Building excellent web experiences

with

Advanced Custom Fields and Visual Composer

Stanislav Khromov

WordCamp Norrköping 2014
About me

Stanislav Khromov
Web developer at IDG Sweden
This talk

- Setting up a good workflow for:
  - Custom Post Types
  - Custom Fields
  - Custom Shortcodes
- Providing a good experience for content publishers
- Case study - event sites
Meet the cast

WordPress Core features

- Post Types
- Custom fields
- Shortcodes

[gallery]

[gallery ids="729,732,731"]
Meet the cast

Plugins

● Custom Post Type UI
● Advanced Custom Fields
● MinQueue
● Visual Composer
Custom Post Types - Goals

What do we want?

- Visual interface for creating a CPT
- No configuration in the database!
- Separate CPT configuration into different files
- Autoloading
Custom Post Types - Example

Interface for creating a custom post type in Custom Post Types UI.

Don’t forget the Advanced Options!
### Custom Post Types - Exporting

<table>
<thead>
<tr>
<th>Action</th>
<th>Name</th>
<th>Label</th>
<th>Public</th>
<th>Show UI</th>
<th>Hierarchical</th>
<th>Rewrite</th>
<th>Rewrite Slug</th>
<th>Total Published</th>
<th>Total Drafts</th>
<th>Supports</th>
</tr>
</thead>
</table>

Place the below code in your themes functions.php file to manually create this custom post type. This is a BETA feature. Please report bugs.

```php
add_action('init', 'cotul_register_my_cpt_book');
function cotul_register_my_cpt_book() {
    register_post_type('book', array(
        'label' => 'Books',
        'description' => 'A database of books.',
        'public' => true,
        'show_ui' => true,
        'show_in_menu' => true,
        'capability_type' => 'post',
        'map_meta_cap' => true,
    );
}
```

### CPT UI lets us export to code!

But where do we put this code and how do we organize it?
Custom Post Types - A plugin

Our CPT Plugin

● Each CPT in a separate file
● Plugin handles loading of the CPT:s

● Basic folder structure:
Problem:
Post types will be loaded twice. (Once from code, once from database.)
Custom Post Types - custom-cpt.php

Solution:
Add a config variable to wp-config.php and load conditionally.

```php
<?php
/** The base configurations of the WordPress. ...*/
define('CUSTOM_CPT_DEVELOPMENT_MODE', true);
```
Custom Post Types - custom-cpt.php

Improved Custom CPT plugin:

```php
add_action('plugins_loaded', function() {
    if(defined('CUSTOM_CPT_DEVELOPMENT_MODE') && CUSTOM_CPT_DEVELOPMENT_MODE !== true) {
        foreach (glob(__DIR__ . '/cpt/*.php') as $filename) {
            include $filename;
        }
    }
});
```
Thought process

Create CPT (Visual) -> Configure CPT (Visual) -> Export to PHP file -> Commit to version control
What did we accomplish?

★ Visual interface for creating a CPT
★ No configuration in the database!
★ Separate CPT configuration into different files
★ Autoloading

Result = A good workflow for working with Custom Post Types
Moving on to Custom Fields
Custom Fields

What do we want?

- Visual interface for creating fields
- No configuration in the database!
- Separate field configuration into logical groups
- Autoloading
Custom Fields - Example

Advanced Custom Fields provides an interface for creating custom fields that are grouped into Field Groups - collection of fields that can be attached onto a post type.
Custom Fields - Exporting

ACF lets us export field groups!

Export Field Groups to PHP

Instructions:
1. Copy the PHP code generated
2. Paste into your functions.php file
3. To activate any Add-ons, edit and use the code in the first few lines.

```php
if(function_exists("register_field_group "))
{
    register_field_group(array ( 
        'id' => 'acf_author-fields',
        'title' => 'Author fields',
        'fields' => array ( 
            array ( 
                'key' =>
                'field_535938b0fcee5',
```
Custom Fields - Plugin

You know the drill.

Same basic idea as the CPT plugin!
Thought process

Create field groups and fields (Visual)

Configure field groups and fields (Visual) → Export to PHP file → Commit to version control
What did we accomplish?

- Visual interface for creating fields
- No configuration in the database!
- Separate field configuration into logical groups
- Autoloading

Result = A good workflow for working with fields
Moving on to [shortcodes]
Shortcodes

What do we want?

- Separate shortcodes into portable packages.
- Handle dependencies (CSS, Javascript, Images, PHP libraries)
- Internationalization
- Autoloading
A typical shortcode

- Main shortcode logic
- Templates
- Resources
  - CSS (Regular + Responsive)
  - JavaScript
  - Images
- Dependencies
Custom Shortcodes plugin
A typical shortcode

```php
add_shortcode('show_authors', function($atts)
{
    /** Extract vars */
    extract(shortcode_atts(array('order' => 'ASC'), $atts));

    /* ... */

    ob_start();
    ?>
    <ul class="authors-list">
        <?php if($query->have_posts()) : ?>
            <?php while($query->have_posts()) : ?>
                <?php $query->the_post(); ?>
                <li class="author author-id-<?php get_the_ID(); ?>">
                    <a href="<?php the_permalink(); ?>" title="<?php the_title_attribute(); ?>">
                        <?php the_title(); ?>
                    </a>
                </li>
            <?php endwhile; ?>
        <?php endif; ?>
    </ul>
    <?php enddo_shortcode(ob_get_clean());
});
```
A typical shortcode - CSS

+ media queries in separate file
Custom Shortcodes plugin

● More autoloaders!
Load libraries on `plugins_loaded`
Load shortcodes on `after_setup_theme`
Load CSS/JS on `wp_enqueue_scripts`
Shortcodes

What did we accomplish?

★ Separate shortcodes into portable packages.
★ Handle dependencies (CSS, Javascript, Images, PHP libraries)
★ Internationalization
★ Autoloading
WON'T SOMEBODY PLEASE THINK OF THE PERFORMANCE
Performance

- OpCode cache solves issues with including files
- MinQueue integration lets us concatenate all CSS and JS into one file.

http://wordpress.org/plugins/minqueue/
Let’s talk user experience
Visual Content Builders

- More user friendly than shortcodes
- Enhance content publisher flexibility
- Each shortcode becomes a “block” in the visual editor
Before and after

Authors

[show_authors order="ASC"]

Books

[show_books order="ASC"]
Integrating a shortcode is easy!

From regular shortcode to Visual Composer block in a few rows of code!

Documentation of parameters available at:
Integrating a shortcode is easy!

```php
vc_map(array(
    "name" => "Show books",
    "base" => "show_books",
    "description" => "Shows books",
    "show_settings_on_create" => true,
    "class" => "vc-show-books-block",
    "params" => array(
        array(
            "type" => "dropdown",
            "heading" => 'Order',
            "param_name" => "order",
            "value" => array(
                'Ascending' => 'ASC',
                'Descending' => 'DESC'
            ),
            "std" => "ASC",
            "description" => 'Choose sort order',
            "admin_label" => true
        )
    )
));
```

But where do we put this code?
Our shortcode plugin!

- Since Visual Composer is based around shortcodes, let’s use our Shortcodes plugin for this.
Thought process

- Data that belongs to our custom post type is stored in custom fields on the custom post type.
- Data that controls how blocks are displayed on the frontend is put into the block configuration.
Case Study
IDG Event Sites

- Unify event site creation into a single platform
- Enable full flexibility for content publishers
- Enable easy maintenance
End results

- 3 custom post types
- 12 custom field groups
- 21 shortcodes
- Completely block-based
Demo!

http://computersweden.event.idg.se
Open source!

- Source code for CPT, Custom Fields and Shortcodes loader plugins available on GitHub with examples from this talk!

https://github.com/khromov/wp-custom-cpt
https://github.com/khromov/wp-custom-fields
https://github.com/khromov/wp-custom-shortcodes
Thank you!

Questions?