

**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

Create New Custom Post Type or Taxonomy ·

If you are unfamiliar with the options below only fill out the Post Type Name and Label fields and check which meta boxes to support. The other settings are set to the most common defaults for custom post types. Hover over the question mark for more details.

Post Type Name * ? (e.g. movie)
Max 20 characters, can not contain capital letters or spaces. Reserved post types: post, page, attachment, revision, nav_menu_item.

Label ? (e.g. Movies)

Singular Label ? (e.g. Movie)

Description ?

[Advanced Label Options](#) | [Advanced Options](#)

[Create Custom Post Type](#)

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

Don't forget the Advanced Options!

Custom Post Types - Exporting

Action	Name	Label	Public	Show UI	Hierarchical	Rewrite	Rewrite Slug	Total Published	Total Drafts	Supports
Delete / Edit / Get Code	book	Books	true	true	false	true	book	0	0	title editor excerpt trackbacks custom-fields comments revisions thumbnail author page-attributes post-formats

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](#).

```
add_action('init', 'cptui_register_my_cpt_book');
function cptui_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,
```

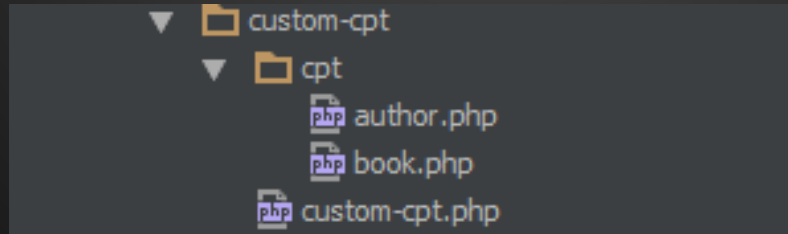
CPT UI lets us export to code!

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:



Custom Post Types - custom-cpt.php

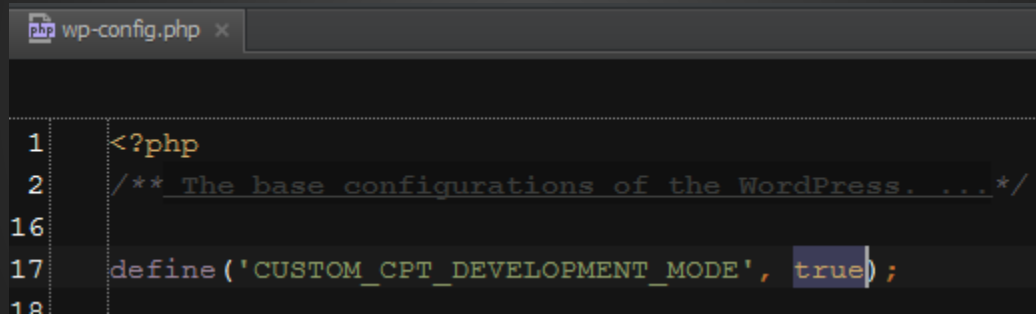
```
1  <?php
2  /*
3  Plugin Name: Custom Content Types
4  Plugin URI:
5  Description: Provides additional content types
6  Version: 2014.04.24
7  Author: khromov
8  Author URI: http://profiles.wordpress.org/khromov/
9  License: GPL2
10 */
11
12 add_action('plugins_loaded', function()
13 {
14     foreach (glob(__DIR__ . "/cpt/*.php") as $filename)
15         include $filename;
16 });
```

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.



```
wp-config.php x
1 <?php
2 /** The base configurations of the WordPress. ... */
16
17 define('CUSTOM_CPT_DEVELOPMENT_MODE', true);
18
```

Custom Post Types - custom-cpt.php

Improved Custom CPT plugin:

```
12  add_action('plugins_loaded', function()
13  {
14      if(defined('CUSTOM_CPT_DEVELOPMENT_MODE') && CUSTOM_CPT_DEVELOPMENT_MODE !== true)
15      {
16          foreach (glob(__DIR__ . "/cpt/*.php") as $filename)
17              include $filename;
18      }
19  });
```

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

Screen Options ▾

Add New Field Group

Author fields

Field Order	Field Label	Field Name	Field Type
1	First name	first_name	Text
2	Last name	last_name	Text

Drag and drop to reorder + Add Field

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 ▼ author ▼ and

or

Add rule group

Publish

Status: Draft [Edit](#)

Move to Trash

Publish

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

Export

Export Field Groups to PHP

Instructions

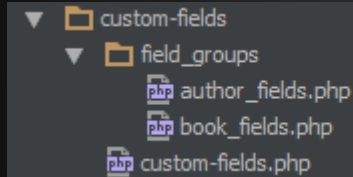
1. Copy the PHP code generated
2. Paste into your functions.php file
3. To activate any Add-ons, edit and use the code in the first few lines.

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

ACF lets us export field groups!

Custom Fields - Plugin

You know the drill.



Same basic idea as the CPT plugin!

```
12 add_action('plugins_loaded', function()
13 {
14     if(defined('CUSTOM_CPT_DEVELOPMENT_MODE') && CUSTOM_CPT_DEVELOPMENT_MODE !== true)
15     {
16         foreach (glob(__DIR__ . "/field_groups/*.php") as $filename)
17             include $filename;
18     }
19 });
```

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

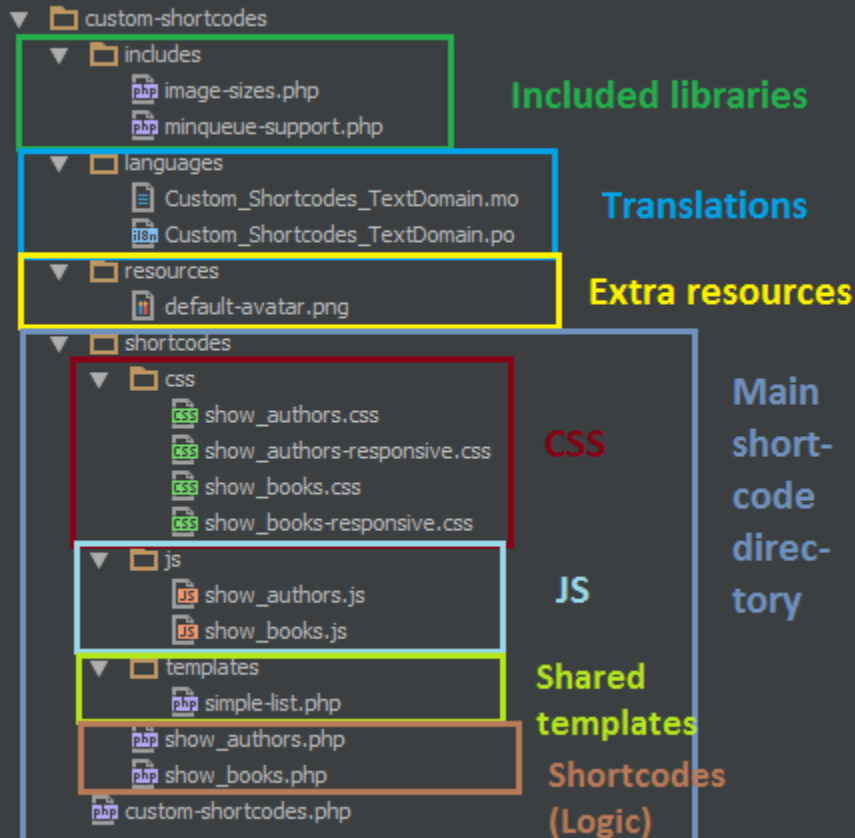
A typical shortcode

- Main shortcode logic
- Templates
- Resources
 - CSS (Regular + Responsive)
 - JavaScript
 - Images
- Dependencies

Custom Shortcodes plugin

- ▼ custom-shortcodes
 - ▼ includes
 - image-sizes.php
 - minqueue-support.php
 - ▼ languages
 - Custom_Shortcodes_TextDomain.mo
 - Custom_Shortcodes_TextDomain.po
 - ▼ resources
 - default-avatar.png
 - ▼ shortcodes
 - ▼ css
 - show_authors.css
 - show_authors-responsive.css
 - show_books.css
 - show_books-responsive.css
 - ▼ js
 - show_authors.js
 - show_books.js
 - ▼ templates
 - simple-list.php
 - show_authors.php
 - show_books.php
 - custom-shortcodes.php

Custom Shortcodes plugin



A typical shortcode

```
5 add_shortcode('show_authors', function($atts)
6 {
7     /** Extract vars **/
8     extract(shortcode_atts(array(
9         'order' => 'ASC',
10    ), $atts));
11
12    /* ... */
13
14    ob_start();
15    ?>
16    <?php if($query->have_posts()) : ?>
17        <ul class="authors-list">
18            <?php while($query->have_posts()) : ?>
19                <?php $query->the_post(); ?>
20                <li class="author author-id-<?=get_the_ID()?>">
21                    <a href="<?php the_permalink(); ?>" title="<?php the_title_attribute(); ?>">
22                        <?php the_title(); ?>
23                    </a>
24                </li>
25            <?php endwhile; ?>
26        </ul>
27    <?php endif;
28
29    /* ... */
30
31    return do_shortcode(ob_get_clean());
32    });
```

A typical shortcode - CSS

```
1 .authors-list .author a
2 {
3   color: green;
4 }
```

+ 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.

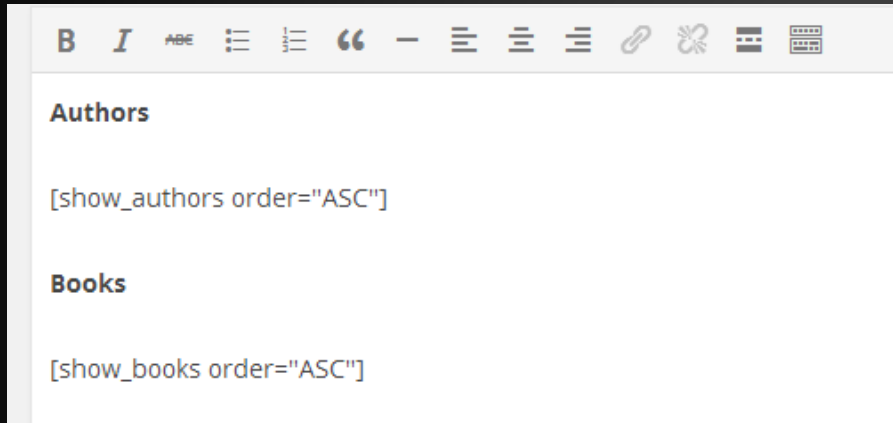
<http://wordpress.org/plugins/minqueue/>

**Let's talk
user experience**

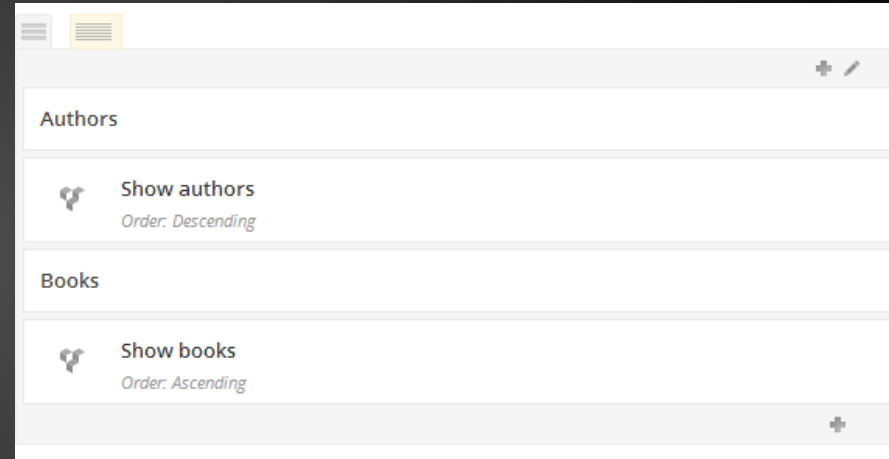
Visual Content Builders

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

Before and after



A screenshot of a rich text editor interface. The top toolbar contains icons for bold (B), italic (I), text color (ABC), bulleted list, numbered list, quote, link, unlink, table, and table border. Below the toolbar, the text area contains two sections: "Authors" with the code snippet `[show_authors order="ASC"]`, and "Books" with the code snippet `[show_books order="ASC"]`.



A screenshot of a user interface showing a list of authors and books. The interface has a light gray header with a menu icon and a plus icon. Below the header, there are two sections: "Authors" and "Books". Each section has a "Show" button with a heart icon and a sorting option: "Order: Descending" for authors and "Order: Ascending" for books. A plus icon is visible at the bottom right of the interface.

Demo

The screenshot displays the WordPress Visual Composer editor interface. On the left is a dark sidebar menu with the following items: Dashboard, Posts, Media, Pages (highlighted in blue), All Pages, Add New, Comments, Authors, Books, Appearance, Plugins, Users, and Tools. The main content area is titled "Visual Composer" and features a top navigation bar with "CLASSIC MODE" and "FRONTEND EDITOR" tabs. Below this, a toolbar includes "Add element", "Add row", "Templates", "Frontend Edit", and "</> CSS" buttons. The editor canvas shows a grid with two rows: the first row contains a "Text" block with the text "Authors", and the second row contains a "Text" block with the text "Books". A "Revisions" section is visible at the bottom of the editor. On the right side, a "Page Attributes" sidebar contains the following settings: Status: Published, Visibility: Public, Revisions: 10, Published on: April, and a "Move to Trash" button. The "Page Attributes" section also includes a "Parent" dropdown menu set to "(no parent)", a "Template" dropdown menu set to "Default Template", and an "Order" field.

[view gif](#)

Integrating a shortcode is easy!

```
2  vc_map(array(  
3      "name" => "Show books",  
4      "base" => "show_books",  
5      "description" => "Shows books",  
6      "show_settings_on_create" => true,  
7      "class" => "vc-show-books-block",  
8      "params" => array(  
9          array(  
10             "type" => "dropdown",  
11             "heading" => 'Order',  
12             "param_name" => "order",  
13             "value" => array(  
14                 'Ascending' => 'ASC',  
15                 'Descending' => 'DESC'  
16             ),  
17             "std" => "ASC",  
18             "description" => 'Choose sort order',  
19             "admin_label" => true  
20         )),  
21      "icon" => "show_books"  
22  ));
```

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

Documentation of parameters available at:
http://kb.wpbakery.com/index.php?title=Vc_map

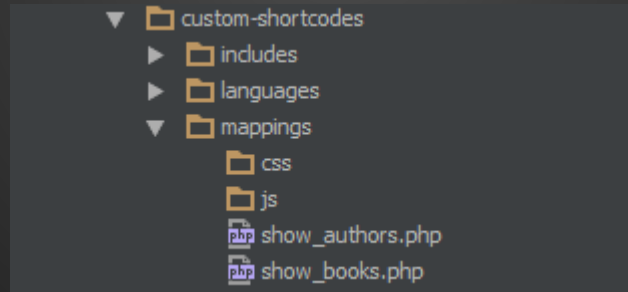
Integrating a shortcode is easy!

```
2  vc_map(array(
3      "name" => "Show books",
4      "base" => "show_books", Shortcode base
5      "description" => "Shows books",
6      "show_settings_on_create" => true,
7      "class" => "vc-show-books-block",
8      "params" => array(
9          array(
10             "type" => "dropdown",
11             "heading" => 'Order',
12             "param_name" => "order", Parameter name
13             "value" => array(
14                 'Ascending' => 'ASC',
15                 'Descending' => 'DESC'
16             ),
17             "std" => "ASC",
18             "description" => 'Choose sort order',
19             "admin_label" => true
20         )),
21      "icon" => "show_books"
22  ));
```

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.

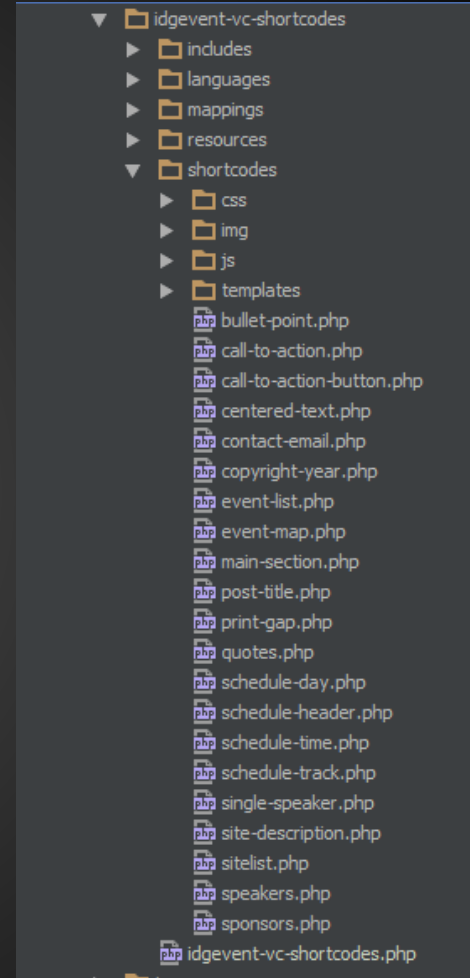
Case Study

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?