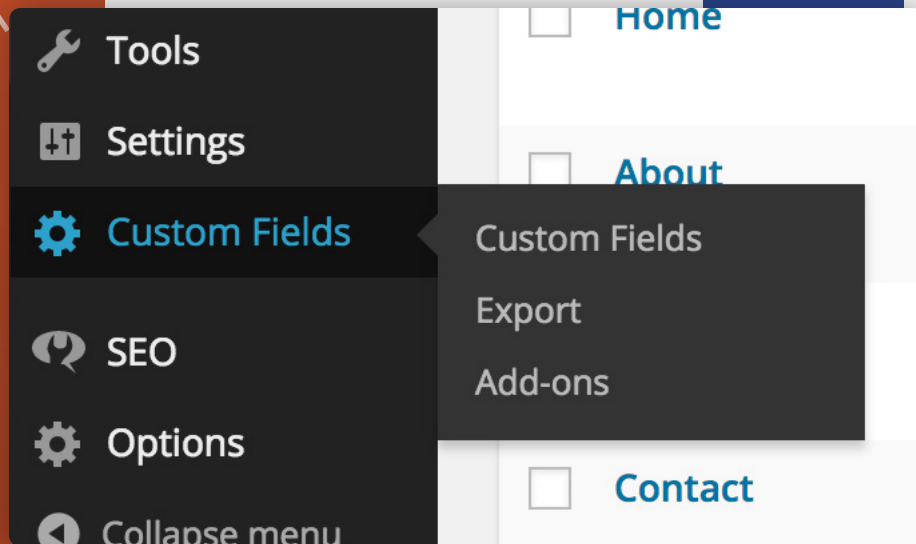
A few years ago, I needed a plugin that would let me design meta boxes to control custom fields on a client theme. After evaluating a few options, I settled on Elliot Condon's Advanced Custom Fields (see Related URLs), and it changed the way I looked at WordPress forever. After working with the plugin extensively over the past several years, I rarely approach any project without using this incredibly powerful tool.

Using Advanced Custom Fields

As is the case with most plugins, the best way to get started is to install it in your WordPress instance. Advanced Custom Fields is installed in the same way as most other WordPress plugins (either uploaded to your plugin directory or installed through the WordPress plugin manager). After activation, a new "Custom Fields" menu becomes available in your WordPress admin menu.

Custom Fields menu, created by
Advanced Custom Fields

FIGURE 1



Building Your Meta Boxes

Before we can do anything with custom fields, we must first create our custom meta boxes. To do this, we create one or more “Field Groups” through the “Custom Fields” menu in the WordPress admin menu. A field group is a collection of one or more custom fields, and represents a single meta box.

Field groups can be assigned to multiple post types, templates, taxonomies, and more. Conversely, it’s possible to limit field groups to specific posts, pages, taxonomies, or pretty much any other type of object you can think of. Custom Field Groups also have page-like menu ordering/prioritization and can be displayed in traditional meta boxes or in “seamless” mode, which forgoes the traditional meta box look and more closely resembles the primary editor for posts and pages.

A number of custom field types are available to you right out of the box with Advanced Custom Fields. Beyond the basics of text fields, `<textarea>`, `<select>` elements, checkboxes, and radio buttons, ACF also gives you access to field types like “Image” (to upload or select a file from your WordPress media library), “Taxonomy” (to select a term within one or more taxonomies), date pickers, and more. With Advanced Custom Fields, you can even create multiple WYSIWYG editors on the same screen, which is incredibly handy for templates that are divided into multiple sections, replacing the need for plugins like Multiple Content Blocks.

Two specific field types, both listed under the “Layout” option group, are worth mentioning: “Message” and “Tab”. A **Message** field enables you to add blocks of HTML to the meta box, perfect for instructions or explanations on client sites. Meanwhile, a **Tab** lets you create tabbed meta boxes, which can clean up your custom field interface tremendously. Together, these two layout field types can make site content far easier for clients to manage.

Display options for an ACF Field Group

FIGURE 2

The screenshot shows the 'Location' and 'Options' tabs for an ACF Field Group. The 'Location' tab is active, showing rules for when the field group should be displayed. The 'Options' tab is also visible, showing settings for the field group's order, position, style, and visibility on the screen.

Location

Rules
Create a set of rules to determine which edit screens will use these advanced custom fields

Show this field group if

Post Type is equal to post and

or

Page is equal to About and

or

Add rule group

Options

Order No.
Field groups are created in order from lowest to highest

0

Position
Normal (after content)

Style
Seamless (no metabox)

Hide on screen
Select items to hide them from the edit screen
If multiple field groups appear on an edit screen, the first field group's options will be used. (the one with the lowest order number)

☐ Hide / Show All

☐ Permalink

☐ Content Editor

☐ Excerpt

☐ Custom Fields

☐ Discussion

A custom field group utilizing both messages and tabs

FIGURE 3

The screenshot shows a custom field group titled 'Credits' with three tabs: 'Cast', 'Crew', and 'Soundtrack'. The 'Cast' tab is active, showing a message and a table for cast members.

Credits

Cast **Crew** **Soundtrack**

The actors and actresses that appear in this film.

Cast members
Connect actor profiles to this film.

	Actor *	Role *
1	- Select -	

Add Cast Member

Displaying Custom Fields

We've discussed adding some custom fields, it's time to display them in our theme. Advanced Custom Fields includes a simple API for interacting with custom fields, the most basic of which is `get_field()`:

```
get_field(
    $field_name,
    [ $post_id = $post->ID, [ $format_value = true ] ]
)
```

This function retrieves a custom field with slug `$field_name` from `$post_id` (defaults to the current post ID). The `$format_value` parameter can be used to disable automatic formatting of the output (for example, running a WYSIWYG field through the typical "the_content" filters).

LISTING 1

ACF also includes `the_field()`, which accepts the same arguments and is actually just a shortcut for `echo get_field()`.

Enhancing the `get_field()` Function

It's generally considered a best practice to wrap calls to functions defined by third-party plugins in a `function_exists()` conditional in your theme, to ensure your site doesn't throw "Call to undefined function" fatal errors if the plugin is ever deactivated. Given how important ACF content can be in our themes, however, I find it much more convenient to add a custom wrapper function to my theme to handle these checks for me. You can see my `theme_get_custom_field()` wrapper function (where "theme_" is a theme-specific prefix to prevent function name collisions with other themes or plugins) in Listing 1.

Using a wrapper function like this guarantees that my theme would continue to function and would even show default values, should I specify them, if Advanced Custom Fields were ever to be disabled. Mimicking `the_field()`, I'll often define a `theme_custom_field()` function as well, which just serves as a shortcut for `echo theme_get_custom_field()`.

```
01. /**
02.  * Get a custom field stored in Advanced Custom Fields
03.  *
04.  * By running it through this function, we ensure that we
05.  * don't get fatal errors if the plugin is uninstalled or
06.  * disabled (thus leaving the function undefined)
07.  *
08.  * @global $post
09.  * @param str $key The key to look for
10.  * @param int $id The post ID
11.  * @param mixed $default What to return if there's nothing
12.  * @return mixed (dependent upon $echo)
13.  *
14.  * @uses get_field()
15.  */
16. function theme_get_custom_field(
17.     $key, $id=null, $default = ''
18. ) {
19.     global $post;
20.
21.     if ( function_exists( 'get_field' ) ) {
22.         if ( ! $id && isset( $post->ID ) ) {
23.             $id = $post->ID;
24.         }
25.
26.         $result = get_field( $key, $id );
27.
28.         if ( $result == '' ) {
29.             $result = $default;
30.         }
31.
32.         // get_field() is undefined, maybe ACF is disabled?
33.     } else {
34.         $result = $default;
35.     }
36.
37.     return $result;
38. }
```

Example: Alternate Page Headlines

A pretty common requirement for marketing sites is the ability to change the `<h1>` on a page without having to change the page object's `post_title` (which then would have to be overridden in menus, URLs, etc.). To get around this, I typically create a custom field for “Alternate Headline” and, if it exists, display that instead of the default page title:

```
<h1 class="post-title"><?php
    $key = 'alternate_headline';

    // Default to the post_title
    theme_custom_field( $key, null, get_the_title() );
?></h1>
```

Wasn't that easy? Thanks to the `$default` parameter for `theme_custom_field()` (or `theme_get_custom_field()`) we're always sure to have something in the H1!

Getting Fields from Non-posts (Taxonomies, Users, etc.)

As I mentioned earlier, Advanced Custom Field groups can be applied to non-post objects, namely taxonomies and users. As taxonomies don't live in the `wp_posts` table (and thus have a different index of IDs), ACF requires that you specify the “ID” of a taxonomy using the pattern `{ $term->taxonomy }_{ $term->term_id }`.

For example, if you were to assign a “category_image” custom field to the post category taxonomy, retrieving the image for category ID #67 would look like this:

```
theme_custom_field( 'category_image', 'category_67' );
```

As of Advanced Custom Fields 4.3.3, you may also pass the term object (returned from `get_term()`) as the ID without having to manually construct the `{ $term->taxonomy }_{ $term->term_id }` key:

```
// On a category page
$term = get_queried_object();
theme_custom_field( 'category_image', $term );
```

Similarly, to retrieve custom fields from a user it's as simple as passing an ID with the pattern `user_{ $user->ID }`. If, for instance, you had a “Hometown” custom field for users and wanted to display it in the loop, you might write something like this:

```
$id = sprintf( 'user_%d', get_the_author_meta( 'ID' ) );
printf(
    __( 'Hometown: %s', 'theme-text-domain' ),
    theme_get_custom_field( 'hometown', $id )
);
```

Add-ons

There are a number of official and third-party add-ons for Advanced Custom fields, which can take the plugin from being “extremely useful” to “holy guacamole, how did I live without these?”. Under the current ACF 4.x version of the plugin, each add-on is installed and activated as a separate plugin. This process was a departure from the 3.x versions (which bundled the add-ons but required users to purchase an activation code) and is slated to change again in the coming months as ACF version 5 is rolled out to the general public

(version 5 is in a public beta at the time of this writing, which we'll be discussing a bit later).

Repeater

The Repeater Field add-on (see Related URLs) was the first add-on I purchased for Advanced Custom Fields and is by far the one I get the most use out of. With the repeater field, building lists of repeatable interfaces is simple: creating carousels, galleries, and other interfaces goes from a delicate game of copy+paste to simply clicking "Add row" and filling out the newly created fields. You can even set minimum and maximum numbers of rows, to help keep your content under control.

Using Repeater Fields in Your Theme

Just like with singular custom fields, there are a couple of functions for retrieving repeater field content: `have_rows()` is like `WP_Query`'s `have_posts()` and is used for looping through the repeater rows. Meanwhile, `the_sub_field()` retrieves the sub-fields within a row.

If you've done any work with the WordPress loop, then the ACF repeater loop will look familiar:

```
if ( have_rows( 'repeater_field_name' ) ) {
    while ( have_rows( 'repeater_field_name' ) ) {
        the_row();
        the_sub_field( 'sub_field_name' );
    }
}
```

Of course, I also like to wrap these functions in my own function (see Listing 2), which ensures that my theme won't break should the repeater add-on ever get disabled. This particular wrapper function also ensures that every sub-field key that I ask for exists (although it can be empty) and, as a bonus, gives me a location in the source to see all the sub-field keys in one place (the third parameter of my `theme_repeater_content()` function call).

To see the repeater field and our wrapper function in action, let's look at some practical applications.

```
01. /**
02.  * Get specified $fields from the repeater with slug $key
03.  *
04.  * @global $post
05.  * @param str $key The custom field slug of the repeater
06.  * @param int $id The post ID (default: $post->ID)
07.  * @param array $fields The sub-fields to retrieve
08.  * @return array
09.  */
10. * @uses theme_get_custom_field()
11. * @uses has_sub_field()
12. * @uses get_sub_field()
13. */
14. function theme_get_repeater_content(
15.     $key, $id = null, $fields = array()
16. ) {
17.     global $post;
18.
19.     if ( ! $id ) $id = $post->ID;
20.     $values = array();
21.
22.     if ( theme_get_custom_field( $key, $id, false )
23.         && function_exists( 'has_sub_field' )
24.         && function_exists( 'get_sub_field' ) ) {
25.
26.         while ( has_sub_field( $key, $id ) ) {
27.             $value = array();
28.             foreach ( $fields as $field ) {
29.                 $value[ $field ] = get_sub_field( $field );
30.             }
31.             if ( ! empty( $value ) ) {
32.                 $values[] = $value;
33.             }
34.         }
35.     }
36.
37.     return $values;
38. }
```

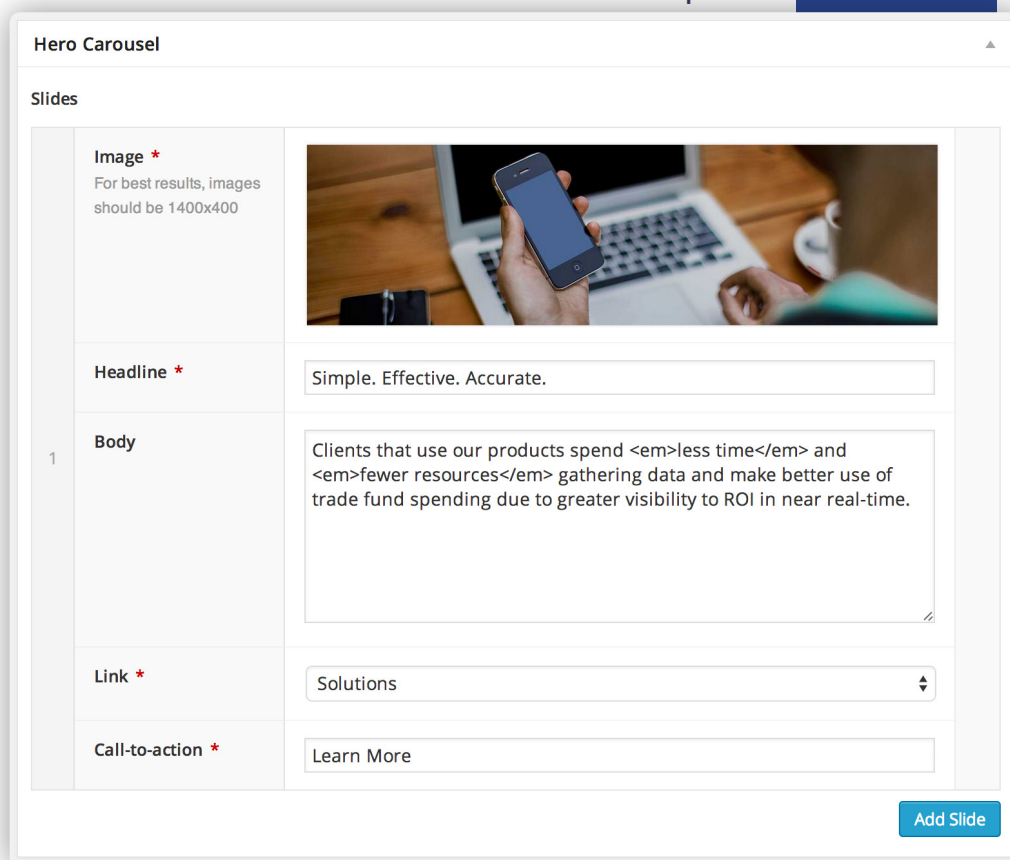
Designing a Carousel

Ah, the hero carousel: the universal pacifier for “every stakeholder needs to be represented at the top of the homepage!” situations and the bane of every designer’s existence. Studies show that these performance-killers generally don’t convert well, but when the client insists we often have to swallow our objections and comply with their demands.




Fortunately, the ACF repeater field is the perfect tool for designing things like carousels without having to resort to messy carousel plugins or dirty hacks (raise your hand if you’ve ever resorted to the “oh, just create a sub-page under the homepage and we’ll load it dynamically” move). Using ACF repeaters, we can quickly build an interface that, for example, includes an image, a title, some body copy, a page link, and the anchor for that link:

A simple interface for clients to manage a hero carousel thanks to ACF repeaters

FIGURE 4



The screenshot shows a WordPress admin interface for a 'Hero Carousel'. It features a table with one slide. The slide contains an image of a hand holding a smartphone, a headline 'Simple. Effective. Accurate.', body text about product benefits, a link 'Solutions', and a call-to-action 'Learn More'. An 'Add Slide' button is at the bottom right.

Hero Carousel											
Slides											
1	<table><tr><td>Image *</td><td></td></tr><tr><td>Headline *</td><td><input type="text" value="Simple. Effective. Accurate."/></td></tr><tr><td>Body</td><td><div>Clients that use our products spend less time and fewer resources gathering data and make better use of trade fund spending due to greater visibility to ROI in near real-time.</div></td></tr><tr><td>Link *</td><td><input type="text" value="Solutions"/></td></tr><tr><td>Call-to-action *</td><td><input type="text" value="Learn More"/></td></tr></table>	Image *		Headline *	<input type="text" value="Simple. Effective. Accurate."/>	Body	<div>Clients that use our products spend less time and fewer resources gathering data and make better use of trade fund spending due to greater visibility to ROI in near real-time.</div>	Link *	<input type="text" value="Solutions"/>	Call-to-action *	<input type="text" value="Learn More"/>
Image *											
Headline *	<input type="text" value="Simple. Effective. Accurate."/>										
Body	<div>Clients that use our products spend less time and fewer resources gathering data and make better use of trade fund spending due to greater visibility to ROI in near real-time.</div>										
Link *	<input type="text" value="Solutions"/>										
Call-to-action *	<input type="text" value="Learn More"/>										

[Add Slide](#)

For this interface, I just created a repeater field with five sub-fields: an image, two text fields (a headline and the call-to-action), a textarea for the body copy, and a page link for the call-to-action’s destination. I chose to make all but the body copy required (if there has to be a hero carousel there should be a reason), and I put a limit of five items in the repeater (to discourage clients from putting a hundred performance-killing slides on the homepage). Once they have more than one slide, content managers can drag the repeater items to re-order them, and it all takes place on the same screen where they’re editing the rest of the homepage content.

Frequently-asked Questions

Another great use for ACF repeaters is to control data formatting that might be difficult to pull off in the main editor. As an example, let’s take a look at a common occurrence: the Frequently Asked Questions page.

Semantically speaking, FAQs are perfect for definition list (<dl>) elements. There's a question (the <dt>) and an answer (<dd>), and a definition list more closely relates the two components than a simple heading/paragraph combination might. Ultimately, we'd want our FAQ markup to look something like this:

```
<dl class="faq">
  <dt>First question?</dt>
  <dd>Answer to first question.</dd>
  <dt>Second question?</dt>
  <dd>Answer to second question.</dd>
</dl>
```

Unfortunately, definition lists are difficult to pull off in TinyMCE (the WYSIWYG editor used by WordPress) without resorting to editing the HTML directly, which can be a deterrent for non-developers. To make this as easy as possible for people managing content on our site, we'll create a new ACF repeater field that simply prompts for a question and an answer.

Since the FAQ will only be on a few pages, it might make sense to create a new page template named "Frequently Asked Questions" for the sake of this example (although you could easily target a specific page without resorting to named WordPress templates). We'll create a new field group named "Frequently Asked Questions", and assign it to our "Frequently Asked Questions" page template.

When creating our repeater field, I'm going to specify the "Question" as a text field, and give editors a WYSIWYG editor for the answers, which will make things like links and images much easier for people entering content. I'll also make both fields required, as a question without an answer makes no sense and an answer without a question is...well, not a frequently asked question. When all is said and done, our FAQ repeater interface will look something like this:

An ACF repeater designed for frequently-asked questions

FIGURE 5

The screenshot displays a WordPress admin interface for a custom field group titled "Frequently-Asked Questions". It features a repeater field with a table structure. The table has two columns: "Question" (marked with a red asterisk) and "Answer" (also marked with a red asterisk). The first row of the table contains the text "First question?" in the Question column and "Answer to the first question." in the Answer column. The Answer column is equipped with a WYSIWYG editor, showing a toolbar with various formatting options like bold, italic, link, and image. Below the table, there is a blue button labeled "Add FAQ".

To generate the definition list in our theme, we'll use the same `theme_get_repeater_content()` function we built earlier. First, we'll start by getting our FAQs:

```
$fields = array( 'question', 'answer' );  
$faqs = theme_get_repeater_content( 'faqs', null, $fields );
```

Then, we'll loop through our `$faqs` array to build our list:

```
<?php if ( $faqs ) : ?>  
  <dl class="faqs">  
    <?php foreach ( $faqs as $faq ) : ?>  
      <dt><?php echo $faq['question']; ?></dt>  
      <dd><?php echo $faq['answer']; ?></dd>  
    <?php endforeach; ?>  
  </dl>  
<?php endif; ?>
```

The end result should be the same definition list markup we planned earlier, but the content is controlled through an intuitive interface that doesn't require content managers to edit any markup.

Options Page

Another extremely useful add-on for Advanced Custom Fields is the Options Page add-on (see Related URLs), which enables you to define one or more option pages within WordPress. These pages are great for things like contact information (I like to split up the address fields into separate custom fields so I can assemble them using Schema.org markup in the footer), default sidebar content, or anything else that you might need to configure for more than one page at a time.

Other Add-ons

There are two other first-party add-ons for ACF: the Gallery Field (see Related URLs), which enables you to easily create image galleries, and the Flexible Content Field (see Related URLs) which breaks your main content into more modular components (useful in certain situations, but probably not necessary for most sites).

In addition to the four official upgrades, there are a number of third-party add-ons for Advanced Custom Fields, adding integrations and/or support for everything from PayPal to Gravity Forms. You can view a complete list of first- and third-party add-ons on the ACF site (see Related URLs).

Extending ACF

If there isn't a field type to accomplish what you need, ACF also features an API for registering your own custom field types within the plugin. The official docs have a fantastic write-up on how to get started with extending ACF (see Related URLs), and there are a number of custom field types on GitHub and in links on the Advanced Custom Fields site that you can use as a jumping-off point.

Moving Between Environments

One of the trickiest things for developers who are getting started with Advanced Custom Fields is managing changes across instances of a WordPress site. Having to manually re-create fields on staging and production is tedious and prone to errors, and having more than one developer working on the site makes the process even more complicated. Fortunately, Advanced Custom Fields enables you to export your field group configurations in two different ways:

1. A XML export that can easily be imported using the WordPress Importer (Tools > Import)
2. A PHP file that can be included in your theme and/or plugin

The XML version is great for developers—it can be imported into a new development environment, the fields can be modified through the “Custom Fields” admin menu, and then it can be re-exported and checked into version control. Meanwhile, the PHP version is great for staging and production, as it doesn’t enable the field groups to be modified, preventing your clients from accidentally breaking the site.

As each export method has distinct advantages, I recommend exporting **both** versions of the custom field configuration and keeping them both under version control. This enables other developers to modify the custom field arrangement in a development environment by importing the XML, then re-exporting both versions. The staging and production copies of the site will remove the custom field groups from the database, instead loading the tamper-proof PHP version. This also guarantees that changes can be rolled out to multiple servers at once, as a fresh deployment would automatically contain the latest ACF configuration.

To pull this off, I export my ACF configuration to a file, usually in `wp-content/themes/{my-theme}/advanced-custom-field-export.php`. Then, in my theme’s main `functions.php` file, I add the following:

```
if ( ! defined( 'USE_LOCAL_ACF_CONFIGURATION' )
    || USE_LOCAL_ACF_CONFIGURATION ) {
    require_once __DIR__ . '/advanced-custom-field-export.php';
}
```

This guarantees that any environment not explicitly ignoring the PHP export will load it. Next, in my development environment, I add the following to my `wp-config.php` file to prevent my PHP export from getting loaded:

```
define('USE_LOCAL_ACF_CONFIGURATION', true);
```

Note: If you’re seeing double fields in your staging and/or production environments, the custom fields are probably being defined both in the database and through the PHP export. Simply delete the duplicate field groups from the Custom Fields menu and let the PHP version run on your site.

Finally, to prevent clients from creating or editing custom field groups on staging or production, you can add the following constant to your `wp-config.php` file to put ACF in “Lite” mode, hiding the “Custom Fields” menu item and interfaces:

```
define('ACF_LITE', true);
```

Caveats

Like most plugins that have a profound impact on how you work with WordPress, there can be drawbacks to Advanced Custom Fields if you're not careful. To help protect your site from becoming totally dependent on ACF, please keep the following points in mind:

Don't Replace/Replicate Default WordPress Functionality

At its core, WordPress content is built around `WP_Post` objects. These can be posts, pages, attachments, or custom post types, but your main content is stored in the `wp_posts` table, which has columns for a title, content, excerpt, and more. These fields can be exported, are heavily integrated with the core WordPress functionality, are often used by other plugins, and will persist long after Advanced Custom Fields is disabled. As custom fields are meant to supplement, not replace, the content of your posts, it's wise to keep as much content as possible in the actual post object.

Similarly, familiarize yourself with custom taxonomies before you go about trying to re-invent the taxonomy system. You'll save yourself hours of work and headaches if you use ACF to enhance—not circumvent—the way that WordPress wants to work.

Separate Theming from Functionality

When you're building complicated systems within WordPress (especially for clients), there's often an expectation that what you build will continue to work even after the client changes themes down the road. In the same way that you might register things like custom post types, taxonomies, and capabilities in a companion plugin to preserve functionality between themes, I would urge you to separate theme-specific custom field groups from those used for more functional purposes. By isolating the components that should be available across themes and adding their exports to a plugin (rather than a theme), you can ensure that these features will live on, regardless of what happens with the current active theme.

Don't Rely on Advanced Custom Fields for Third-party Plugins

Unless you're writing an add-on for Advanced Custom Fields, it's best to not count on (or require) site owners to have ACF installed in order to use your plugin. In those situations, it's best to create meta boxes manually (the WordPress Codex has great instructions on creating these using core APIs, see Related URLs) rather than relying on a third-party plugin.

Avoid Changing Field Keys

Advanced Custom Fields will not automatically update stored custom fields when you change the field's key in the field group, so it's advisable that once you pick a field key (and there's content for that field) that you leave it alone. Labels can be changed at any time, but altering a field key will disassociate the old data, causing you to either a) update the key name manually in the database or b) re-enter the data.

Search Results

Another thing to consider is that the default WordPress search functionality doesn't query against custom fields—if you have major portions of your site content living in ACF, your site search may be less effective than you or your users would like. To remedy this, a third-party WordPress search plugin (I'm partial to Relevanssi, myself—see Related URLs) might be a good route to ensuring that users can still find everything they're looking for on your site.

What's New in 5.0?

It's an exciting time for the Advanced Custom Fields community. Elliot is preparing the release of Advanced Custom Fields version 5 for public release (check the ACF website) after several months in a public beta. Version 5 is a complete rebuild of ACF, designed to be as flexible as possible without compromising performance.

Here is a brief run-down of some of the changes coming in ACF 5.

ACF and ACF PRO

ACF 5 does away with the add-on via plugin model that it has used throughout the version 4 life-cycle, replacing it with a “PRO” version of the plugin that bundles all four add-ons. Pricing starts at \$25 AUD for the single-site “Personal” license, while the unlimited-site “Developer” license will set you back a mere \$100 AUD. According to Elliot, there are no recurring fees, so a developer license will very quickly pay for itself if you work on multiple sites.

New Fields and Locations

ACF 5 introduces a new field type, “oEmbed,” which takes advantage of WordPress’ growing number of supported oEmbed endpoints to embed rich media like audio, video, and tweets. This new field can be placed anywhere other custom fields can be assigned, which now includes comments, widgets, and user forms (login, registration, etc.)!

Automatic, VCS-friendly Exports

One of the most exciting features of ACF 5 is the auto-generated JSON exports. Rather than having to remember to export the XML and PHP files, then load the PHP version in via a PHP include (as I described earlier), simply creating a directory named “acf-json” in your theme will prompt ACF 5 to automatically export JSON representations of your fields, which can be imported into a new environment should you need to change the field configuration later. This will make versioning of your custom fields nearly effortless, which is **huge** for people working in team development environments.

Summary

I’ve only scratched the surface of what can be done with Advanced Custom Fields, but the more people I talk to, the more apparent is the profound impact that this tool has had on the WordPress developer community. I’d urge you to download a copy and take it for a spin: it might just change how you build with WordPress.



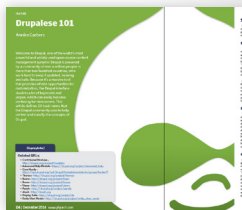
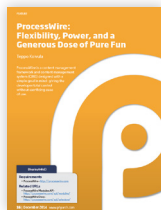
STEVE GRUNWELL is a Senior Web Engineer with 10up, living and working in Columbus, OH. When he’s not writing software, he can be found speaking at conferences, blogging about software development, or continuing his search for the perfect cup of coffee.

Twitter: @stevegrunwell

Want more articles like this one?

Keep your skills current and stay on top of the latest PHP news and best practices by reading each new issue of php[architect], jam-packed with articles.

Learn more every month about frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.



magazine
books
conferences
training
phparch.com

**Get the complete issue
for only \$6!**

We also offer digital and print+digital subscriptions starting at \$49/year.