Bumblebee Framework

Release

Apr 19, 2018

Contents

Unyson is a framework for WordPress that facilitates development of a theme. This framework was created from the ground up by the team behind ThemeFuse from the desire to empower developers to build outstanding WordPress themes fast and easy. Bumblebee is an extensions package by createIT built on top of Unyson. This documentation is heavily modified by createIT to ensure all custom extensions are well documented.

CHAPTER 1

Getting Started

1.1 Tools

In this chapter, you will learn basic tools needed to work in a project.

1.1.1 Ticketing System

The system to communicate with the team is Active Collab. You will receive an invitation e-mail. The documentation is available at Notes section once you are logged in.

1.1.2 Source Code

You can find the boilerplate source codes at GitLab:

- Theme boilerplate
- Theme Plugin boilerplate
- Theme Demo Plugin boilerplate

1.1.3 Server - Pinky & PHinky

Pinky (or PHinky in Philippines) is a shared local server you can use for development. Every developer has their own server space and server id, nonetheless it is sometimes useful to use them for collaborative work, eg. building a theme demo page.

1.2 Setup

In this chapter, you will find basic tools installation instructions.

1.2.1 Required Info

Please make sure you got from the project manager the following:

- Your e-mail in createIT domain
- Your CT server number and address
- GitLab access
- Active Collab access

1.2.2 Wordpress And Boilerplates Installation

Video Walkthrough

If you like, you can watch a video walkthrough below covering steps of this chapter setup.

There is also another video covering “Project setup” and basic git workflow.

Download Tools

Here are the links to necessary tools. Please start downloading them and in the meantime go ahead and proceed with next instructions.

Everyone:

- PHPStorm as preferred IDE: PHPStorm EAP
- Git for GitLab access: Git
- PHP (for Composer): PHP
- Composer for getting the framework code: Composer

Backend tools:

- Database tool: either Adminer or HeidiSQL
- (optional) Xdebug helper for Chrome: Xdebug helper

Frontend tools:

- Node + NPM: Node
- gulp: gulp

Setup Hosts

This step is optional, however sometimes it helps to keep the connection alive.

1. Open file `C:\Windows\System32\drivers\etc\hosts` in administrator mode
2. Add the following lines at the end (where CT_SERVER_NUMBER is you CT number)

```
10.0.0.111 pinky.createit
10.1.0.101 CT_SERVER_NUMBER.phinky.createit
```

3. The addresses may change, please ask your project manager for the actual addresses.

Setup Pinky (Poland) or PHinky (Philippines)

First, mount your server space as a local drive.

Basically, what you need to do is:

1. Go to *My computer*
2. Click *Map network drive* button
3. Select drive letter (we'll use *V* in the examples)
4. Enter location: `\\pinky.createit\www\CT_SERVER_NUMBER\` or `CT_SERVER_NUMBER.phinky.createit\wwww\CT_SERVER_NUMBER` (replace CT_SERVER_NUMBER with your information, eg. *ct11*)
5. Done. Check if you can browse your Pinky or PHinky.

More detailed instruction can be found here

WordPress Installation

WordPress

1. Download the latest WordPress
2. Unpack the contents to `V:\wp\`
3. You should have the following structure: `V:\wp\wp-config-sample.php`
4. In your browser, open: `http://CT_SERVER_NUMBER.pinky.createit`
5. WordPress Installation will appear
6. Set database name: `wp_CT_SERVER_NUMBER`, eg. `wp_ct11`
7. Set database username: `wp_CT_SERVER_NUMBER`, eg. `wp_ct11`
8. Set database password: `ct`
9. Set database host: `localhost`
10. Set table prefix: `wp_`
11. Click: *Submit*
12. Choose *English* language
13. Set your `CT_SERVER_NUMBER` as a site title
14. Set username: *createit*
15. Set password: *createit*
16. Check: *Confirm use of weak password*
17. Set email: Your CT domain e-mail address
18. Check: *Discourage search engines from indexing this site*
19. Click: *Install WordPress*
20. Click: *Log in* (twice)
21. Now, You should be logged in to WordPress wp-admin panel

Multisite

Multisite will allow you to have multiple themes installations at one place.

1. In notepad open `V:\wp\wp-config.php`
2. Above the line `/* That's all, stop editing! Happy blogging. */` add text: `define( 'WP_ALLOW_MULTISITE', true );`
3. In a browser, open: `http://CT_SERVER_NUMBER.pinky.createit/wp-admin/`
4. Enter menu: *Tools* -> *Network setup*
5. Select: *Sub-directories*
6. Click: *Install*
7. Follow the on screen instructions
8. Now, You should have access to network settings on top of wp-admin panel

Got stuck? Here's a Codex tutorial on how to Create a Network

Install PHP

1. Download the current stable PHP version at PHP Windows
2. Unzip contents to `C:\php\`
3. Rename file `C:\php\php.ini-development` to `C:\php\php.ini`
4. Open `C:\php\php.ini` and uncomment *openssl* extension, ie. change line

   ```
   ;extension=php_openssl.dll
   ```

   to

   ```
   extension=php_openssl.dll
   ```

5. That's all

Install Composer

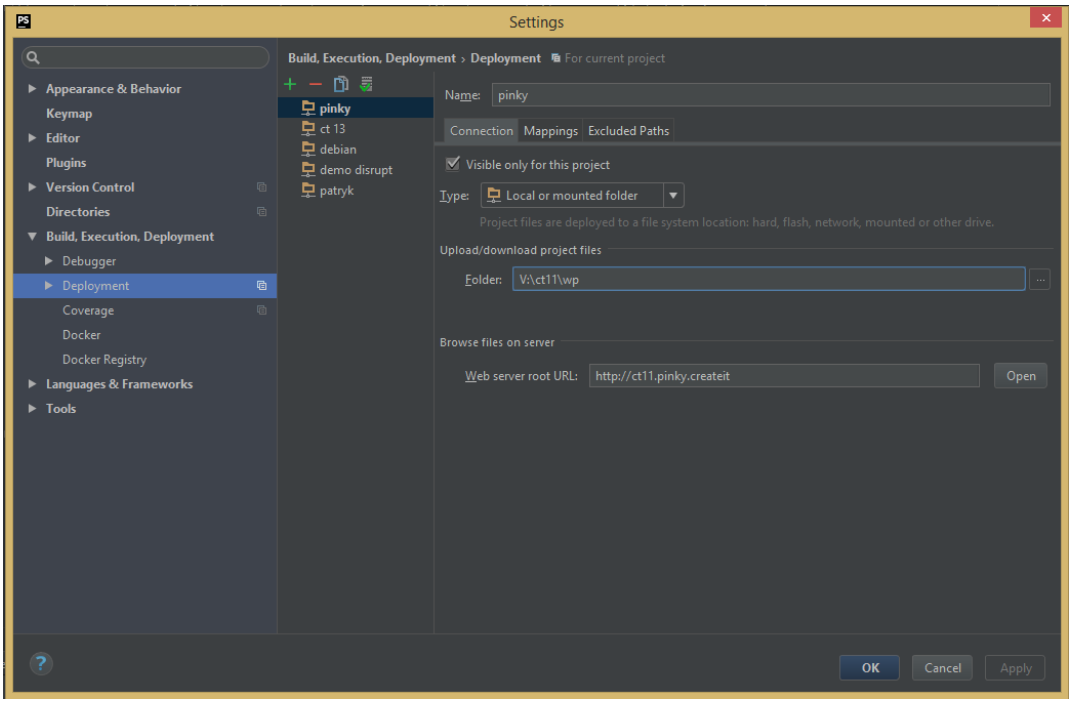
1. Just in case, install Visual C++ Redist
2. Download Composer
3. Run the installer with default options

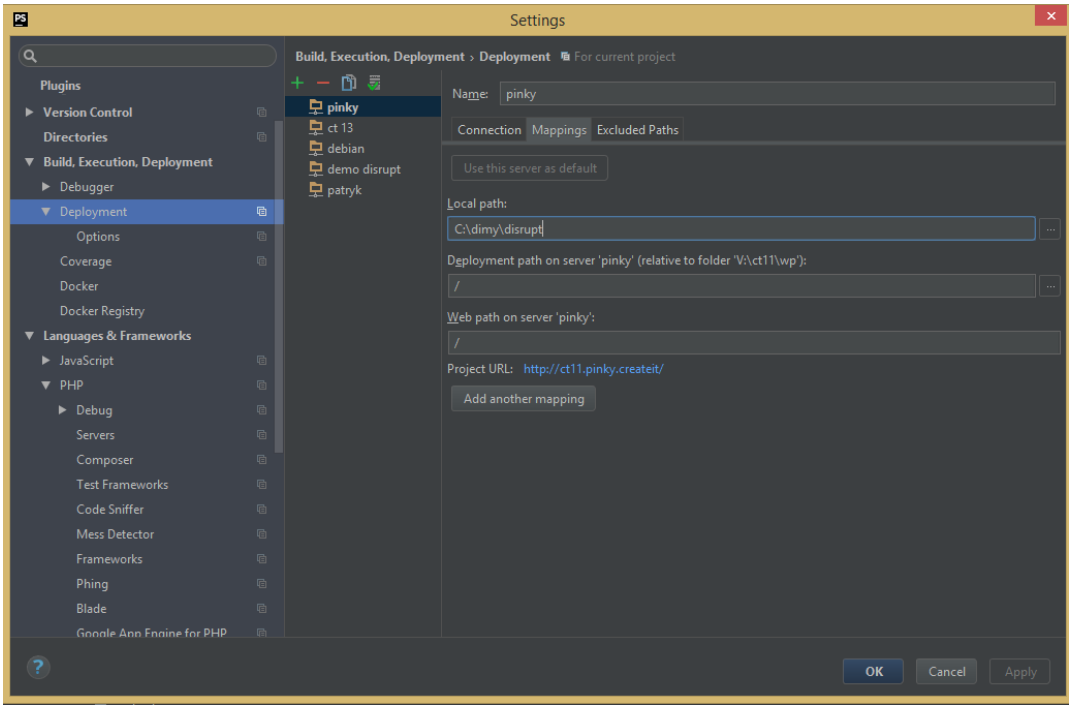
Install Git

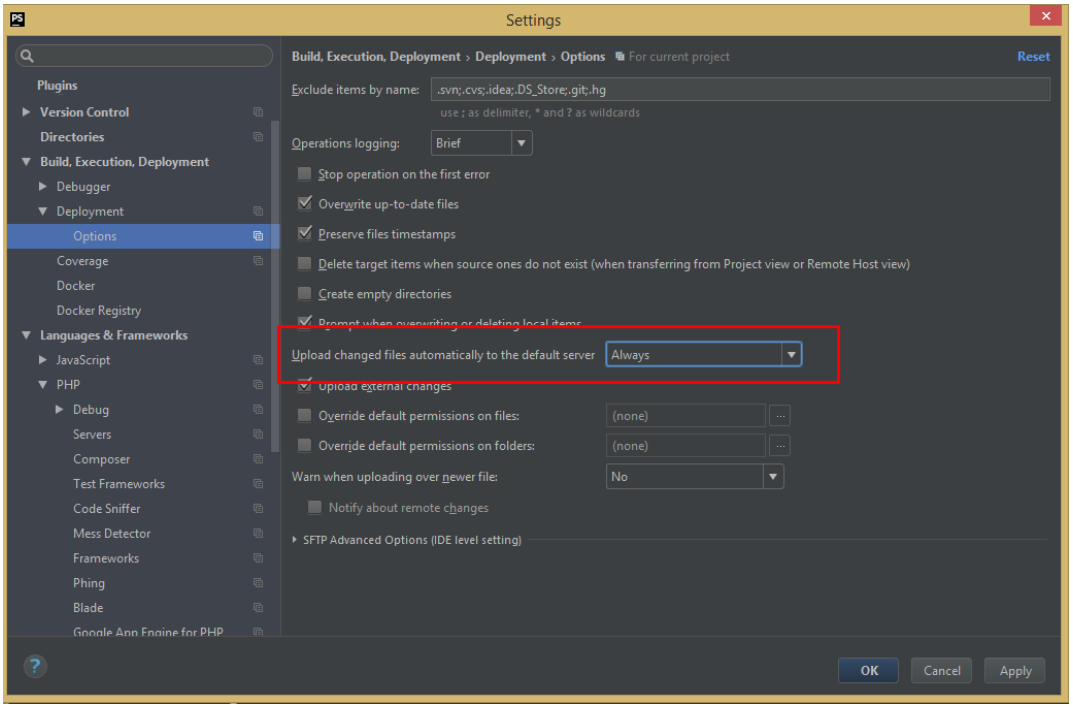
1. Download Git
2. Run the installer with default options

Setup PHPStorm

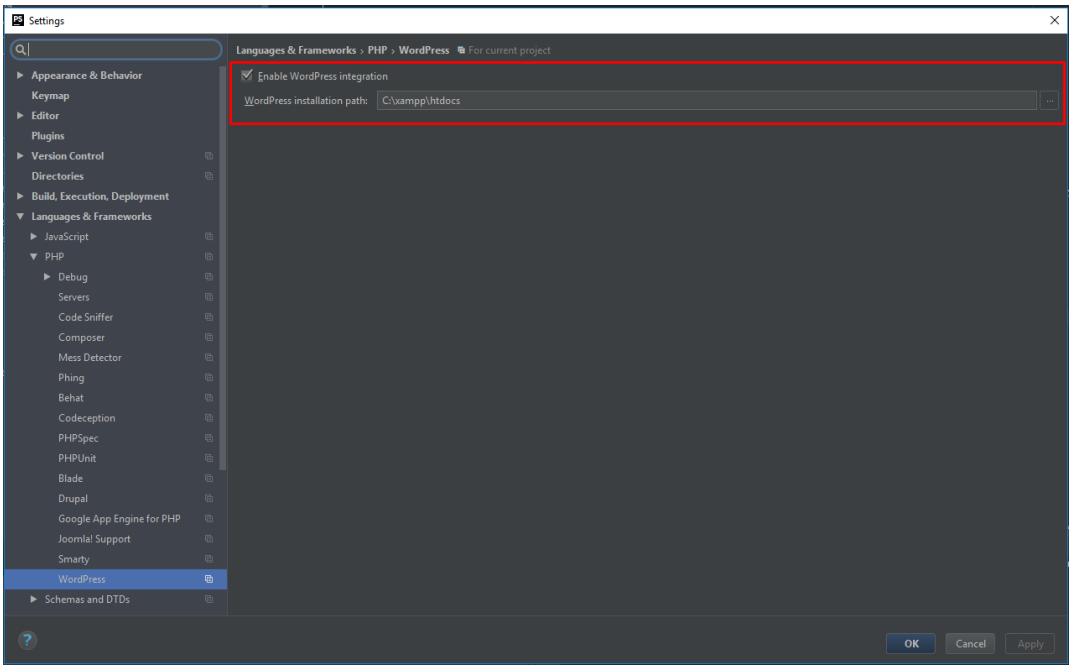
1. Download PHPStorm EAP . It is an EAP, so you will need to download new version and reinstall it every 30 days.
2. Run the installer
3. Check to install: JRE by JetBrains
4. Install
5. Run and open *File -> Settings -> Version Control -> Git* and enter path to installed *git.exe*
6. Open *File -> Settings -> Build & Execution & Deployment -> Deployment* and enter settings as follows changing `ct11` to your CT Server Number

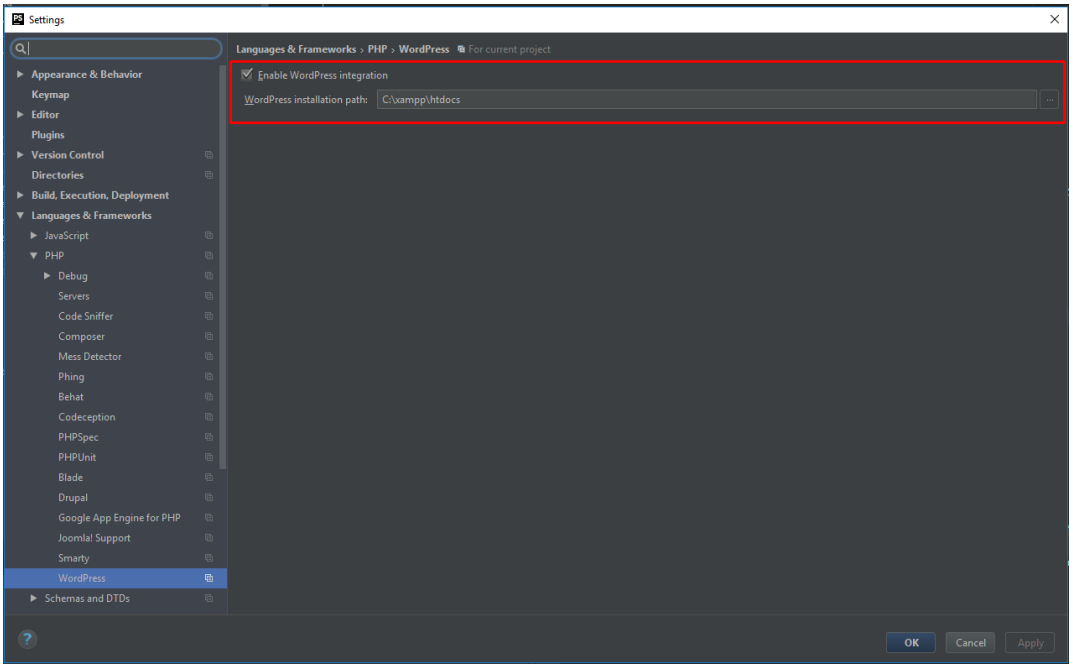

Settings
Build, Execution, Deployment › Deployment For current project
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Directories
Build, Execution, Deployment
Debugger
Deployment
Coverage
Docker
Docker Registry
Languages & Frameworks
Tools
pinky
ct 13
debian
demo disrupt
patryk
Name: pinky
Connection Mappings Excluded Paths
Visible only for this project
Type: Local or mounted folder
Project files are deployed to a file system location: hard, flash, network, mounted or other drive.
Upload/download project files
Folder: V:\ct11\wp
Browse files on server
Web server root URL: http://ct11.pinky.createit
Open
OK
Cancel
Apply


Settings
Build, Execution, Deployment › Deployment For current project
Plugins
Version Control
Directories
Build, Execution, Deployment
Debugger
Deployment
Options
Coverage
Docker
Docker Registry
Languages & Frameworks
JavaScript
PHP
Debug
Servers
Composer
Test Frameworks
Code Sniffer
Mess Detector
Frameworks
Phing
Blade
Google App Engine for PHP
pinky
ct 13
debian
demo disrupt
patryk
Name: pinky
Connection Mappings Excluded Paths
Use this server as default
Local path:
C:\dimy\disrupt
Deployment path on server 'pinky' (relative to folder 'V:\ct11\wp'):
/
Web path on server 'pinky':
/
Project URL: http://ct11.pinky.createit/
Add another mapping
OK
Cancel
Apply



7. Make sure you have set *pinky* as default deployment (in servers list, right click on your *pinky* and check *set as default*)
8. You can add Wordpress support yo your PHPStorm by adding path to folder with fresh Wordpress Installation.



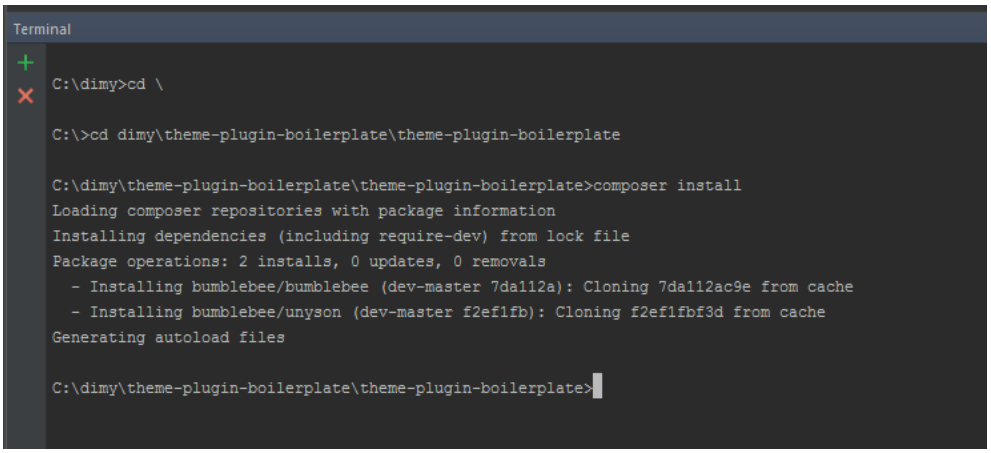


9. Your PHPStorm is configured

Boilerplate Checkout

1. Create a directory `C:\work\boilerplate`
2. Copy contents of `V:\wp` (including hidden files) to `C:\work\boilerplate`
3. Run PHPStorm and click: *Open...*, and select `C:\work\boilerplate`
4. In PHPStorm menu click: *File -> Settings -> Languages & Frameworks -> PHP -> Frameworks* and check: *Enable WordPress integration* and type path to WP files: `C:\work\boilerplate`
5. In PHPStorm menu click: *VCS -> Checkout from version control -> Git*
6. Enter Git repository URL: Theme boilerplate
7. Enter parent directory: `C:\work\boilerplate\wp-content\themes`
8. Enter directory name: `theme-boilerplate`
9. Click: *Clone* and enter credentials
10. After a while you should have `C:\work\boilerplate\wp-content\themes\theme-boilerplate` directory content cloned from Git
11. Repeat checkout with Theme Plugin boilerplate and Theme Demo Plugin boilerplate to `C:\work\boilerplate\wp-content\plugins` directory
12. **For Theme Plugin and Theme Demo Plugin you also need to *composer update* the framework.**
    (a) In PHPStorm, press *Alt+F12* to open the terminal
    (b) Go to Theme Plugin directory (eg. type `cd \`, press *enter*, type `cd work\boilerplate\wp-content\plugins\theme-plugin-boilerplate`, press *enter*)
    (c) Type *composer update* and press *enter*

(d) Here's a screenshot of PHPStorm Terminal:



(e) Unyson and Bumblebee frameworks should now be present in *vendor* directory.

(f) Repeat these steps for Theme Demo Plugin

Deployment

Let's try to send those files to server.

1. In Project view, right click on `C:\work\boilerplate\wp-content\themes\theme-boilerplate` directory and click: *Upload to pinky*
2. Go to file explorer and check if files were sent to the server. If not, check your Deployment settings
3. Go to `http://CT_SERVER_NUMBER.pinky.createit/wp-admin` and select on the top: *My Sites -> Network Admin -> Themes*
4. Theme boilerplate should be visible on the list of themes. Network activate it.
5. Now, you can activate the theme in site wp-admin.

Setup new project

You can wath a video covering this chapter

When you have your project assigned, follow these steps.

1. Create a new folder, eg. `c:\work\projectname` (replace `projectname` with the actual project name)
2. Copy contents of `c:\work\boilerplate` to `c:\work\projectname`
3. Optionally, delete contents of `c:\work\projectname\wp-content\themes` and `c:\work\projectname\wp-content\plugins`
4. In PHPStorm choose `File -> Open` and select `c:\work\projectname`
5. Once opened, checkout your project files from Git (most likely, they will be empty)
   (a) Checkout project theme to `c:\work\projectname\wp-content\themes` (so in result you have `c:\work\projectname\wp-content\themes\projectname\`),
   (b) Checkout project plugin to `c:\work\projectname\wp-content\plugins` (so in result you have `c:\work\projectname\wp-content\plugins\projectname-plugin\`)

(c) Checkout project demo plugin to `c:\work\projectname\wp-content\plugins` (so in result you have `c:\work\projectname\wp-content\plugins\projectname-demo-plugin\`)

6. Once checked out, in file explorer, copy boilerplate files to your project folders

(a) Copy `theme-boilerplate` contents to `c:\work\projectname\wp-content\themes\projectname\them` (create directory `theme`), so in result you have `c:\work\projectname\wp-content\themes\projectname\th` `css`

(b) Copy `theme-plugin-boilerplate` contents to `c:\work\projectname\wp-content\plugins\projectn`

(c) Copy `theme-demo-plugin-boilerplate` contents to `c:\work\projectname\wp-content\plugins\pro`

Commit changes

1. In PHPStorm open `c:\work\projectname`
2. In PHPStorm project tree view, right click on `c:\work\projectname\wp-content\themes\projectname` folder and select `git -> add`
3. In PHPStorm project tree view, right click on `c:\work\projectname\wp-content\plugins\projectname-plugin` folder and select `git -> add`
4. Press `Ctrl+K`, select all files you wish to send to git, then press 'commit & push'. Alternatively, you can right click on a folder and select `git -> push`.

Additional backend settings

wp-config.php

Add below lines to your wp-config.php content (anywhere above wp-settings.php load line):

```
# set this for easy plugin installation
define("FS_METHOD",'direct');

# set error show
define('WP_DEBUG', true);
define('WP_DEBUG_LOG', true);
```

.htaccess (optional)

You may take advantage of xdebugger by setting .htaccess something like this:

Note: This may have a little negative effect on site performance

```
RewriteEngine On
RewriteBase /
RewriteRule ^index\.php$ - [L]

# add a trailing slash to /wp-admin
RewriteRule ^([_0-9a-zA-Z-]+/)?wp-admin$ $1wp-admin/ [R=301,L]

RewriteCond %{REQUEST_FILENAME} -f [OR]
RewriteCond %{REQUEST_FILENAME} -d
```

```
RewriteRule ^ - [L]
RewriteRule ^([_0-9a-zA-Z-]+/)?(wp-(content|admin|includes).*) $2 [L]
RewriteRule ^([_0-9a-zA-Z-]+/)?(.*\.php)$ $2 [L]
RewriteRule . index.php [L]

# general errors
php_flag display_startup_errors on
php_flag display_errors on
php_flag log_errors on
php_value error_log  /var/www/YOUR-PATH/logs/PHP_errors.log
php_value error_reporting -1
php_value short_open_tag false

# xdebug dump
php_value xdebug.var_display_max_depth 20
php_value xdebug.var_display_max_children 256
php_value xdebug.var_display_max_data -1
php_value xdebug.max_nesting_level 5000
php_value xdebug.show_local_vars 1
php_value xdebug.collect_params 1
# php_value xdebug.collect_assignments 1
# php_value xdebug.collect_includes 1
# php_value xdebug.collect_return 1

# xdebug debugger
php_value xdebug.remote_connect_back 1

# xdebug profiler
php_value xdebug.profiler_enable 0
php_value xdebug.profiler_enable_trigger 1
php_value xdebug.profiler_enable_trigger_value PHPSTORM
php_value xdebug.profiler_output_dir /var/www/YOUR-PATH/logs/profiler
php_value xdebug.profiler_output_name callgrind.%H%R.callgrind

# xdebug trace
php_value xdebug.auto_trace 0
php_value xdebug.trace_enable_trigger 1
php_value xdebug.trace_enable_trigger_value PHPSTORM
php_value xdebug.trace_output_dir /var/www/YOUR-PATH/logs/trace
php_value xdebug.trace_output_name trace_%H%R
php_value xdebug.trace_format 0
```

Database access

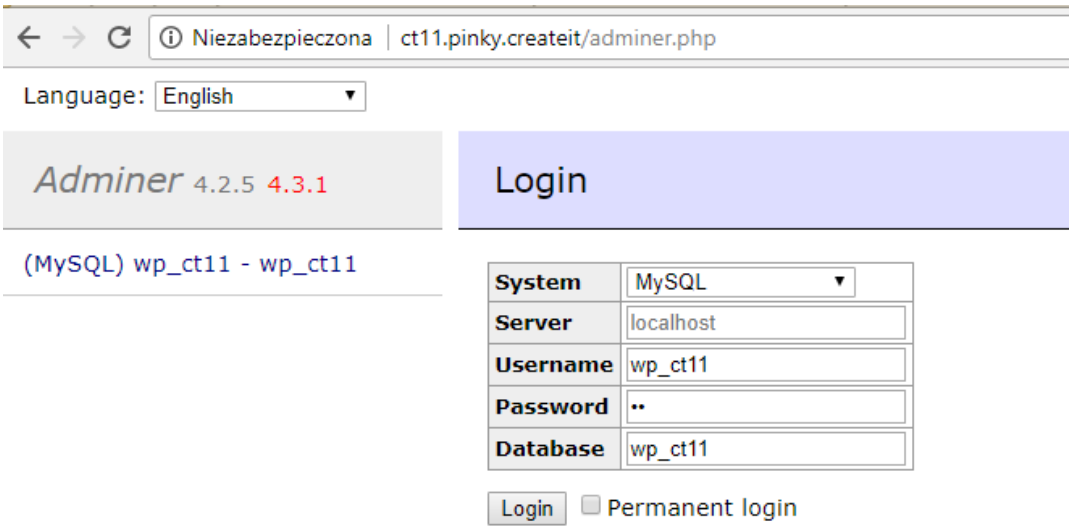
If you need database access, use one of the following tools:

- PHPStorm (menu View -> Tool windows -> Database)
- Adminer or
- HeidiSQL

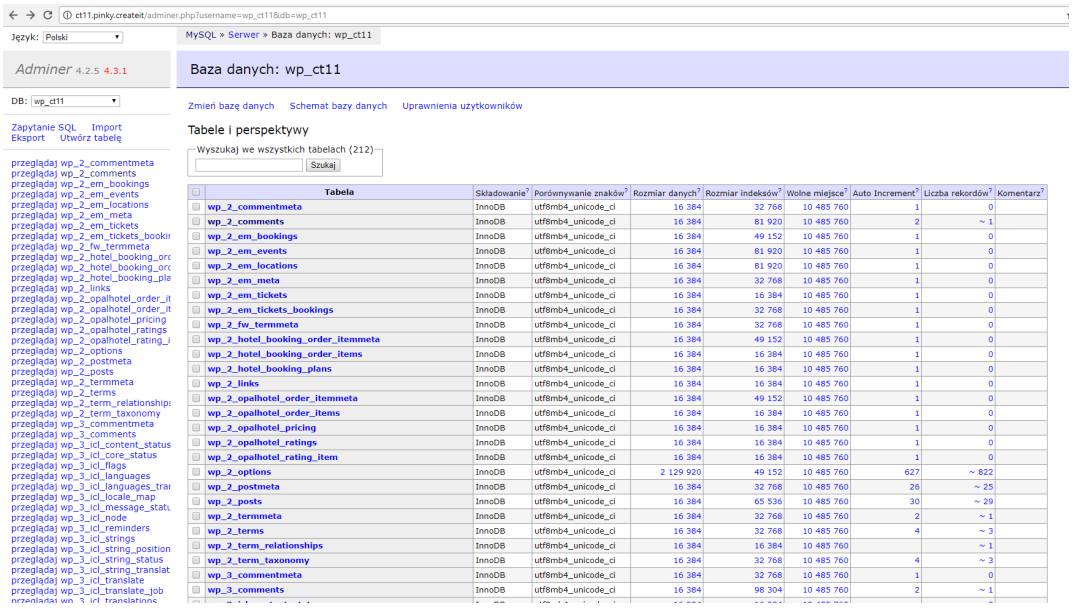
Here is an Adminer basic setup. Adminer is a simple PHP tool.

1. Download Adminer
2. Place `adminer-#####-mysql.php` file in your server wordpress directory, eg. `V:\ct11\wp\adminer.php`

3. Open the file in browser, eg. `http://ct11.pinky.createit/adminer.php`
4. Enter db credentials (replace `ct11` with your pinky number)



5. You should see all tables of your database:



6. That's it.

Tip: If you need to batch replace contents of your db, use SRDB2 PHP tool.

1.3 Project Structure

In this chapter you will learn the basic project structure, how it interconnects, links to repositories.

1.3.1 Theme Plugin

Every theme requires to have one dedicated plugin created. This allows customers who are using our theme to change theme and still have 'some' functionality available. Although it won't look too pretty, all custom post types, basic shortcode structure and hooks will still be available.

Note: Basic structure can be downloaded here: Theme Plugin boilerplate

---

It is recommended to put most functionality in the plugin as it has higher chances of passing the review. In particular put there:

- custom post types settings
- shortcodes
- widgets

Note: When developing functionality which can be used in other projects, place it in the plugin (preferably as an extension) to allow easy copying from project to project.

Structure

```
plugins/
|--themename-plugin/
   |--theme-name.php              # load bumblebee
   |--vendor/
      |--bumblebee/
         |--bumblebee/            # bumblebee core
         |--unyson/               # unyson core
   |--extensions/
      |--example-extension-name/
         |--includes/             # additional extension files
         |--views/                # view files
            |--view.php
         |--class-fw-extension-example-extension-name.php   # ( optional )␣
↪shortcode class to handle extension logic
         |--manifest.php          # Extension details: title, description,␣
↪version, dependencies, etc.
         |--hooks.php             # Extension specific hooks ( add_filter, add_
↪action )
         |--helpers.php           # Extension functions accessible globally
         |--static.php            # Add script and styles
         |--posts.php             # Register post types, taxonomies etc.
         |--shortcodes/
            |--example-shortcode-name/
               |--includes/        # additional shortcode files
               |--class-fw-shortcode-example-shortcode-name.php   # ( optional␣
↪) shortcode class to handle shortcode logic
               |--config.php       # simple options: title, tab, icon,␣
↪description, custom markup
               |--options.php      # shortcode options array
               |--views/
               |--view.php
```

```
    |    |--...
    |--...
```

Note: Theme plugin slug should be `themename-plugin`, ie. My Theme plugin slug would be `mytheme-plugin`, see Coding Standards for more explanation

Plugin description

Sample project data do fill in the theme plugin would be (for an example My Theme theme):

```
/**
 * Plugin Name: My Plugin
 * Plugin URI: http://createit.pl/
 * Version: 1.0
 * Description: Plugin containing essential functionality required by My Theme.
 * Author: createIT
 * Author URI: http://createit.pl
 */
```

Vendor directory

When you clone Theme Plugin from repository, there will be `composer.json` and `composer.lock` files there. In order to get vendor files, you will need to have `composer` installed and run in the main plugin dir:

```
composer install
```

ie.:

```
C:\dimy\disrupt\wp-content\plugins\disrupt-plugin>composer install
```

This will pull both Bumblebee and Unyson from repositories. Unyson is the framework core with basic functionality. Bumblebee adds additional features to the core.

1.3.2 Theme

Theme files include all WordPress view templates, theme related functions, static assets, theme specific framework customizations including demo content settings, and plugin dependencies.

Note: Basic structure can be downloaded here: Theme boilerplate

Structure

```
themes/
├-projectname/
  ├-theme/
  |  |--vendor/                    # Commerial plugins zip files (if required by the␣
↪theme)
  |  |--page-templates/
```

```
  |  |--languages/                  # Leave empty for autogenerated .pot file (create␣
↪a .gitkeep file if needed)
  |  |--demo-content/               # Demo contents manifests and screenshots. The␣
↪content itself should be autogenerated.
  |  |  |--example-multipager/
  |  |  |  |--manifest.php          # Demo content name, image, preview link
  |  |  |  |--screenshot.png
  |  |  |--example-onepager/
  |  |  |--...
  |  |--assets/                     # Static assets
  |  |  |--css/
  |  |  |  |--style.css
  |  |  |--sass/
  |  |  |  |--style.scss
  |  |  |  |--..
  |  |  |--javascripts/
  |  |  |--images/
  |  |  |--.../
  |  |--inc/
  |  |  |--includes/
  |  |  |  |--bee-plugins-check/    # Library to check if required plugins are active␣
↪and render install instructions
  |  |  |  |--mocks/                # Functions mocking required plugins (such as␣
↪Unyson) when not active
  |  |  |  |--tgmpa/                # Plugin dependencies library & settings
  |  |  |  |--sub-includes.php      # Includes files in this directory
  |  |  |--vendor/                  # Autoincluded settings for external vendors
  |  |  |  |--megamenu.php
  |  |  |  |--...
  |  |  |--helpers.php              # Theme functions
  |  |  |--hooks.php                # Theme related hooks
  |  |  |--init.php                 # Includes everything else
  |  |  |--menus.php                # Menus related settings
  |  |  |--posts.php                # Default post types & taxonomies options
  |  |  |--static.php               # Enqueue scripts and styles
  |  |--framework-customizations/
  |  |  |--extensions/
  |  |  |  |--extension-name/
  |  |  |  |--...
  |  |  |--theme/
  |  |  |  |--manifest.php          # Theme id, supported extensions etc.
  |  |  |  |--config.php            # Theme specific configuration
  |  |  |  |--options/
  |  |  |  |  |--settings.php       # Theme settings panel
  |  |  |  |  |--{settings_tab}.php # Theme settings example tab
  |  |  |  |  |--customizer.php     # Customizer options
  |  |  |  |  |--...
  |  |--child-theme/
  |  |  |--framework-customizations/
  |  |  |  |--... # same as in then parent theme, but here you can overwrite specific␣
↪files from the parent theme
```

Structure overview

theme/languages/ In this directory, there should be .pot translation files for the theme. These are autogenerated in the package build process. See Build for more information.

theme/demo-content In this directory, there should be demo contents for each theme flavor or version (ie. onepager, multipager). They are autogenerated in the package build process. See Build for more information.

theme/vendor If theme requires any plugins which are not free to download from WordPress.org repository, they should be available in the theme vendor directory as zip files so that once theme is activated customer can quickly install them via WP Admin. Note this is relevant to external plugins only (ex. Visual Composer or Revolution Slider). The Theme Plugin will be pulled and packed automatically in the package build process. See Build for more information.

theme/inc This directory contains main theme functionality which is available to users **without** the Theme Plugin being active. When developing a theme, put all the basic functionality in here.

Note: Theme slug should be `themename`, ie. My Theme slug would be `mytheme`, see Coding Standards for more explanation

---

Here are sample information to be put in the main theme stylesheet

```
/*
Theme Name: Mytheme
Theme URI: https://createit.pl
Author: Create IT
Author URI: Create IT
Description: My Theme
Version: 1.0
License: GNU General Public License v2 or later
License URI: http://www.gnu.org/licenses/gpl-2.0.html
Text Domain: My
Domain Path: /languages
*/
```

1.3.3 Demo Plugin

Demo plugin is a set of dev tools which either help during development or change the way theme is displayed on the demo page (ie. production server), as well as provides automatic demo content export functionality.

Common use cases:

- Test shortcodes and VC templates
- Use theme hooks to alter demo website looks, ie. layout urls
- Use Unyson Backups hooks to auto generate theme demo contents
- Provide WP_CLI functions to recompile SCSS. Read more: Build

---

Note: Basic structure can be downloaded here: Theme Demo Plugin boilerplate . After cloning, run *composer update* to get plugin core.

Structure

```
plugins/
|--theme-name-demo-plugin/
   |--class-fw-demo-plugin.php      # Demo functionality hooks
   |--class-fw-demo-content.php     # Demo content import/export related
 ↪functionality
```

```
  |--class-fw-settings-form.php   # Demo plugin settings form settings in wp-
↪admin
  |--wp-cli-functions.php         # Functions accessible to WP CLI
  |--...
  |--extensions
    |--ct-demo
      |--class-fw-extension-ct-demo.php
      |--hooks.php
      |--manifest.php
      |--shortcodes
        |--ct_shortcode_markup
          |--config.php
          |--options.php
          |--views
            |--view.php
```

1.4 Basics You Should Know Before Your First Project

The following documentation structure was made assuming you already have some WordPress experience and therefore focuses on the framework use. In other cases, we encourage you to head to WordPress Basics first.

1.4.1 Coding Standards

Here are some starting rules to keep in mind:

- Although WordPress recommends php 7, the code should work on **php 5.3**. Don't use php 5.4+ features (such as short array syntax, array dereference on call), because some hosting providers don't have php 5.4+ installed on the servers.
- Follow WordPress Coding Standards.

> **Note:** If you already have some code written with spaces indentation (that does not follow WordPress Coding Standards), use this RegExp to replace spaces with tabs:
>
> `(?<=^\s*) {4}` replace with `\t`

Project names

- The Project name should be one word and not contain any special characters such as -, _ etc. Examples of project names are: `disrupt`, `estato`.
- The Theme name should be `projectnametheme` and not contain any special characters such as -, _ etc. Examples of theme names and theme folder names are `disrupttheme`, `estatotheme`.
- The Theme Plugin folder should be `projectname-plugin`, examples: `disrupt-plugin`, `estato-plugin`. Plugin class name should be `FW_CT_Bee_Projectname_Plugin`, examples: `FW_CT_Bee_Disrupt_Plugin`, `FW_CT_Bee_Estato_Plugin`.
- The Theme Demo Plugin folder should be `projectname-demo-plugin`, examples: `disrupt-demo-plugin`, `estato-demo-plugin`. Demo Plugin class name should be `FW_CT_Bee_Projectname_Demo_Plugin`, examples: `FW_CT_Bee_Disrupt_Demo_Plugin`, `FW_CT_Bee_Estato_Demo_Plugin`.

Text domains

In every theme, the text domain to use is `ct_theme` which at later point is automatically changed to theme name.

```
$label = __( 'This text is translatable', 'ct_theme' );
```

Important: Do not use `ct_theme` as a part of variable name or function name or anywhere besides text-domain as it will be replaced during the build.

Prefixes

In both theme and plugin everything is prefixed to prevent naming conflicts and to give a meaning to functions, classes and methods names. See the guidelines below.

Important: Prefix everything! Lack of proper prefixes is a frequent theme reject reason. Read more: Reject Reasons

Theme

In theme development, a unique and more than 2 letters prefix is required by ThemeForest reviewers, therefore the prefix to use in a theme is `fw_ct_bee_` for variables, functions, classes, and `fw-ct-bee-` for styles, image sizes, db options etc.

- Functions and classes should be prefixed with:
  - `fw_ct_bee_` for functions
  - `FW_Ct_Bee` for classes

```
function fw_ct_bee_head() {
    // ...
}

class FW_Ct_Bee_Pagination
{
    // ...
}
```

- Private functions and classes should be prefixed the same way due to ThemeForest standards
- Functions used for hooks should be prefixed with:
  - `fw_ct_bee_action_` for `add_action()`
  - `fw_ct_bee_filter_` for `add_filter()`

```
/**
 * @internal
 */
function fw_ct_bee_filter_the_content($content) {
    // ...

    return $content;
}
add_filter('the_content', 'fw_ct_bee_filter_the_content');
```

```
/**
 * @internal
 */
function fw_ct_bee_action_init() {
    // ...
}
add_action('init', 'fw_ct_bee_action_init');
```

- Filters and actions should be prefixed with `fw_ct_bee_`.

```
$data = apply_filters('fw_ct_bee_whatever', $data);

do_action('fw_ct_bee_whatever');
```

Extensions

- Public functions and classes should be prefixed with:
  - `fw_ext_<extension-name>_` for functions
  - `FW_Ext_<extension-name>_` for classes
- Private functions and classes should be prefixed the same way due to ThemeForest standards
- Functions used for hooks should be prefixed with:
  - `fw_ext_<extension-name>_action_` for `add_action()`
  - `fw_ext_<extension-name>_filter_` for `add_filter()`

  For e.g. if extension name is `demo`:

```
/**
 * @internal
 */
function fw_ext_demo_filter_the_content($content) {
    // ...

    return $content;
}
add_filter('the_content', 'fw_ext_demo_filter_the_content');

/**
 * @internal
 */
function fw_ext_demo_action_init() {
    // ...
}
add_action('init', 'fw_ext_demo_action_init');
```

- Filters and actions should be prefixed with `'fw_ext_<extension-name>_'`.

  For e.g. if extension name is `demo`:

```
$data = apply_filters('fw_ext_demo_whatever', $data);

do_action('fw_ext_demo_whatever');
```

Unyson Core

- Public functions and classes should be prefixed with:
  - `fw_` for functions
  - `FW_` for classes

```
function fw_useful_function() {
    // ...
}

class FW_Useful_Class {
    // ...
}
```

Note: A **Public function** is meant to be used by anyone. Usually it's a helper function that does something useful.

- Private functions and classes should be prefixed with:
  - `_fw_` for functions
  - `_FW_` for classes

```
/**
 * @internal
 */
function _fw_private_function() {
    // ...
}

/**
 * @internal
 */
class _FW_Private_Class
{
    // ...
}
```

Note: A **private function** is used somewhere internally. Don't forget to use the @internal tag in your PhpDoc in order to make it clear to other developers that this is a private function. It will also remove the function from your documentation (if you are using an automatic documentation generator)

- Functions and methods used for hooks should be prefixed with:
  - `_action_` for `add_action()`
  - `_filter_` for `add_filter()`

```
/**
 * @internal
 */
function _action_init_something() {
    // ...
```

```
}
add_action('init', '_action_init_something');
```

Important: Be sure the function name is unique enough in order to minimize the chances to be defined by someone else. Do not use too simple function names like `_action_init`.

```
class FW_Example
{
    public function __construct()
    {
        add_filter('the_content', array($this, '_filter_the_content'));
    }

    /**
     * @internal
     */
    public function _filter_the_content($content) {
        // ...

        return $content;
    }
}
```

- Filters and actions should be prefixed with `'fw_'`.

```
$data = apply_filters('fw_whatever', $data);

do_action('fw_whatever');
```

1.4.2 Visual Composer

Visual Composer is a page builder plugin for WordPress. Please take a look at introductory videos or skip directly to the docs below.

Introduction

Videos

Documentation

Visual Composer documentation

Download The Latest Version

Start using Visual Composer now - you can download it here: Visual Composer download

Visual Composer & Unyson

While Visual Composer has its own way of defining shortcode parameters (see vc_map), thanks to Bumblebee package we can take leverage of Unyson elegant approach to shortcode options and define our shortcodes Unyson way (read more about Unyson options).

Note: This integration is possible thanks to Bumblebee Visual Composer extension which needs to be enabled in Unyson settings in wp-admin.

1.4.3 Create Your First VC Shortcode

Sample Shortcode

Here is a simple shortcode created Unyson way which will be automatically mapped to and available in Visual Composer. This shortcode displays an icon.

Desired shortcode structure

Create the following file structure in your theme plugin (see Plugin structure). Please name the shortcode folder using `ct-` prefix. Here, it's `ct-icon`.

```
plugins/
|--theme-plugin/
  |--extensions/
    |--shortcodes/
        |--shortcodes/
          |--ct-icon/
            |----config.php       # shortcode config
            |----options.php      # shortcode options
            |----views
                 |----view.php    # shortcode view
```

Shortcode options: options.php

```
<?php if ( ! defined( 'FW' ) ) {
    die( 'Forbidden' );
}

$options = array(

    'icon' => array(
            'label' => __( 'Icon:', 'ct_theme' ),
            'value' => 'fa-users',  // default value
            'type'  => 'icon',      // unyson option type 'icon' (will translate to
↪VC 'iconpicker' param type)
            'tab'   => __( 'Icon', 'ct_theme' ),
    ),

    'title' => array(
            'label'   => __( 'Icon title:', 'ct_theme' ),
            'type'    => 'text',    // unyson option type 'text' (will trasnalte to
↪VC 'textfield' param type)
```

```
            'tab'      => __( 'Icon', 'ct_theme' ),
    ),

    'description' => array(
            'label'    => __( 'Icon description:', 'ct_theme' ),
            'type'     => 'text',    // unyson option type 'text' (will trasnalte to␣
↪VC 'textfield' param type)
            'tab'      => __( 'Icon', 'ct_theme' ),
    ),

);
```

Shortcode view: views/view.php

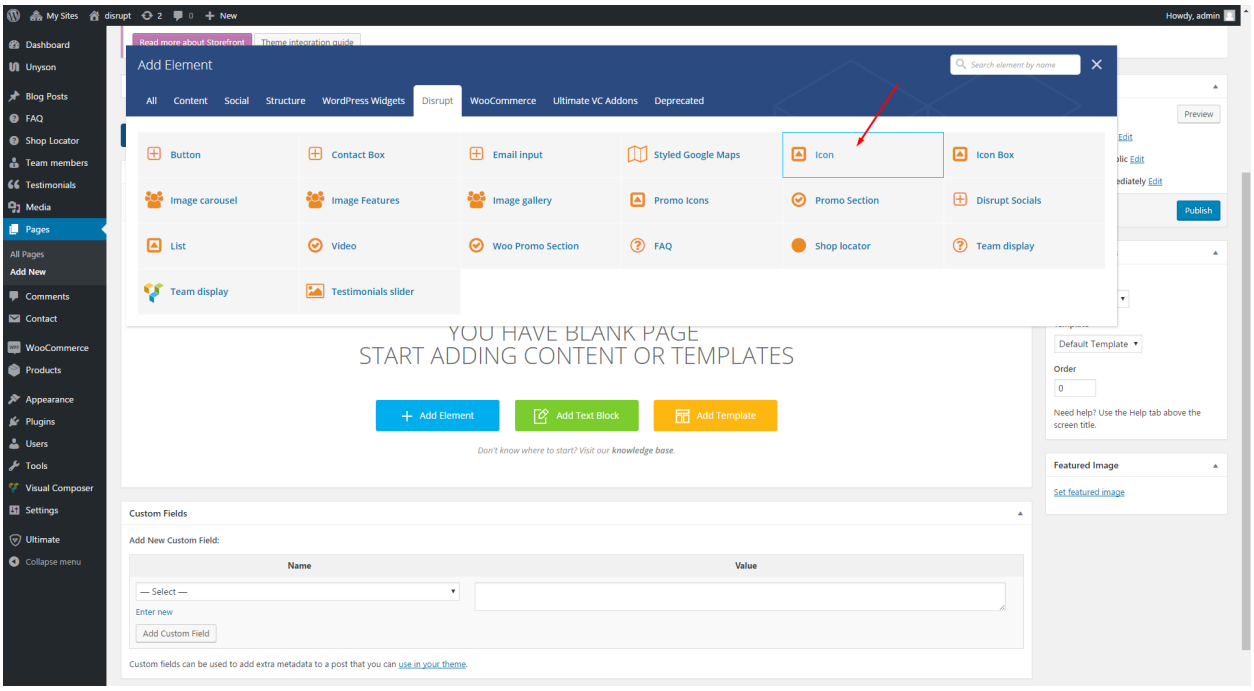
```
<?php if ( ! defined( 'FW' ) ) {
    die( 'Forbidden' );
}

/** @var $atts array The shortcode attributes */

?>

<div class="ct-iconBox ct-iconBox--type1 ct-iconBox--white">
    <div class="ct-iconBox-icon"><i class="<?php echo esc_attr( $atts['icon'] ) ?>"></
↪i></div>
    <h6 class="ct-iconBox-title"><?php echo esc_html( $atts['title'] ) ?></h6>
    <p class="ct-iconBox-description"><?php echo esc_html( $atts['description'] ) ?></
↪p>
</div>
```

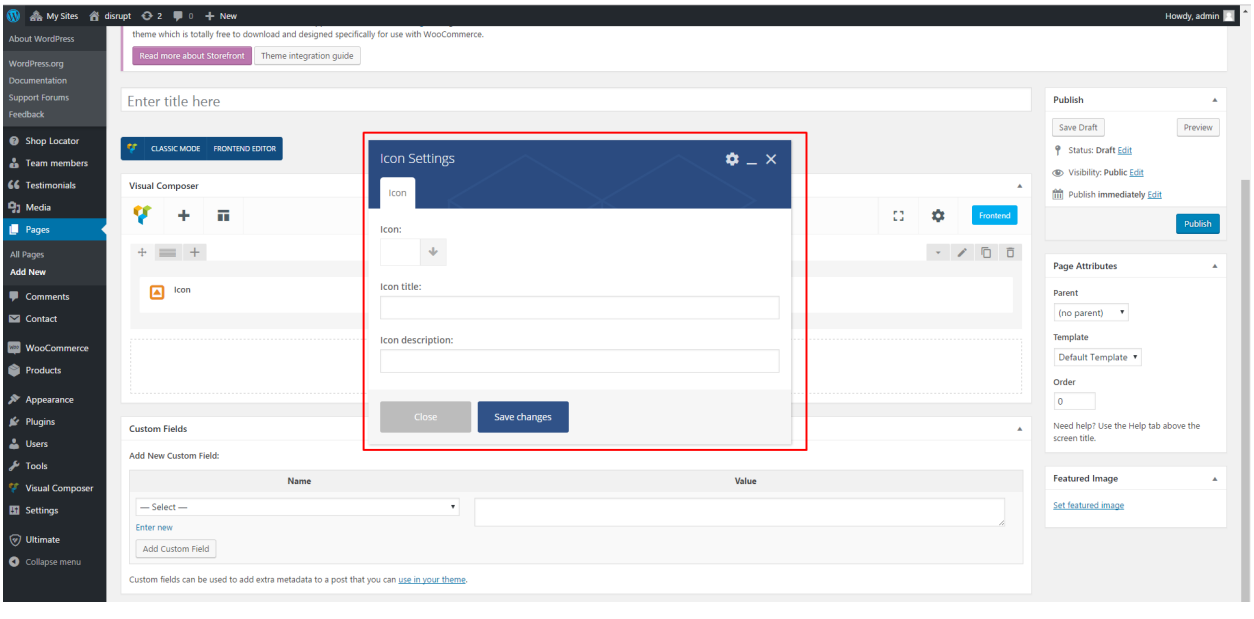
Note: All outputs are and should be escaped. Read more: Escaping

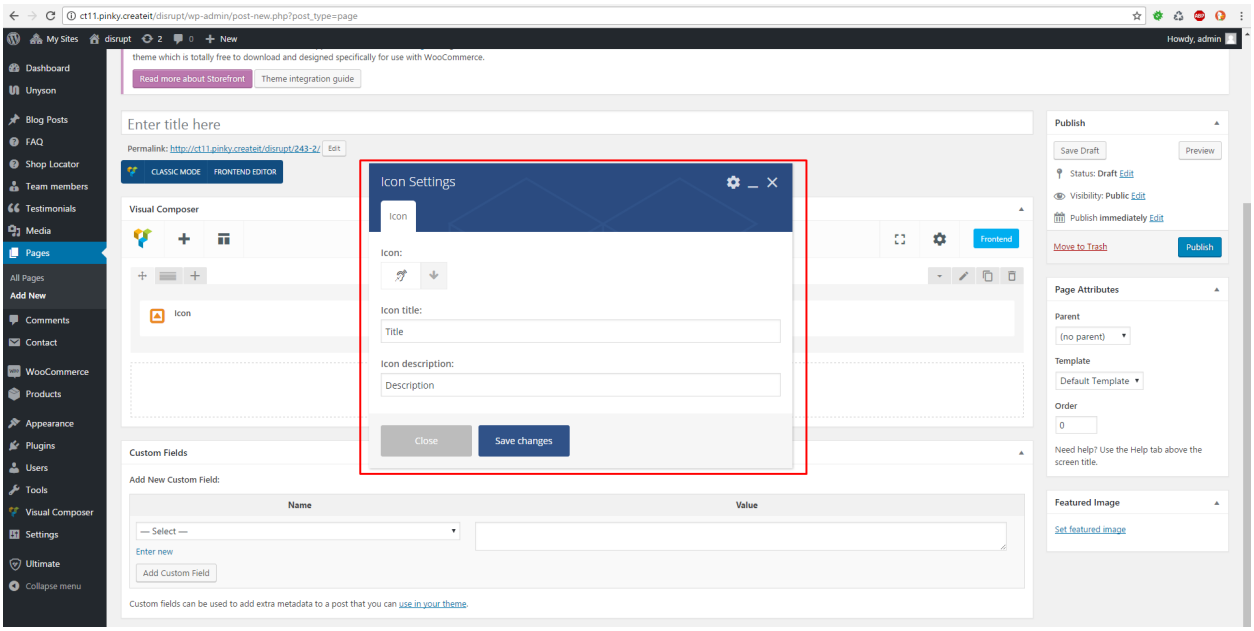
Shortcode configuration: config.php

```
<?php if ( ! defined( 'FW' ) ) {
    die( 'Forbidden' );
}

$cfg = array(
    'page_builder' => array(
            'title'            => __( 'Icon', 'ct_theme' ),
            'description'      => __( 'Displays icon', 'ct_theme' ),
            'tab'              => __( 'Disrupt', 'ct_theme' ),      // usually the␣
↪theme name
            'icon'             => 'fa fa-caret-square-o-up fa-2x',
    )
);
```

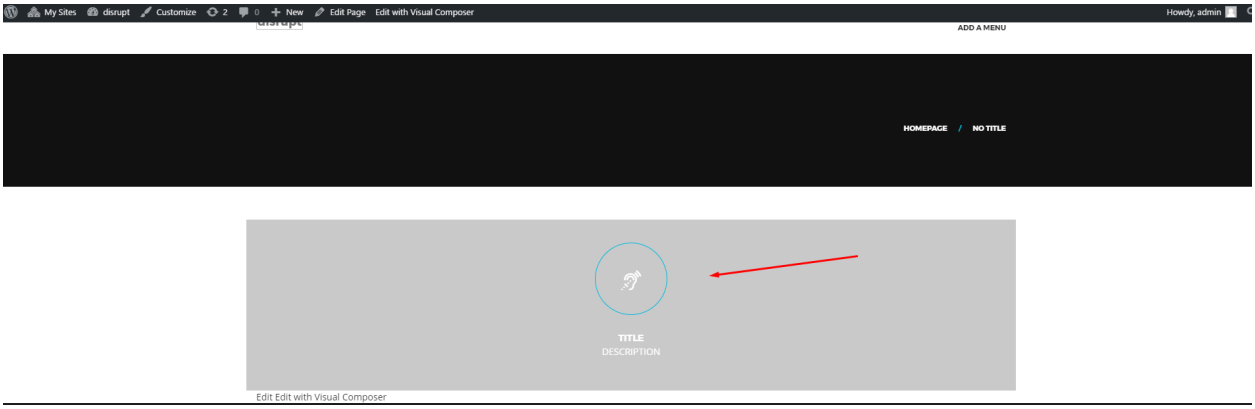
Final effect: shortcode in VC backend editor







Final effect: shortcode rendered



1.4.4 Custom Post Types

If you're regularly building different types of websites, you may reach a point when the default WordPress Content Types are no longer suffice for your projects. That's when you need to get started using Custom Content Types on your website. Pods gives you the functionality to build Content Types, but you need to put thought and planning into what you need.

Think About Your User

Workflow

The first thing to keep in mind, is that your Content Types are going to be used day-to-day by the website's user, which may not necessarily be you. Think about your user's workflow. This is going to be very different to a development workflow. Put yourself in the position of your user. How are they going to create content within this website? What

sort of content is going to be output on the front end? The type of content that your user needs is going to determine what you create.

Sit down with a piece of paper and sketch out the user's workflow. Figure out how they're going to create and use your content types.

Labelling

When you are creating an interface for someone else, you need to ensure that all of your terminology and labelling is clear. Avoid using developer terminology or terms that would be unfamiliar to the user. As much as possible, be precise. If you can't think of the correct word, a thesaurus is your friend.

Think About Your Visitor

As well as thinking about your user, you need to think about the content that your website's visitor will see on the front-end of the website. The Content Types that you create are going to be output, either through your theme or using Pods's built-in shortcodes. Ask yourself some questions:

What information does the visitor need?

What information is most important?

How will the visitor navigate the content?

How will the visitor search the content?

Plan Your Content

With that advice in mind, start planning out your content. You're going to need to decide the following things:

What Custom Post Types do you need on your website? Each of these will be an individual item of content. For example, on an Ecommerce website, the Custom Post Type needed would be Products. How should the content be organized? You may decide that you want to create more than one Taxonomy for a Post Type. This will create a website in which the content can be queried in multiple ways. For example, for an Ecommerce website you might wish to have the non-hierarchical taxonomy Brand and the hierarchical taxonomy Product Type. What additional data will enrich your content? You can create Custom Fields which are meta data attached to your Content. Examples for an Ecommerce website might be price, release date, and colour. Remember that while Taxonomies are ways to group Post Types together, Custom Fields are a mechanism for attaching individual data to your content.

Here are some examples of basic Content Types you might create for different types of websites.

An Ecommerce website:

POST TYPE	TAXONOMIES	FIELDS
Products	Brand	Price
	Model	Sale Price
		Release date
		Shipping method

A book review website:

A book review website:

POST TYPE	TAXONOMIES	FIELDS
Books	Author	Number of Pages
	Genre	Cover Photo
		Publication Date

An events website:

POST TYPE	TAXONOMIES	FIELDS
Event	Event Type	Venue
		Start Time
		End Time
		Date
		Map

1.4.5 Register First Custom Post Type

Posts Settings Locations

Post related options should be set in the following locations:

- for custom post types defined in extensions: plugin extension directory (see `posts.php` in Plugin structure)
- for basic post types (post, page) and post formats: `theme/inc/posts.php` (see Theme structure)

It is necessary to register basic post types settings in the theme because they need to work **without** the theme plugin activated.

Note: Please note Unyson encourages to use `theme/framework-customizations` directory for basic posts settings (Unyson theme options), but it is **not** advisable to do so (unless such settings, metaboxes etc. are not necessary for basic blog/theme functionality).

Custom Post Type Example

To analyse example extension `ct-portfolio` with custom post type and taxonomy, there is a file `theme-plugin/extensions/ct-portfolio/posts.php` with the following example content:

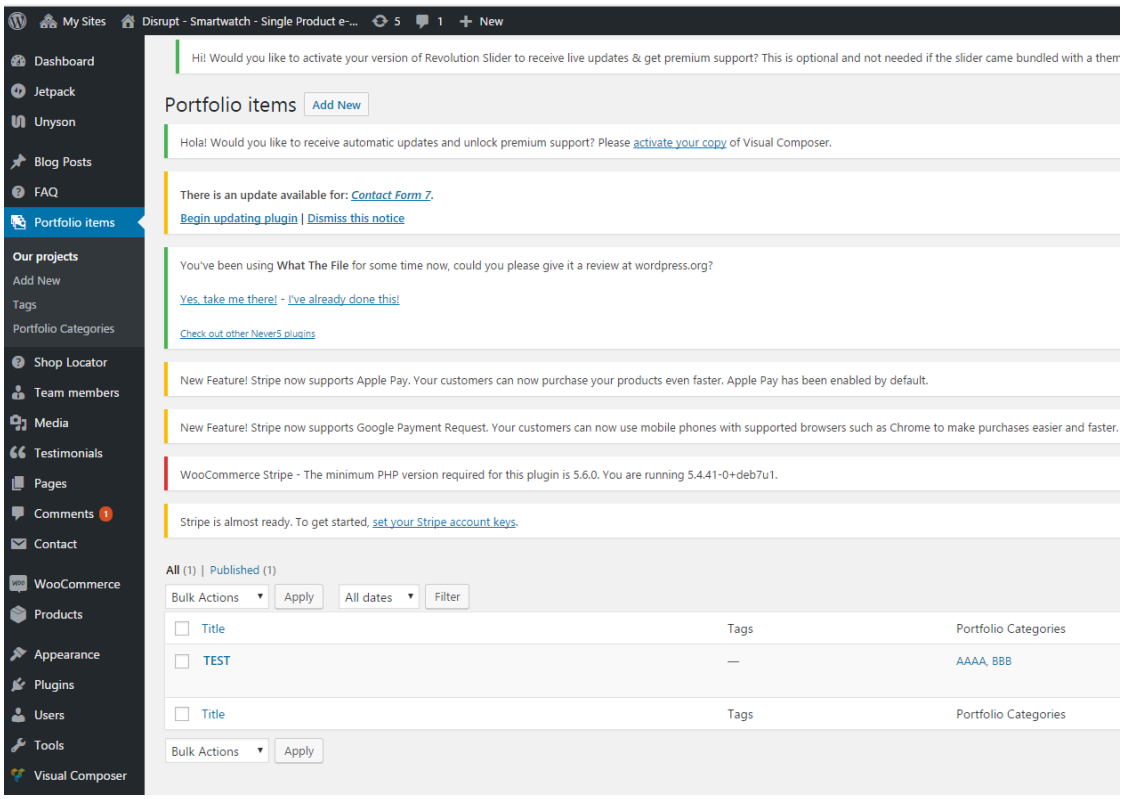
```
<?php if ( ! defined( 'FW' ) ) {
        die( 'Forbidden' );
    }

/** Register post type & taxonomy for extension */

$ct_post = fw()->extensions->get( 'ct-portfolio' );

register_post_type( $ct_post->get_post_type(), array(
```

```
        'labels'              => array(
                'name'               => __( 'Portfolio items', 'ct_theme' ),
                'singular_name'      => __( 'Portfolio Item', 'ct_theme' ),
                'menu_name'          => __( 'Portfolio items', 'ct_theme' ),
                'add_new'            => __( 'Add New', 'portfolio', 'ct_theme
↪' ),
                'add_new_item'       => __( 'Add New Portfolio Item', 'ct_
↪theme' ),
                'new_item'           => __( 'New Portfolio Item', 'ct_theme'␣
↪),
                'edit_item'          => __( 'Edit Portfolio Item', 'ct_theme
↪' ),
                'view_item'          => __( 'View Portfolio Item', 'ct_theme
↪' ),
                'all_items'          => __( 'Our projects', 'ct_theme' ),
                'search_items'       => __( 'Search Portfolio Items', 'ct_
↪theme' ),
                'parent_item_colon'  => '',
                'not_found'          => __( 'No portfolio item found', 'ct_
↪theme' ),
                'not_found_in_trash' => __( 'No portfolio items found in␣
↪Trash', 'ct_theme' ),
        ),
        'singular_label'      => __( 'portfolio', 'ct_theme' ),
        'public'              => true,
        'publicly_queryable'  => true,
        'exclude_from_search' => false,
        'show_ui'             => true,
        'show_in_menu'        => true,
        'menu_position'       => 4,
        'menu_icon'           => 'dashicons-images-alt',
        'capability_type'     => 'post',
        'hierarchical'        => false,
        'supports'            => array( 'title', 'editor', 'excerpt',
↪'thumbnail', 'comments', 'page-attributes' ),
        'has_archive'         => false,
        'rewrite'             => array( 'slug' => $ct_post->get_portfolio_
↪slug() ),
        'query_var'           => false,
        'can_export'          => true,
        'show_in_nav_menus'   => true,
        'taxonomies'          => array( 'post_tag' )
) );

/**
* Creates taxonomies
*/


//Register taxonomy
$category_names = array(
        'singular' => __( 'Portfolio Category', 'fw' ),
        'plural'   => __( 'Portfolio Categories', 'fw' )
);

$labels = array(
        'name'              => sprintf( _x( '%s', 'taxonomy general name',
↪'fw' ), $category_names['plural'] ),
```

```
        'singular_name'     => sprintf( _x( '%s', 'taxonomy singular name',
↪'fw' ), $category_names['singular'] ),
        'search_items'      => __( 'Search categories', 'fw' ),
        'all_items'         => sprintf( __( 'All %s', 'fw' ), $category_
↪names['plural'] ),
        'parent_item'       => sprintf( __( 'Parent %s', 'fw' ), $category_
↪names['singular'] ),
        'parent_item_colon' => sprintf( __( 'Parent %s:', 'fw' ), $category_
↪names['singular'] ),
        'edit_item'         => sprintf( __( 'Edit %s', 'fw' ), $category_
↪names['singular'] ),
        'update_item'       => sprintf( __( 'Update %s', 'fw' ), $category_
↪names['singular'] ),
        'add_new_item'      => __( 'Add New category', 'fw' ),
    'new_item_name'     => sprintf( __( 'New %s Name', 'fw' ), $category_
↪names['singular'] ),
    'menu_name'         => sprintf( __( '%s', 'fw' ), $category_names['plural
↪'] )
);

$args = array(
    'labels'            => $labels,
    'public'            => true,
    'hierarchical'      => true,
    'show_ui'           => true,
    'show_admin_column' => true,
    'query_var'         => true,
    'show_in_nav_menus' => true,
    'show_tagcloud'     => false,
    'rewrite'           => array(
            'slug' => $ct_post->get_taxonomy_slug()
    ),
);

register_taxonomy( $ct_post->get_taxonomy_type(), esc_attr( $ct_post->get_
↪post_type() ), $args );
```

This file is automatically included as long as the extension is activated.

Result:



Create New Post Type Extension

Before you create a new extension, it is a good idea to browse already written extensions at Samples Gitlab repository

In order to create a new post type extension, please copy the example **ct-team** extension from boilerplate at *theme-plugin-boilerplate/extensions/ct-team* and change settings in *class-fw-extension-ct-team.php* and *posts.php* file according to new extension needs. More information about extensions can be found in Extensions documentation

Post Settings Example

If you need to add custom metaboxes to post formats, you can do so in `theme/inc/posts.php` file, as in the snippet below:

```
<?php if ( ! defined( 'ABSPATH' ) ) {
        die( 'Direct access forbidden.' );
}

/**
* Post settings
*/

/** Register metaboxes */
add_action( 'admin_print_scripts', 'fw_ct_bee_display_metaboxes', 1000 );
function fw_ct_bee_display_metaboxes() {
        add_meta_box( "post-video-meta", esc_html__( "Video format settings",
↪ 'ct_theme' ), "fw_bee_post_video_meta", "post", "normal", "high" );
        add_meta_box( "post-audio-meta", esc_html__( "Audio format settings",
↪ 'ct_theme' ), "fw_bee_post_audio_meta", "post", "normal", "high" );
```

```
}

/** Register custom post save action */
add_action( 'save_post', '_fw_ct_bee_post_save_details' );

/** meta render callback */
function fw_bee_post_audio_meta() {
    // TODO implement callback
}

/** meta render callback */
function fw_bee_post_video_meta() {
    // TODO implement callback
}

/** post save callbacks */
function _fw_ct_bee_post_save_details() {
    // TODO implement callback
}

...
```

1.4.6 Customizer

Customizer lets users live preview and modify many of their sites' global appearance settings, such as

- header: display/hide, choose image, set position
- breadcrumbs display/hide, set custom names
- theme motive colors,
- blog page and single post,
- custom post types,
- footer,
- widgets,
- any global setting

You can access the Customizer from any page or post on your site: just click `customize` at the top bar of a page when logged in.

Intro to what Customizer is:

Implementation details documentation: Customizer

1.4.7 Create Your First Customizer Option

In your project, activate theme plugin with Unyson and create a file which will contain all customizer options: `/theme/framework-customizations/theme/options/customizer.php` with the following content:

```
<?php if ( ! defined( 'FW' ) )    {
      die( 'Forbidden' );
}

$options = array(
```

```
  'page-head'  => array(
    'title'              => 'Page Header',
    'options'            => array(

      /** whether or not the page head should be visible */
      'head_display'            => array(
        'value' => 1, // always define a default value
        'type'  => 'checkbox',
        'label' => esc_html__( 'Display page header', 'ct_theme' ),
            'desc'  => esc_html__( 'Check to display page header in top of the page␣
↪(customizable in options of every page)', 'ct_theme' ),
            'help'  => esc_html__( 'Do you want to display page header at the top of␣
↪the page?', 'ct_theme' ),
          ),

      /** whether or not the breadcrumbs should be visible */
      'head_breadcrumbs_display' => array(
        'value' => 0, // always define a default value
        'type'  => 'checkbox',
            'label' => esc_html__( 'Display breadcrumbs', 'ct_theme' ),
            'desc'  => esc_html__( 'Check to display breadcrumbs in page header', 'ct_
↪theme' ),
            'help'  => esc_html__( 'Do you want to display breadcrumbs at the top of␣
↪the page?', 'ct_theme' ),

        /** additional wp-customizer attributes*/
            'wp-customizer-args' => array(

          /** show this option only when 'head_display' option is active */
          'active_callback' => function () {
            return !! fw_get_db_customizer_option( 'head_display' );
          },

        ),
      ),
    ),
  ),
);
```

Then, in your project's `theme/header.php` file put the following code:

```
<?php

// check if the option has any value. if not ( eg. the default value was not set )␣
↪return true
if ( fw_get_db_customizer_option( 'head_display', true ) ) :

    get_template_part( 'page-templates/page-head' );

endif;

?>
```

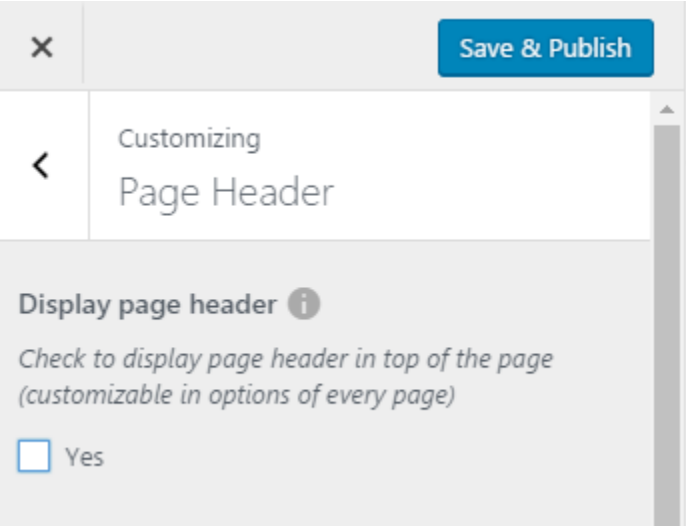
Next, create a file `/theme/page-templates/page-head.php` with the following content:

```
PAGE HEADER HERE
```

```
<?php

// check if the option has any value. if not ( eg. the default value was not set )␣
↪return false
        if ( fw_get_db_customizer_option( 'head_breadcrumbs_display' ) ):

    echo 'BREADCRUMBS HERE';

        endif;
```

Also, make sure you have `get_header()` call in your `theme/index.php` file or whichever page you should test.

Go now to the Customizer and check the results.



Further reading:

- Customizer options: Customizer
- Additional arguments: Customizer arguments
- Sass Compiler options in Customizer: Sass Compiler

1.4.8 Tasks & Examples & Hints

Contents

Shortcodes

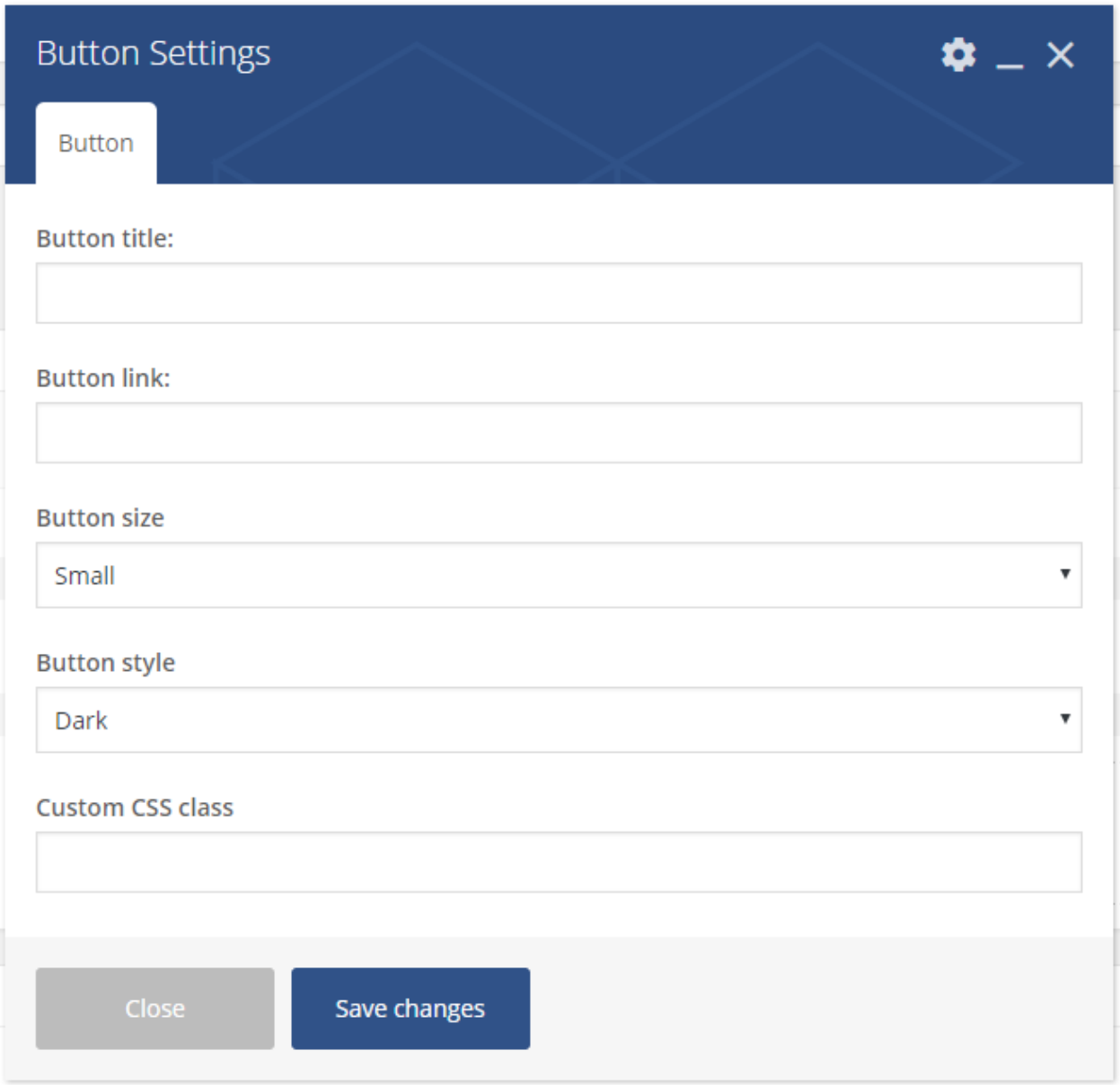
Test your knowledge of shortcodes by creating the following shortcodes in your project. Unyson way.

1. Simple Button

Create a simple button shortcode which allows you to:

- Enter a title,
- Enter a link,
- Select button size (small or large),
- Select button style (white or dark),
- Enter additional css classes.

The shortcode should look like this in the backend Visual Composer editor:



And should render an HTML like below, depending on options chosen:

```
<!-- Example one -->

<a href="http://link_example.com" class="btn btn-large bth-white example_custom_class
↪">title_example</a>

<!-- Example two. -->

<a href="#" class="btn btn-small btn-dark example_custom_class_2">title_example_2</a>

<!-- Example three. No link entered in options. No custom class entered. -->

<a class="btn btn-small btn-dark">title_example_2</a>
```

Helpful reading:

- Unyson options
- Unyson options: text
- Unyson options: select

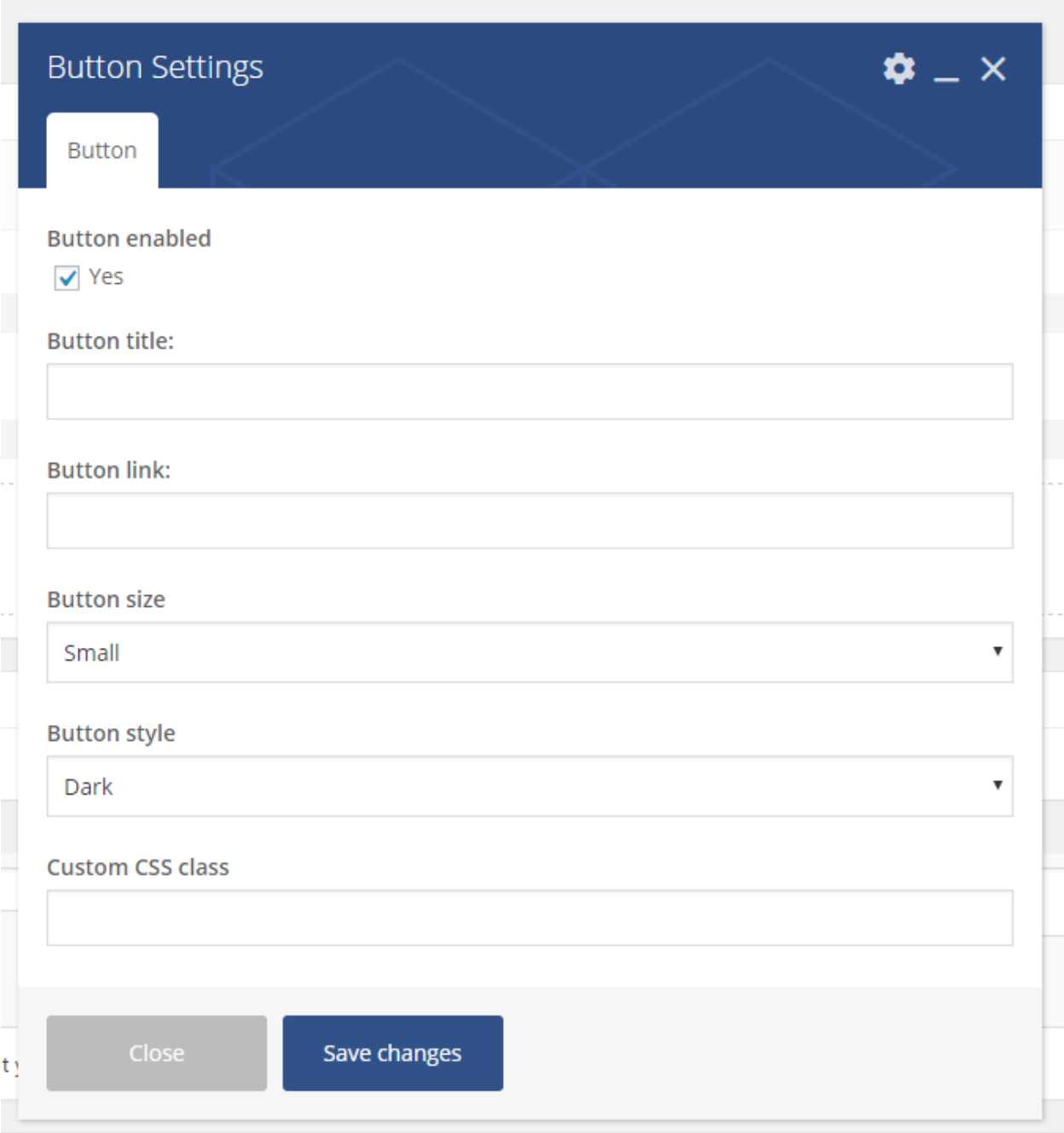
2. Button With Related Shortcode

Extend the previous button with following options:

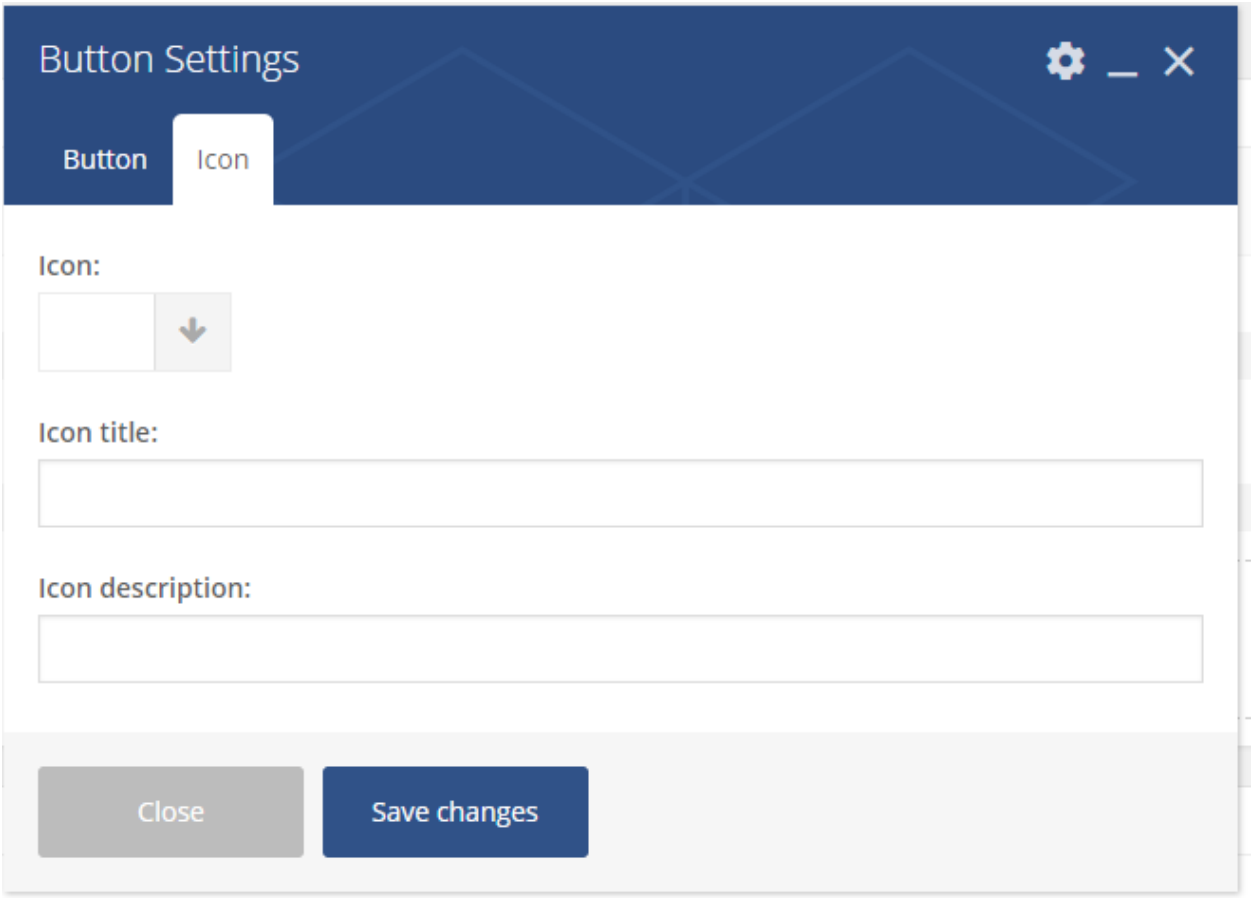
- Checkbox to enable / disable the button (render nothing when disabled),
- Select simple or separated button,
- If grouped, allow to choose an icon (as related shortcode `ct_icon` from the example),
- If simple, hide icon select.

The shortcode should look like this in the backend Visual Composer editor:

Simple button options view:



Separated button options view:



And should render an HTML like below, depending on options chosen:

```
<!-- Example one. Simple button. -->

<a href="http://link_example.com" class="btn btn-large bth-white example_custom_class
↪">title_example</a>

<!-- Example two. Separated button with icon. -->

<a href="http://button_link.com" class="btn-group btn-group--separated">
    <span class="btn btn-dark btn-separated btn-small ">button_title</span>
    <span class="btn btn-dark btn-separated btn-small ">
        <div class="ct-iconBox ct-iconBox--type1 ct-iconBox--white">
            <div class="ct-iconBox-icon"><i class="fa fa-instagram"></i></div>
            <h6 class="ct-iconBox-title">icon_title</h6>
            <p class="ct-iconBox-description">icon_desc</p>
        </div>
    </span>
</a>

<!-- Example three. Button disabled. Nothing to render. -->
```

Make sure you created `ct_icon` shortcode (as in example) beforehand in your project so you use it and not double functionality in button shortcode.

Helpful reading:

- Option type: 'shortcode'

- Dependency: Shortcodes Visual Composer Integration

Customizer

1. Background Color & Image

Create a simple customizer section which will allow users to:

- Select between background color and background image,
- Pick a background color (visible only when background color is selected),
- Upload a background image (visible only when background image is selected),
- Section should appear above all others Customizer options
- Implement basic HTML for preview

Helpful reading:

- Customizer options: Customizer
- Additional arguments: Customizer arguments
- Controls & further documentation: Customizer controls

1.4.9 Code samples

Before you start coding your own shortcodes or extensions, check out if they are not already written at Bumblebee Samples documentation.

1.4.10 Framework

BumbleBee is a WordPress Theme Framework created by createIT. It partially derives from the Unyson Framework. It expands Unyson with new extensions and options and internally also has Unyson code.

Documentation

All the docs regarding both Bumblebee and Unyson can be found here: Bumblebee

1.4.11 For Dummies - Wordpress Basics

Here you can find basic Wordpress programming tutorials.

1. Introduction to Awesome WordPress

Let's start from the basics ie. what is WordPress and why it's so cool. Below you will find step by step tutorials which should explain the basics.

- Watch the tutorial: WordPress Beginners Guide
- Understanding and Working with Data in WordPress
- Understanding WordPress Multisite

2. WordPress Themes and Plugins

So you now what WordPress is. Now it's time to find our what makes WordPress awesome and universal platform. Below you will find tutorials about two most powerful WordPress elements: Themes and Plugins

- Watch the tutorial: WordPress Themes
- Child Theme
- Creating Child Themes
- Overriding Parent Theme Functions in Your Child Theme
- How to Install WordPress Plugin
- Plugin Territory

3. Post Types, Shortcodes, Widgets, Customizer

WordPres Post Types

Originally WordPress was only a blogging platform. That's why the most important content type in WordPress are posts. But WordPress post types are not only refer to blog section. Read the article below to find out more.

- Watch the tutorial: Custom Post Types
- Understanding and Working with Posts in WordPress
- Posts Formats Inside and Out
- Beginner's Guide to WordPress Taxonomies
- Post Types, Taxonomies and Permalinks
- Taxonomies: Themes or Plugins

Shortcodes and Widgets

A shortcode is a WordPress-specific code that lets you do nifty things with very little effort. Shortcodes can embed files or create objects that would normally require lots of complicated, ugly code in just one line. Shortcode = shortcut. Widget can be treated as shortcode, but dedicated to sidebar areas

- Getting Started With WordPress Shortcodes
- Introduction to Creating Your First WordPress Widget
- Building Custom WordPress Widgets

Images and Meta Data

- WordPress Images
- Retina Support

WordPress Customizer

Customizer can be a powerful tool for editing general website look. In one place you can define number of theme settings, which user will be able to changes with live preview of changes

- Watch the tutorial: WordPress Theme Customizer

4. Visual Composer and Common Plugins

Plugins enhance the capabilities of WordPress.There are over 45 055 free plugins in WordPress.org repository

- Getting Started with Visual Composer
- Beginner's Guide to Using WooCommerce
- Slider Revolution Beginner Video Guides
- Mini Guide to Contact Form 7
- The Beginner's Guide to WordPress SEO by Yoast

5. Tips & Tricks

- Tips for Aspiring WordPress Developers
- 5 "Saintly" Practices that All WordPress Developers Should Strive For
- Common WordPress Development Mistakes and How to Fix Them
- How to Include JavaScript and CSS in Your WordPress Themes and Plugins
- Using Page Templates in Your WordPress Theme

6. Theme Testing

Each WordPress project before upload needs to be tested. By testing we mean checking every single theme element, shortcode and option. It will save your time when you do first tests while creating each element.

Developers Tools for testing

- Debug Bar - check if any PHP errors, warnings are displayed and fix them,
- Debug Bar Cron - extension for *Debug Bar* plugin
- Monster Widget - test all default WordPress widgets in all types of sidebars,
- WooCoomerce Monster Widget - if your theme support WooCoommerce
- Theme Check - necessarily!
- ThemeForest theme check
- WPtest.io

1.4.12 FAQ - Developers Questions

Theme Plugin

Error after activating theme plugin: Cannot redeclare class ctBeePlugin?

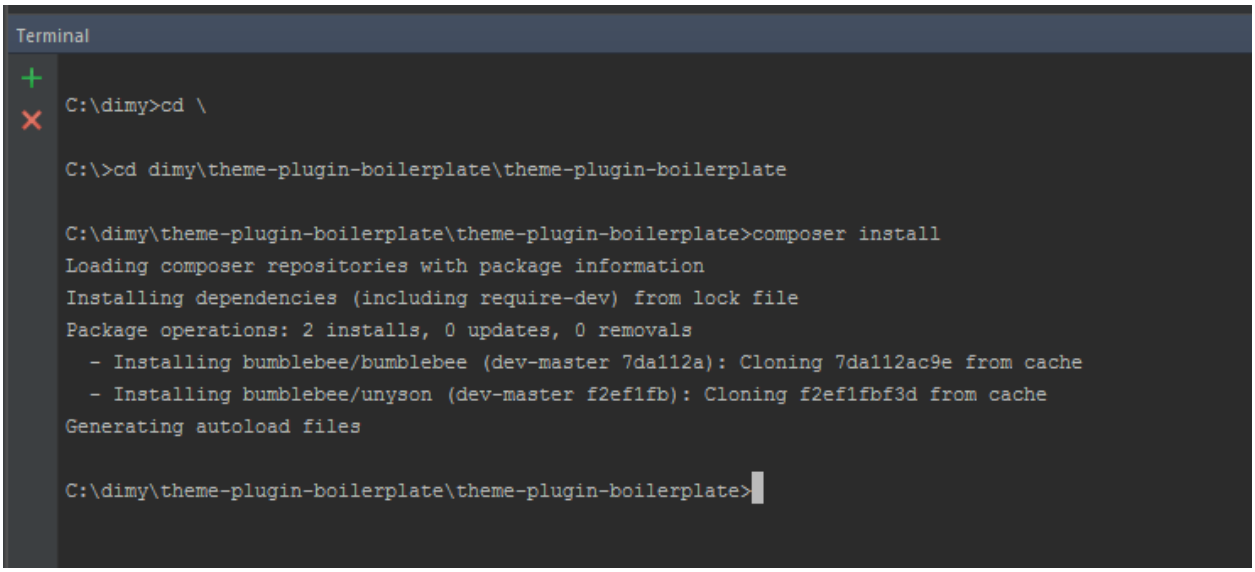
It means there is more than one theme plugin activated. Please deactivate all theme plugins (or all plugins in general), then activate the theme plugin again.

Site crashes after activating theme plugin. What to do?

Update framework files (see below)

How to update framework files?

Run *composer update* in theme plugin directory terminal. Just like in this example:



Visual Composer

How to add/modify an option to an existing Visual Composer shortcode?

There are two possible scenarios

1. Add new param to a shortcode (eg. new select input)
   (a) Open `theme-plugin/extensions/visual-composer/hooks.php`
   (b) Add `_filter_fw_visual_composer_extend_shortcodes` filter hook callback to modify the existing vc shortcode
   (c) Example code to add `decoration` and `clound-bgcolor` options to `vc_section` vc shortcode

```
add_filter( '_filter_fw_visual_composer_extend_shortcodes',
↪'fw_ext_visual_composer_extend_shortcodes' );
function fw_ext_visual_composer_extend_shortcodes( $shortcodes␣
↪) {

    $shortcodes['vc_section'] = array(
        'decoration' => array(
            'type' => 'select',
            'choices' => array(
                '' => esc_html__( 'None', 'ct_theme' ),
                'top-cloud' => esc_html__( 'Top Cloud', 'ct_
↪theme' ),
                'bottom-cloud' => esc_html__( 'Bottom Cloud',
↪'ct_theme' ),
                'top-bottom-cloud' => esc_html__( 'Top and␣
↪Bottom Cloud', 'ct_theme' ),
            ),
            'label' => esc_html__( 'Select cloud decoration',
↪'ct_theme' ),
        ),
        'cloud-bgcolor' => array(
            'type'      => 'color-picker',
```

```
            'value'     => '#fff',
            'label'     => esc_html__( 'Cloud Background Color',
↪ 'ct_theme' ),
        ),

    );

    return $shortcodes;
}
```

2. Add new options to an existing shortcode param (eg. new style option)
   (a) Open `theme-plugin/extensions/visual-composer/hooks.php`
   (b) Add `vc_after_init` action hook callback to modify the existing vc shortcode param
   (c) Example code to add new `style` param options of `vc_single_image` vc shortcode

```
add_action( 'vc_after_init', 'fw_ext_visual_composer_update_
↪shortcodes' );
function fw_ext_visual_composer_update_shortcodes() {

    // get 'style' param of 'vc_single_image' shortcode
    $param = WPBMap::getParam( 'vc_single_image', 'style' );

    // add options
    $param['value'][__( 'Framed', 'ct_theme' )] = 'ct-framed-
↪image';
    $param['value'][__( 'Framed light', 'ct_theme' )] = 'ct-
↪framed-image ct-framed-image--Light';
    $param['value'][__( 'Framed dark', 'ct_theme' )] = 'ct-
↪framed-image ct-framed-image--Dark';

    // update the shortcode param
    vc_update_shortcode_param( 'vc_single_image', $param );

}
```

That's all. If you furthermore need to change the markup of a VC shortcode, read below.

How to change the markup of a VC shortcode?

Just create `vc_templates` folder in your theme root folder and copy and modify vc shortcode templates there, ie. `wp-content/themes/project_name/theme/vc_templates/vc_single_image.php`. Read more.

General

The link of single custom post type page doesn't work or redirects to homepage

You may need to rebuild the permalinks. Go to Settings -> Permalinks and press save.

(supports) I'm trying to work on a legacy theme however files are missing

After checking out the theme, run *composer update* in main theme directory.

Composer update doesn't work

Please make sure you have a valid ssh key entered here https://gitlab.createit.pl/profile/keys . If not, follow the instructions to generate one.

After you add the key, run below command in git bash:

```
ssh -T git@gitlab.createit.pl
```

If asked to add host, type *yes* and press *enter*.

How to make shortcodes (like CF7) render in widget areas?

Add this filter to `theme-plugin/extensions/shortcodes/hooks.php`

```
// parse all text in widget text as shortcodes
add_filter( 'widget_text', 'do_shortcode' );
```

1.5 Quiz

Below, there is a quiz with 20 or so questions. You can find answers to all of them in this documentation. A few questions require some investigation, so take your time.

CHAPTER 2

First Project Flow

2.1 Introduction

All projects created in ThemeForest/Marketplaces Team are dedicated to be uploaded for sale on themeforest.net, which is a part of Envato Marketplaces. Since every item available on Envato Marketplaces is first subjected to review process, our items (including your project) need to meet number of requirements. The basic requirements can be found here.

But there is a lot more - please refer to:

- Quality Assurance
- Upload & Reject

You will mostly work on WordPress Themes and all information in this documentation refer to coding the WordPress Theme. In the following chapters you will find guide to our internal WordPress framework - Bumblebee, daily project flow, and development process.

First of all, you need to be aware that your responsibilites would be much more than just coding.

Each project is divided to the following stages of development:

- Project Specification and Kick-off Meeting - Prepare the plan for your project and list all elements in tasks. Make sure that you understand how each element should work and what needs to be done.
- Development - Divide your tasks in 4 weekly SPRINTS and code!
- Testing - Once development is done test everything. Twice.
- Demo - Create demo pages for our customers.
- Optimizations - We want our themes to be blazing fast so PageInsights will help us out
- Upload - Please deliver theme package and promo material to Team Manager - she wil do the rest.
- Reject - Unfortunately there will be rejects. Be brave and fix the bugs!

2.2 Before You Start

Before you start the project, please make sure you have all the necessary information provided by the project manager, including:

- Project name
- Project team members: backend / frontend / qa
- psds and/or contact to a graphics designer
- Gitlab project access
- Todo project access
- Specification due date
- Kickoff date
- Project deadline

If anything from the list above is missing - don't hesitate to create appropriate task on todo and assign it to Project Manager.

2.3 Project Specification

Specification is a document which contains all the project's screens that needs to be created along with the comments how each element should work and what options it should have.

Specification for your first project will be provided by Project Manager (PM). For next projects you will need to prepare your own specification and submit it for PM acceptance before you will start working on the project.

Comments in specification will describe the following types of elements:

- **VC shortcode** - element should be created with Visual Composer plugin; There are 2 main types of shortcodes: with static and dynamic content; Static content can be defined directly via shortcode (e.g. via input or text area field). Dynamic content should be loaded from the custom post types content; Before you start creating custom VC shortcode - make sure that we don't have any similar element in default Visual Composer plugin pack.
- **CPT** - Custom Post Type - for elements like: portfolio, testimonials, FAQ, team members, products, etc. you should register a dedicated post type. Thanks to that user can easily add e.g. a new product, team member or question for FAQ section without need to change the static pages content.

  Content created with custom post types can be also displayed with a shortcode. EVERY CPT dedicated shortcode should have additional settings for filtering and sorting:

  - Limit - number of elements to display,
  - Order by - in which order data should be fetched: date, ID, date modified, page order, random order,
  - Order - descending/ascending,
  - Category/TAG/Custom Taxonomy filtering,
- **Widget** - element can be added via sidebar areas,
- **Customizer** - element editable via Theme Customizer,
- **Plugin** - when element need specific plugin eg. "Plugin - Slider Revolution",

Once you've got your first project specification, read it carefully and please make sure that you understand how each element should work.

Maybe you would like to present your ideas for the particular components? Write it down and speak up on the Project Kick-off Meeting

2.3.1 Sample specification

Here you can view a sample specification just to know what to expect: Sample specification

2.4 Project Kick-off Meeting

You already have the project specification. You also had some time (1-2 days) to analyze all the project components.

Maybe you have also some ideas how to improve elements described by Project Manager? That's the perfect time for project kick-off meeting.

It's time to discuss all the tasks and your ideas, raise doubts or objection.

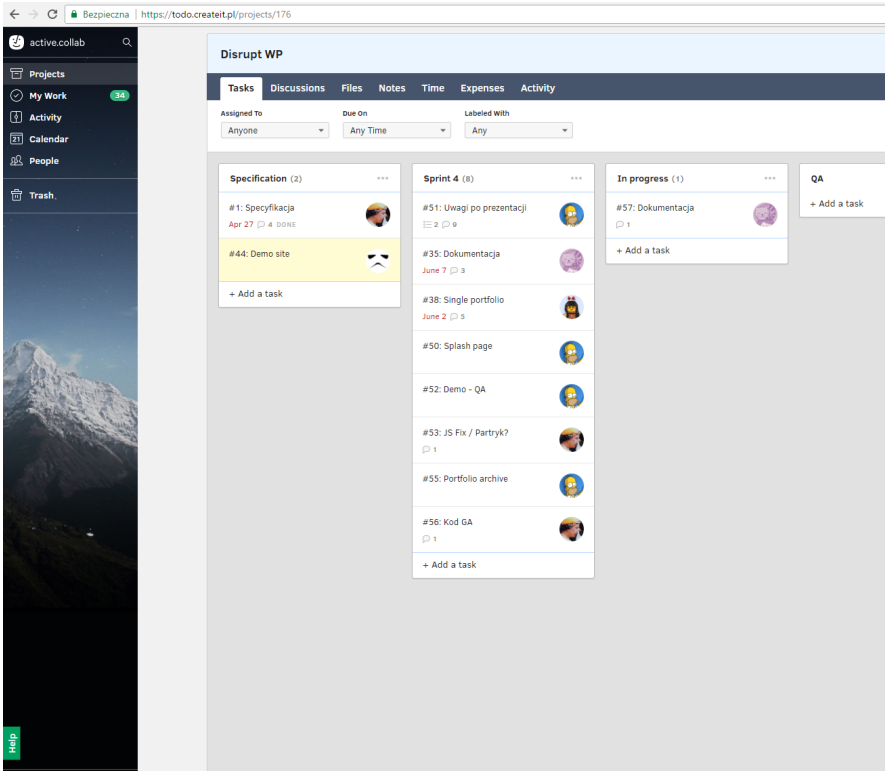
Take your notebook with you and make sure that you ask anything that concerns you.

2.5 Daily Project Flow

Each project is divided to weekly **SPRINTS**. Each SPRINT is a **TASK LIST** scheduled for the next week. There are also the following task lists:

- **SPRINT No.** - next, numbered sprint(s); always start work from the current sprint (from the top of the list to bottom),
- **In Progress** - Move your task to *In Progress* task while working on it,
- **Feedback** - if you have any questions or comments, use the *Feedback* list and assign the task to the person who you expect the reply/information from,
- **QA** - when development is done move the task to *QA list* and assing to QA or Team Leader,
- **Resolved** - once QA and fixes are done PM or QA will move the task to *Resolved* list.

Completed - task will be marked as **Completed** by a Team Leader after verification.



Here's a screenshot of a sample project task list:

If some tasks from SPRINT 1 are not finished at the end of the sprint they will be moved to SPRINT 2 by Team Leader at the end of the sprint (usually it's Friday).

2.6 Project Tasks

Now as you know everything about your project - create a separate task for each component described in specification.

Component is a single shortcode, template or widget, so if you will be using WooCommerce in your theme you need to create more then 1 task for this, e.g.:

- WooCommerce Index Page,
- WooCommerce Single Product Template,
- WooCommerce Customizer Options,
- Product Shortcode,
- Product Shortcode 2,
- etc.

Add as much information as you have about component settings, the way of working, useful extensions/plugins that you could use for it, etc.

Once tasks are created you need to divide them into SPRINTS.

Feel free to use *Project Template* if you are not able to estimate how much time you will need for particular tasks. Once your SPRINTS are scheduled contact Project Manager - he will help you to verify the schedule.

2.7 Weekly Progress Meetings

At the end of every SPRINT you will have the meeting with Project Manager to verify the work progress. It's the best time to report any problems with project, ask questions or just show off your awesome work. The meetings and possible task redistribution will be hold on Fridays.

Before every meeting please make sure that the latest changes are pushed to repository and your dev server is up to date (contains all the changes coded by other team members working on the project).

CHAPTER 3

Coding Your First Project

3.1 Coding Your First Project

Before you start coding, please make sure you have everything you need, Required info checked, as well gone through Wordpress And Boilerplates Installation successfully.

Important: Before doing sprint tasks, make sure you have all necessary understanding of the project flow.

3.1.1 Header

Header is the first part of theme you should develop. It's placed in header.php file wich contains html head section, and content (usualy it is logo, menu and registrated widgets area).

Things that header should contains:

- Doctypes,
- Conditionals IE8, 7, 6 (optionally),
- Meta tags to ensure that you theme is rendered properly,
- Site header (site title / logo),
- Navigation menu,
- Registered widgets area (if project need them),
- Celan and commented code.

Head area

First part of header.php file is head area. Be shure that things listed below are included corectly:

- Proper DOCTYPE

- Opened `<html>` tah with `language_attributes()`
- The `<meta>` charset element should be placed as first
- Use `bloginfo()` to set the `<meta>` charset and description elements
- Use `wp_title()` to seth the `<title>` element
- Use Automatic Feed Links to add feed links
- Add call tp `wp_head()` before you close `</head>` tag.

Remember - do not link the theme stylesheets in the Header template. Use the `wp_enqueue_script` and `wp_enqueue_style` action hooks in inc/static.php file.

Example of correctly-formatted HTML head area:

```
<!DOCTYPE html>
    <html <?php language_attributes(); ?>>
        <head>
            <meta charset="<?php bloginfo( 'charset' ); ?>" />
            <title><?php wp_title(); ?></title>
            <link rel="profile" href="http://gmpg.org/xfn/11" />
            <link rel="pingback" href="<?php bloginfo( 'pingback_url' ); ?>" />
            <?php if ( is_singular() && get_option( 'thread_comments' ) ) wp_enqueue_
↪script( 'comment-reply' ); ?>
            <?php wp_head(); ?>
        </head>
```

Body Section

Do not put cutom classes on body element. To pust custom classes on body use function in function.php file (`fw_ct_bee_filter_add_body_classes()`).

Site Header

Usually we should place here a logo and link to the main page but there is very important to protect theme to display page titile when header image is not attached. To use header image you must register function in fuction.php file, to see how you can do it visit Codex

Example of correct-placed logo and page title section.

```
<a href="<?php echo esc_url( home_url( '/' ) ); ?>" rel="home" class="navbar-brand">
    <?php if ( has_header_image() ) : ?>
        <img src="<?php header_image(); ?>" width="<?php echo get_custom_header()->
↪width; ?>" height="<?php echo get_custom_header()->height; ?>" alt="<?php echo esc_
↪attr( get_bloginfo( 'name' ) ); ?>">
            <?php else: ?> <h1> <?php echo esc_attr( get_bloginfo( 'name' ) ); ?></h1>
↪ <?php endif; ?>
</a>
```

Navigation menu

Header is file where we usually place the site main navigation. To do that in wordpress we use function `wp_nav_manu()`. If you want to use the default bootstrap classes in wp navigation you can use this function . Sumbenu items should display correctly, you should supprt drop-down menu for submenu items. Dropdown menus is good practice, them allowing showing menu depth instead of just showing the top level.

> **Note:** **Remember** when you use mega menu be sure to use default plugin style method. DO NOT HARDCODE MEGA-MENU PLUGIN STYLE IN OUR CSS FILE.

Another thing you should remember when you use mega-menu plugin is that you mustd style default wordpress menu too.

Widgets and custom areas

> **Note:** As you now the theme should be widgetized as fully as its possible it means that in header should be registered at least one widget area.

Every additional things (like buttons, links, phone numbers and other that are not a widgets) should be placed in customizer in header/navigation section. Think about put every options for header/navigation (eg. type of navigation - big/small or text in aditional button) to section header/navigation in customizer.

For more info abiut developing customizer options see Customizer section in this documentation.

3.1.2 Footer

Footer is a second thing that you should develop when you building a new wp theme. It's in footer.php file.

Footer should contain a severals widget areas, one to each column/row (it depends at the project - see Sample specification). Every option of footer (eg. numbers of cols, rows, post-footer) must be placed in Customizer customizer. Remember to style every widget area properly. You should use monster widget to put every default Wordpress widget and style them to looks good in footer area. Some things in footer could be placed from customizer opstions (eg. telephone numbers in complicated designs, copyrights) but when it is possible avoid that in favour of widegt area.

Similar to the header, when you placed the logo in the footer it should be linked to the main page. Second thing that you should check is moment when attached header image is not displayed - it should display page title. You can use code below:

```
<a href="<?php echo esc_url( home_url( '/' ) ); ?>" rel="home" class="footer-brand">
    <?php if ( has_header_image() ) : ?>
        <img src="<?php header_image(); ?>" width="<?php echo get_custom_header()->
↪width; ?>" height="<?php echo get_custom_header()->height; ?>" alt="<?php echo esc_
↪attr( get_bloginfo( 'name' ) ); ?>">
            <?php else: ?> <h2> <?php echo esc_attr( get_bloginfo( 'name' ) ); ?></h2>
↪ <?php endif; ?>
</a>
```

Remember to use `wp_footer()` call before you closing body tag.

3.1.3 Blog

Before you start

First thing you should do before you start developing blog and pages in wp theme (of course only if you didin't done this before) is download and import to your wp installation Wordpress Theme Unit Test Data it helps you to develope your wp theme. Second thing you should have and which helps you in developing blog is WP Monster Widget .

Page structure

Your first step should be index.php file. This one is responsible for displaying post on page. It depends form the project but usually your this page should be displayed in three variants:

- Page with no sidebar
- Page with sidebar on left side
- Page with sidebar on right side

This one like any other blog setting should be set in blog section in Customizer options.

Note: To display post you can use Bumblebee theme function `fw_ct_bee_display_post()`

Remember to proper style all worpdress lists pages:

- Archive `archive.php`
- Author `author.php`
- Category `category.php`
- Tag `tag.php`
- Search `search.php`

Next thing you should do is develop every type of wp post. Wordpress by default have this type of posts:

- Standard `single.php`
- Aside `content-aside.php`
- Image `content-image.php`
- Video `content-video.php`
- Audio `content-audio.php`
- Quote `content-quote.php`
- Link `content-link.php`
- Gallery `content-gallery.php`
- Page `content-page.php`

Note: You can see how every post type should look property on wptest.io.

Sidebar

Blog sidebar is container with registered widget area. You can register blog sidebar in inc/sidebar.php.

Use register function like this:

```
register_sidebar( array(
    'name'          => esc_html__( 'Blog sidebar', 'ct_theme' ),
    'id'            => 'blog-sidebar',
    'description'   => esc_html__( 'Sidebar placed in blog page', 'ct_theme' ),
    'before_widget' => '<div id="%1$s" class="widget %2$s ct-sidebar-widget ct-u-
↪margin-bottom-30">',
```

```
    'after_widget'  => '</div>',
    'before_title'  => '<h4 class="widget-title ct-sidebar-widget-title">',
    'after_title'   => '</h4>',
) );
```

Next insert sidebar into pages and remember to put them into functions `is_active_sidebar()` :

```
<?php if ( is_active_sidebar( 'blog-sidebar' ) ):
        dynamic_sidebar( 'blog-sidebar' );
endif; ?>
```

For sidebar development put monster widget into your blog sidebar.

Note: You should style all widgets from the monster widget (which includes all default wordpress widgets) in your project style.

Custom post Types

When you finished with default blog and sidebar look your next step should be coding Custom Post Types if project need them templates. For more information see Custom Post Types.

Pages

The last thing you should do after you finished developing all posts types, lists pages and default widget is work on all custom pages in your project. Every project has unique pages wich you should code and register as a page template - eg. error 404 page - `404.php` .

After you finished develping Header, Footer and Blog you should go on Codex Theme Testing Process and check all the points.

3.1.4 Shortcodes

Finall stage of this part of your work. At this stage, you are doing tasks according to your task list in Active Collab.

Here's a reminder how to Create Your First VC Shortcode

Form the front-end side of your work the most important thing is details. You should be very precisious when you are coding your shortcode. Be sure to add all functions, think about the finall user and how he will be using this shortcode. Add every detail from your project. Think about extra things like animation or hoovers efects. When you have no idea what is correct behavior of shortcode ask yours PM. With doubts in design ask graphic designer. Good practice is use Pixel Perfect browser extension (For Chrome or For Firefox).

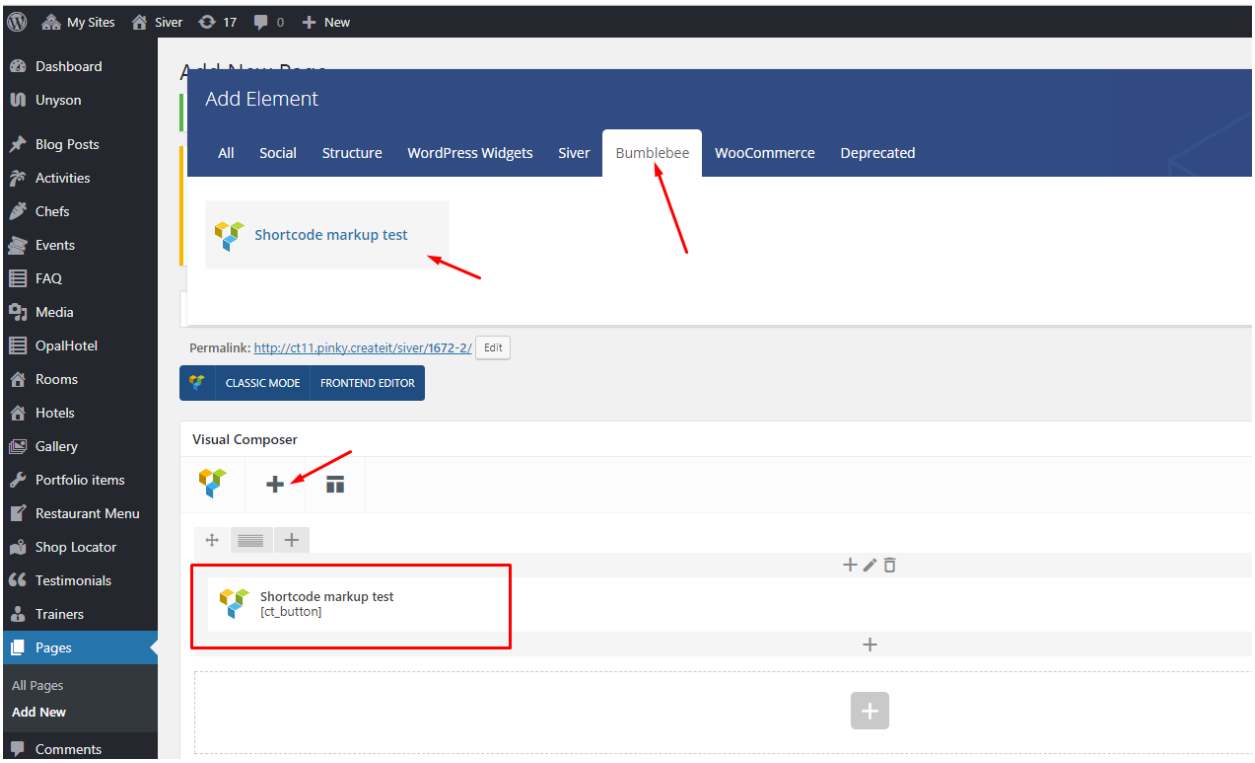
There is a few things you should remember at this stage:

- Remember to name shortcode properly and add him fitting icon
- Options of shordcode should be clear and well described
- Check all options of shortcode and how they looks befor you give shortcode to QA

When yours work at this stage is done you should create the same content as on project/PDF files. Remeber to check all padings, margins and others. They should be the same as in project.

Shortcode testing

You may test your shortcode (or any markup) with *Shortcode markup test* shortcode in *Bumblebee* tab which is available when you activate Theme Demo Plugin.



It will display the markup inside different sections and columns scenarios as in the example:

3.2 Building Package

Coming close to the end of development, it is a good idea to start thinking of building a package. Here is the detailed explanation of how to do it: Build

3.3 Coding Done

When the coding is done there are still few things that you need to take care of.

3.3.1 1. Demo

Except of generating theme package, which will be available for customers after purchase we need to prepare the Demo Pages. First, all demo sites should be created on a dedicated, clean, pinky/phinky. Once all elements, plugins, subpages, functionalities and content are there - we can start migration to the Live Site, which will be presented on ThemeForest. Once dev version of demo (on pinky/phinky) is ready and tested - you need to generate the demo content archive using Backup & Demo Content extension. The backup file will be then used for migration.

3.3.2 2. 1 Click Demo Import

Once production version of a demo, on a dedicated public URL is done (migration from dev server is finished) and tested it's time to generate final backup file with demo content for customers. You need to use for this the same Backup & Demo Content extension as previously. Demo Backup file should be attached in theme package in a separate folder.

Once demo backups are attached to theme package please verify the final size of the package. It should not be bigger than 100MB. If your package is bigger you will need to move 1 or more backups to external server instead of keeping it in theme package.

3.3.3 3. Documentation

Every item available on ThemeForest should contain a documentation. Documentation is prepared by Support Team based on information provided by you in finished tasks. Don't be surprised if Support Team will ask you for more detailed info about some of the components. Every element and option should be described as accurately as possible, including screenshots, videos, etc. Make sure that all component tasks are closed at least 3 days before the Upload Day, because that's the minimum time needed by support to create a documentation. It's important to not change layout, labels or any options in admin panel from the moment when Support starts creating the documentation. Otherwise screenshots in documentation won't be consistent with the admin panel visible for customers.

Documentation is available online - Support will provide you a link, but we need to also attached offline version of it. Usually it's a PDF file, also provided by Support which needs to be attached in final package by you.

3.3.4 4. SPLASH page

When designing a splash page, remember to add the following JavaScript which hides Envato top bar:

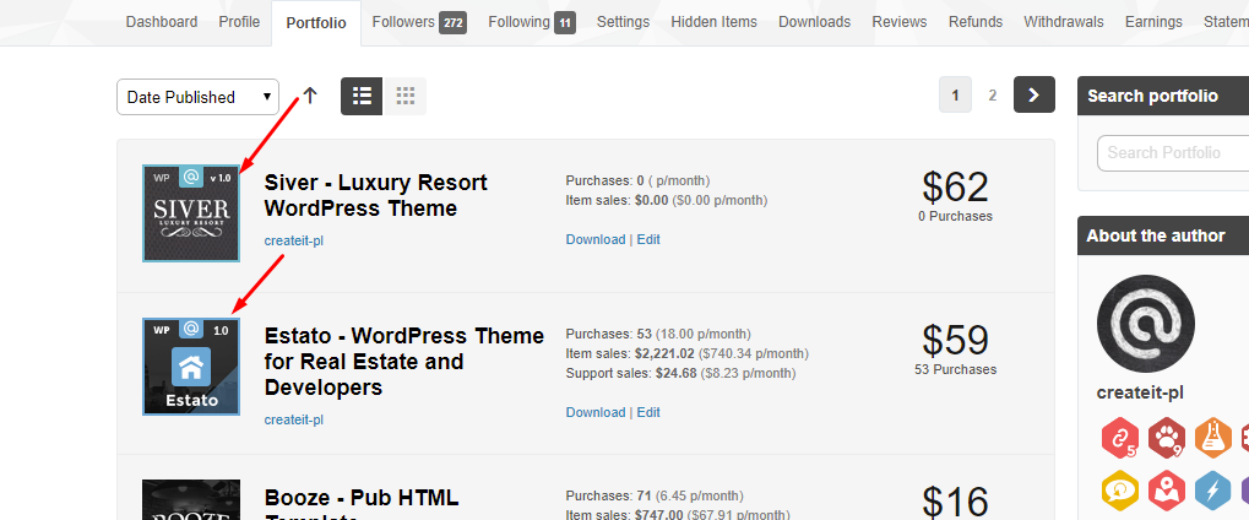
```
// remove any externals iframes
try {
    if (top.location != location) {
        top.location.href = document.location.href;
    }
} catch (e) {
    console.log(e);
}
```

3.4 Promo Materials

All promo materials are prepared by graphic designer and designed by Art Director. But it is your responsibility to inform Art Director via new task on todo respectively earlier (one week before upload day) and provide him required information (list of features and functionalities and access to demo).

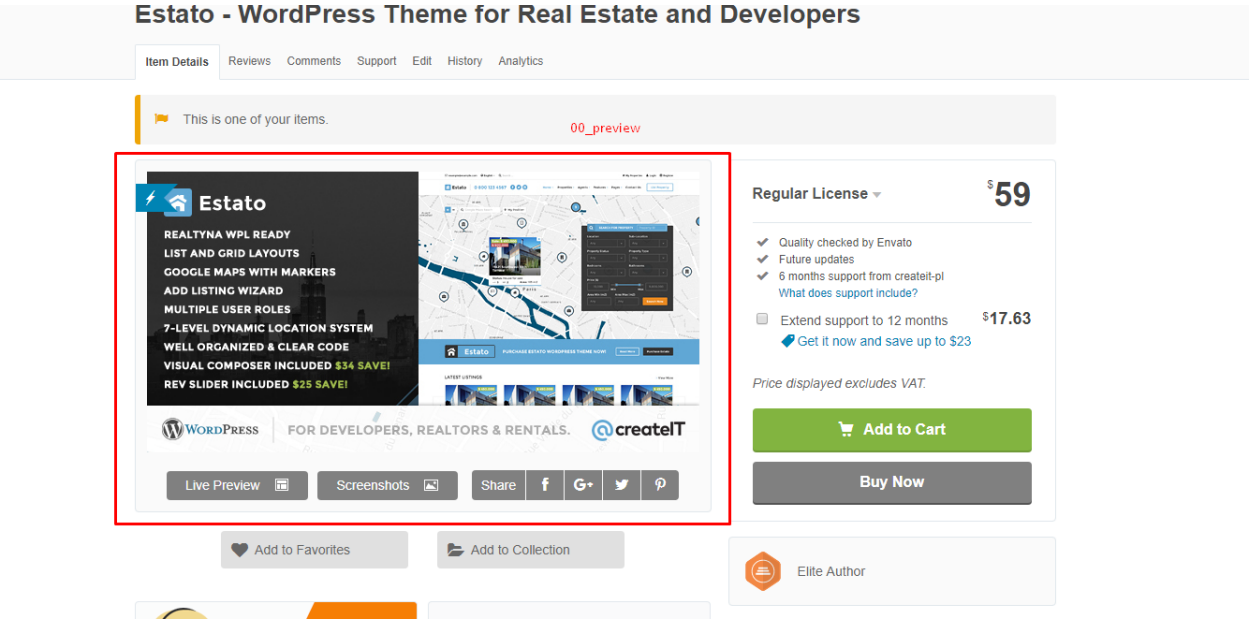
You will need at least the following materials and put them all inside `themeforest` folder in project:

- Thumb - JPEG or PNG 80x80px Thumbnail named *thumb_template.png* and if there is a version number on your thumb like so:



then upload the file without this number like so:

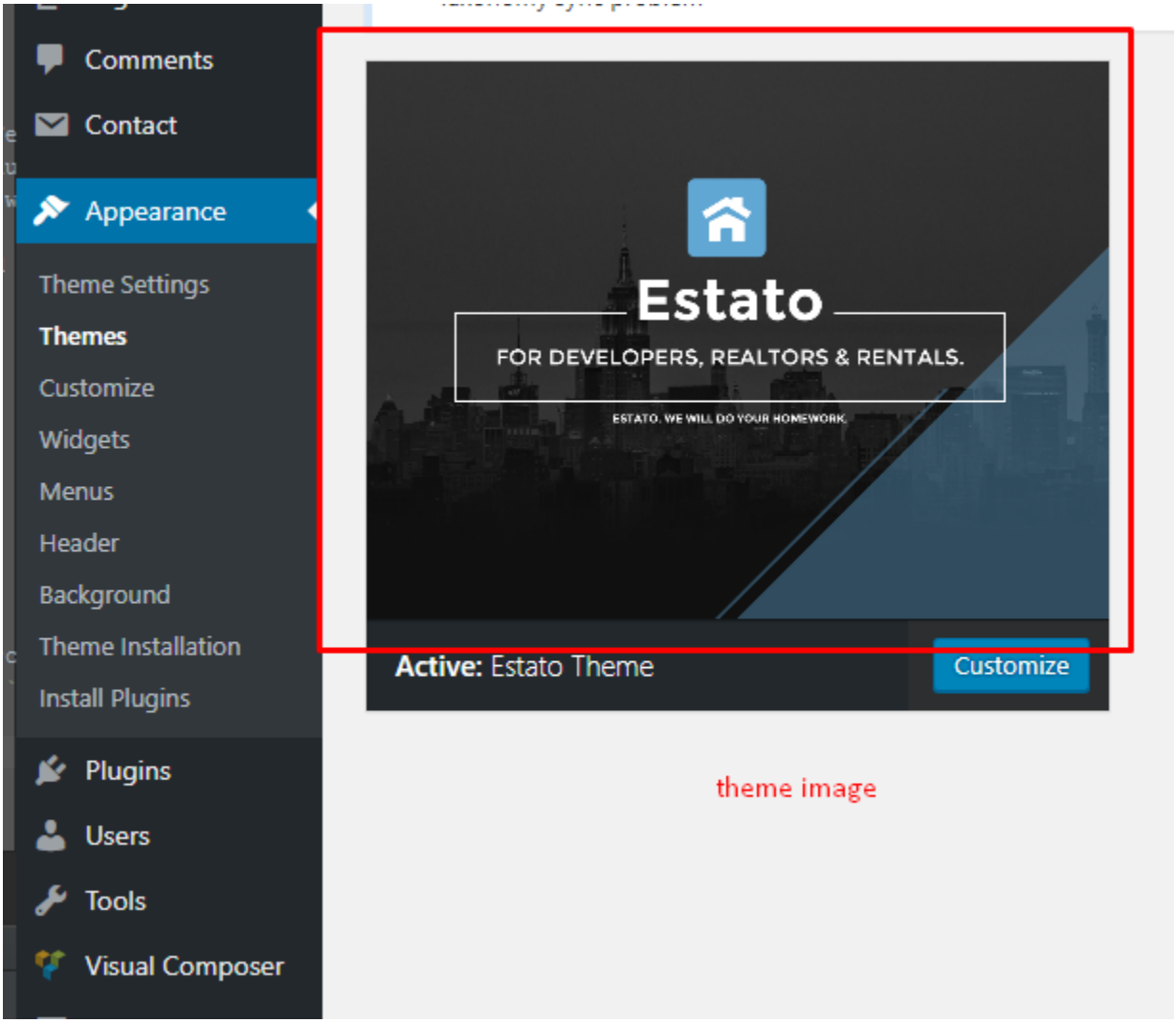
- Theme Preview - ZIP file of images (png/jpg) w/ optional text descriptions for display on the site,
- 00_preview - it should be the first image in the above ZIP package, containg: Name, Main Features, and Preview of the theme



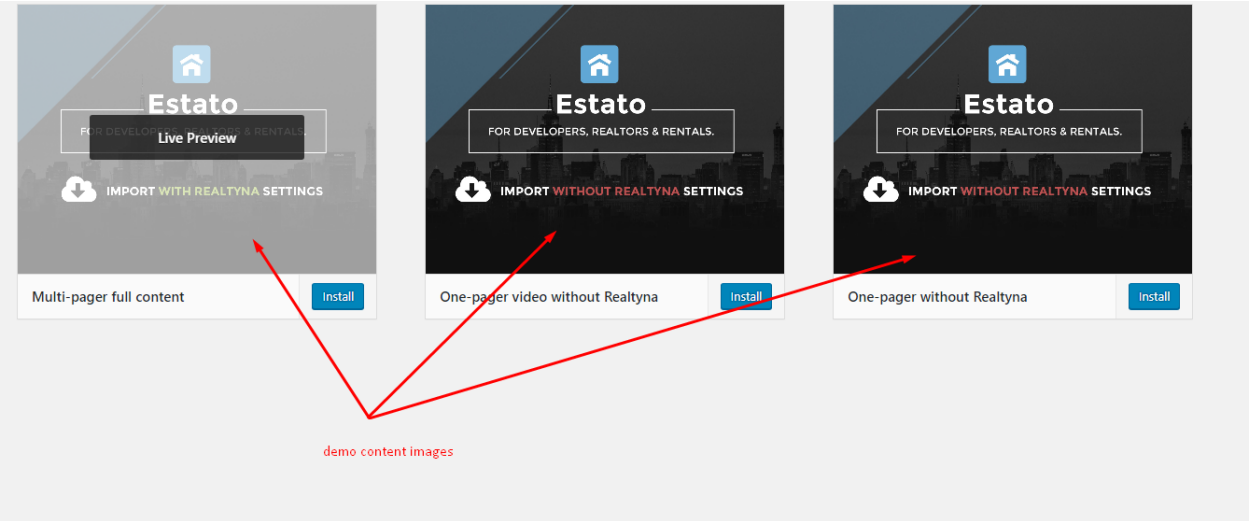
- Promo Description - create a list with all features that comes to your mind when you think about your project, all supported/included plugins, etc. and assign it to Art Director; Sample Promo Description: Estato Description
- Text Description - created in HTML and verified with envatitor full product description which will be visible on themeforest in items details. Example: here

(everything below the 'Add to Favourites' and 'Add to Collection' buttons)

- Theme Image



- Demo Import Image



Please create a separate task with promo materials list, sizes and required info (content for description, link to demo,

etc.) and assign it to Art Director at least one week before the Upload day.

Sample list of images to include:

```
- demo content luxury flavor 1200 x 900 px
- demo content paradise flavor 1200 x 900 px
- theme preview image 590 x 300 px
- theme preview thumb 80 x 80 px
- site icon 512 x 512 px
```

CHAPTER 4

Quality Assurance

4.1 Pre QA Check

4.1.1 Check list

Things to check before sending to QA

1. Run Theme Check
2. Escape and translate everything: Escaping
3. Prefix every global variable, function, class, database option, image size etc. Prefixes
4. Run Unit Test. Unit Test
5. Run Monster Widget. Monster Widget
6. Check for potential Reject Reasons
7. Check every page with Query Monitor to discover any errors in PHP, db queries etc.
8. Make sure all vendor plugins included in the theme (if any) are up to date.
9. (WooCommerce) Fix outdated WooCommerce templates

Developers Tools for testing

Query Monitor - check if any PHP errors, warnings are displayed and fix them,

4.1.2 Theme Check

Activate both of the plugins below and see if `Appearence -> Theme Check` shows any errors in the theme. If so, fix them all.

- Theme Check

- ThemeForest theme check

Note: In particular make sure the only text domain Theme check detects is `ct_theme`.

Hint: You can ignore messages about present zip files if you have vendor plugins attached.

4.2 Themeforest Check

4.2.1 Monster Widget

Developers Tools for testing

- Monster Widget - test all default WordPress widgets in all types of sidebars,
- WooCoomerce Monster Widget - if your theme support WooCoommerce

4.2.2 Unit Test

On a clean WordPress install, activate your theme and **without** importing any demo content, import test data as below link instructs.

1. Check if all contents display correctly as instructed in the link.
2. Then, import theme demo content, import test data one more time and check everything again.

Developers Tools for testing

- WPtest.io

4.2.3 Reject Reasons

Frequent reasons

Below, there are gathered frequent theme reject reasons. Please make sure none of theme are present in your theme.

1. **You should avoid and not to hardcode custom body_class(). Main reason for this is being able to remove the class when ne**
   What you can do - https://gist.github.com/kailoon/d4d22c9204909dff21eb ref: http://themeshaper.com/2014/11/20/mastering-the-post_class-function/ https://codex.wordpress.org/Function_Reference/body_class#Add_Classes_By_Filters
2. Custom widget areas must use the safety condition "is_active_sidebar" to ensure no naming conflict with other plugins.
3. **(Only for themes with WPML support) The file wpml-config.xml is missing.** http://wpml.org/documentation/support/achieving-wpml-compatibility-for-your-themes-and-plugins/
4. **Third party scripts/ styles don't need to be prefixed to avoid double loading.** ref: https://github.com/WordPress/twentysixteen/blob/master/functions.php#L250

5. **It's a very poor practice to deregister WordPress' built-in jQuery and load another specific version.** Please use the WordPress default javascript library. Use the jQuery noConflict mode to avoid issues — http://codex.wordpress.org/Function_Reference/wp_enqueue_script#jQuery_noConflict_wrappers
6. **Please use '-' instead of '_' or '.' for handlers and remove the extension.** For examples and there are more: http://envato.d.pr/PHo1V/2aQmS9y4
7. **Please use a unique prefix for all function names, custom images sizes, classes, CONSTANTS, hooks,** public/global variables, and database entries to avoid conflict issues with plugins and other themes. For example, **themename_** OR **frameworkname_** - Read more at http://themereview.co/prefix-all-the-things/ - You can have **frameworkname_** if you are using a framework while using **themename_** for your themes.
8. No page title by default, no top margin on content, http://envato.d.pr/8DeKF/2QTcjLug - Left margin is an issue too on some pages, http://envato.d.pr/bB6rr/3lx7HZPK
9. Comments / Trackbacks / pingbacks are not displaying: http://envato.d.pr/WZ9fB/5hnvD0Ar
10. Remove the static hard coded styles (style=) and move to classes/IDs and reference via CSS. http://envato.d.pr/N0hG8/4JGuX2Ri
11. http://envato.d.pr/bRfrC/1EIuWXbN is not needed unless you indicate support for these in your item attributes on ThemeForest
12. Missing site title upon intial installation: http://envato.d.pr/MI3VT
13. Please use get_template_directory when including/requiring files: http://envato.d.pr/E5IGl
14. **All dynamic data must be correctly escaped for the context where it is rendered. Please perform a global search** for "echo $" and you will see several issues. Ref: https://vip.wordpress.com/documentation/vip/best-practices/security/validating-sanitizing-escaping/ Example(s): http://envato.d.pr/fLPzH
15. Please disable force_activation: http://envato.d.pr/jmGZq
16. **Please enqueue Google fonts the correct way.** Here's an example: https://gist.github.com/richtabor/b85d317518b6273b4a88448a11ed20d3 and here's a good read: http://themeshaper.com/2014/08/13/how-to-add-google-fonts-to-wordpress-themes/ http://envato.d.pr/7XyrR
17. **Directory path should be get_template_directory() and not dirname(FILE). Use an appropriate path when** including theme files: • get_template_directory(): Returns the absolute template directory path. • get_template_directory_uri(): Returns the template directory URI. • get_stylesheet_directory(): Returns the absolute stylesheet directory path. • get_stylesheet_directory_uri(): Returns the stylesheet directory URI. Don't use: dirname(FILE), basename(FILE) or anything else not listed above. http://envato.d.pr/wNcgO
18. **Some of your files contain validation errors that will need to be fixed. Please be sure that all files validate** before resubmitting. You can validate HTML at http://validator.w3.org
19. **Themes should work out of the box without relying on plugins for basic functionality -** http://envato.d.pr/XBHs/4dQIjsRZ
20. **All theme text strings are to be translatable and properly escaped.** https://gist.github.com/kailoon/01fa8e95d2e910e666c6 example(s) from your code and there are more: http://envato.d.pr/NQsK/1GGgLoRq
21. **Scripts and styles should not be hardcoded anywhere in your theme or added any other way but with wp_enqueue_*** hook and to be added from the functions file. This includes custom JS/CSS. For inline styles use: https://developer.wordpress.org/reference/functions/wp_add_inline_style/ and for scripts https://developer.wordpress.org/reference/functions/wp_add_inline_script/
22. **Do not hardcode title. Title tag support now required.** See https://make.wordpress.org/themes/2015/08/25/title-tag-support-now-required/

23. **Data Validation issues have been found in your theme. All dynamic data must be correctly escaped for the context** where it is rendered. - All dynamic data must be escaped with esc_attr() before rendered in an html attribute. - Whenever you are rendering a url to the screen its value must be passed through esc_url() first. - If dynamic data is rendered inside an attribute that triggers a JavaScript event, it must be escaped with esc_js(). Please make sure you read these articles: https://make.wordpress.org/themes/tags/writing-secure-themes/ http://codex.wordpress.org/Data_Validation http://developer.wordpress.com/themes/escaping/ http://code.tutsplus.com/articles/data-sanitization-and-validation-with-wordpress–wp-25536 https://vip.wordpress.com/documentation/best-practices/security/validating-sanitizing-escaping/
24. **Please use wp_enqueue to load any external scripts/stylesheet - wp_enqueue_script / wp_enqueue_style -** http://codex.wordpress.org/Function_Reference/wp_enqueue_script#Functions For examples and there are more: http://envato.d.pr/GRzd/3JT5PgZJ
25. Please install WordPress plugins from the WordPress repo http://envato.d.pr/p6Ms/448azTjT
26. **http://envato.d.pr/bNkq/1jmnraYb** Always use esc_url when sanitizing URLs, including WordPress related, also, all home_url() must include a trailing slash such as home_url('/') See: https://codex.wordpress.org/Function_Reference/esc_url
27. **Some of your files contain validation errors that will need to be fixed. Please be sure that all files validate before resubmitt** You can validate HTML at http://validator.w3.org Example(s): https://validator.w3.org/nu/?doc=http%3A%2F%2Festato.wp.themeforest.createit.pl%2Fblog%2F
28. Make sure to display the page title: http://envato.d.pr/XbNR0/1jR02zSx
29. wp_footer should be placed immediately before body closing tag.
30. Provide unminified versions of JS files: http://envato.d.pr/Y4sW7d/2q0zcXbp
31. **Don't include libraries that are already pre-packaged with WordPress. Reference: https://developer.wordpress.org/refere** Example(s): http://envato.d.pr/ccWXeZ/4JjyUEc2
32. WooCommerce inner pages need attention.Example(s): https://envato.d.pr/86R3j3/nzsQqAq4MB https://envato.d.pr/wV1TYx/r1PzCSO3MWetc.
33. Post title displaying twice: https://envato.d.pr/mNsB9F/QljldgLXpb
34. Outdated woocommerce templates
35. Ensure all bundled plugins are the latest versions. Example, Visual Composer: http://envato.d.pr/VC77dO
36. Found a translation function that is missing a text-domain. Function **esc_html__**, with the arguments 'kids-theme'
37. wp_head should be placed immediately before head closing tag.
38. Don't combine libraries. This prevents deregistering and can cause double loading issues. Enqueue separately with proper handler instead.
39. Make sure all the WP default widgets display properly in all widgetized areas. You can check with monster widgets plugin. e.g. https://envato.d.pr/Psx477
40. The performance is poor https://envato.d.pr/ltMgUW
41. **Some of your files contain validation errors that will need to be fixed. Please be sure that all files validate before resubmitt** You can validate HTML at http://validator.w3.org
42. $_SERVER['REQUEST_URI'] or $_SERVER['SERVER_NAME'] is not safe or server reliable.

Other reasons

Below, you can find some of less frequent, project specific reject reasons. Look around to see what to expect from reviewers.

1. **Keyword stuffing titles is not permitted, you can use tags for that.** More info - https://help.market.envato.com/hc/en-us/articles/203039204-Tips-for-Excellent-Titles-Descriptions-and-Tags
2. **Would it be possible to get more out-of-the-box look here?**
   - http://envato.d.pr/39SvM/49ylx2vv
   - http://envato.d.pr/0dZso/1Lpdw2Ky
3. **This is bit crowded:** http://envato.d.pr/E0RzM/5Ayy0PjA
4. **A search UI should be improved, it's bit confusing:** http://envato.d.pr/bE8rm/Oj75BSAK
5. **A green button is more important here, therefore it should be first and read more second, it just makes better** sense: http://envato.d.pr/upME9/3oMiY76P
6. **Middle page feels a bit crowded:** http://envato.d.pr/tkNUX/ZPZfbScZ
7. **Not sure why the logo is in this area?** http://envato.d.pr/teLrj/2yfVxJIK
8. **Add more padding please:** http://envato.d.pr/a0T1J/1VG33sJ9
9. **Align the content vertically:** http://envato.d.pr/vruDb/4SZvzVjA
10. **Color contrast should be improved:** http://envato.d.pr/lzebG/G7VqBSOw
11. **Not enough of space:** http://envato.d.pr/gNZIZ/5y8k0D5a
12. **Align all items equally:** http://envato.d.pr/CMV0R/4lNeRb2a
13. **Pages like this feel bland:** http://envato.d.pr/OdZPY/RgphEvdE
14. Test how your theme design breakpoints across desktop, mobile, and tablet: http://design.google.com/resizer
15. By default your theme doesn't look presentable, this is what I get: http://envato.d.pr/6Cd0K/5R3qYGXw
16. No comments, http://envato.d.pr/oivXh/8GlEDwBw
17. Fatal error on category and author archive, http://envato.d.pr/q4IhV/FWOBkfRU
18. Tables require proper styling: http://envato.d.pr/Ouilr
19. Open mobile menu requires proper styling: http://envato.d.pr/bTTig
20. **I found validation errors with the Theme Check plugin. Don't forget to run Theme Check as well** https://wordpress.org/plugins/theme-check/ http://envato.d.pr/9MSTp
21. **'ct' is not an appropriate prefix. Please use a unique prefix for all function names, custom images sizes,** classes, constants, hooks, public/global variables, and database entries to avoid conflict issues with plugins and other themes. Two characters is not enough. Ref: http://themereview.co/prefix-all-the-things/ Example(s): http://envato.d.pr/Oqvxj
22. **Mobile styling needs attention/improvement in some areas. Please check/test your pages carefully and resolve all** issues. example: http://envato.d.pr/yhAR/2y1PckFF and more.
23. Your theme is pretty broken upon activation, even after plugins are installed - http://envato.d.pr/6BpU/4NJWxhFz
24. **How to test the blog/posts layout/functionality - Import the Theme Unit Test** [http://codex.wordpress.org/Theme_Unit_Test] file and make sure that: - Posts display correctly, with no apparent visual problems or errors. - Posts display in correct order. - Page navigation displays and works correctly. - As "sticky posts" are a core feature, the theme should style and display them appropriately. - Lack of body text should not

adversely impact the layout. - Theme must incorporate both the "Tag" and the "Category" taxonomies in some manner. - Floats are cleared properly for floated element (thumbnail image) at the end of the post content. Reference link: https://wpthemetestdata.wordpress.com/

25. **Make sure all the WP default widgets display properly in all widgetized areas. You can check with monster** widgets plugin.
26. **Please use a unique prefix (preferably the theme name or framework name) for all function names, custom images** sizes, classes, hooks, public/global variables, and database entries to avoid conflict issues with plugins and other themes. For example, **themename_** OR **frameworkname_**. - Read more at http://themereview.co/prefix-all-the-things/ - You can still be using **frameworkname_** if you are using a framework while using **themename_** for your themes. example(s) from your code and there are more: http://envato.d.pr/a9JH/1eupoJS0
27. **This http://envato.d.pr/1dm83/3rnqpcgj lead to this http://envato.d.pr/CBsQ/zqoO93Vq and** http://envato.d.pr/YFtf/1qaLZPxW
28. This page http://envato.d.pr/cPAF/WQDh9NRI is not necessary as you can easily write it down in the documentation.
29. Theme should work out of the box http://envato.d.pr/2vYK/1PyYwZGI
30. **You should avoid and not to hardcode custom body_class(). Main reason for this is being able to remove the class** when needed i.e. child themes. What you can do - https://gist.github.com/kailoon/d4d22c9204909dff21eb ref: http://themeshaper.com/2014/11/20/mastering-the-post_class-function/ https://codex.wordpress.org/Function_Reference/body_class#Add_Classes_By_Filters
31. **The localization file should be in English and delivered as .POT file. .POT will contain all translation strings.** .POT file name should match the themes-slug. Theme can include an actual translation files, but it should not add the en_US.mo or en_US.po because English already implies.
32. Directory path should be get_template_directory() and not dirname(FILE).
33. Don't leave unused code fragments as comments in the code. Please check all your files.
34. **Please use a unique prefix for all function names, custom images sizes, classes, CONSTANTS, hooks,** public/global variables, and database entries to avoid conflict issues with plugins and other themes. For example, **themename_** OR **frameworkname_** - Read more at http://themereview.co/prefix-all-the-things/ - You can have **frameworkname_** if you are using a framework while using **themename_** for your themes.
35. **It's a very poor practice to deregister WordPress' built-in jQuery and load another specific version.** Please use the WordPress default javascript library. Use the jQuery noConflict mode to avoid issues — http://codex.wordpress.org/Function_Reference/wp_enqueue_script#jQuery_noConflict_wrappers
36. **Broken logo:** http://envato.d.pr/cMmf/4aLb22ne
37. **Input placeholders will increase the UX:** http://www.w3schools.com/tags/att_input_placeholder.asp http://envato.d.pr/UQzi/3F4koerk
38. **Icons are a bit crowded and not prominent enough:** http://envato.d.pr/zXGP/5wC5axRf
39. **Increase the spacing:**
    - http://envato.d.pr/zAxb/5dWZDVJy
    - http://envato.d.pr/m5tF/qjNpHhTw
40. **Rename to "Latest" Blog:** http://envato.d.pr/BwyP/1pcOP3gr
41. **Not sure what the arrow was for and line-height needs to be increased:** http://envato.d.pr/XwDI/3LXhcFAu
42. **Make the space in between more consistent:** http://envato.d.pr/bysL/1D2a9Oo3

43. **The typographic scale needs balancing http://spencermortensen.com/articles/typographic-scale/** http://envato.d.pr/bysL/1D2a9Oo3
44. **It doesn't align vertically:** http://envato.d.pr/FiOM/3ZjyZuyB
45. **Post titles display twice:** http://envato.d.pr/N6ip/DYpuIavZ
46. **Looks incomplete:** http://envato.d.pr/Zi6r/2uLZn1jx
47. A demo (Presentation) needs to be complete.
48. Test how your theme design breakpoints across desktop, mobile, and tablet: http://design.google.com/resizer/
49. Can't close the side menu: http://envato.d.pr/lBo1Ap/5v6pw1sP
50. Comments area design could be improved: http://envato.d.pr/LrYeMO/ZGH7ZQgr
51. Avoid all-caps: https://envato.d.pr/ajh467/llLuYkeLwW
52. This section looks really odd https://envato.d.pr/qFtszT
53. There are a few inconsistent spacing/padding as well as alignment issues throughout the design. I'd like you to go over it with a fine toothed comb and weed out these basic design issues. For examples and there are more: https://envato.d.pr/1wiqaq

4.3 QA Notes

During the development process each task should be tested separately by you and assigned to QA person. Every time when single task is finished you need to move it to QA task list and assign to designated QA team member. When everything is working properly your task will be moved to "Resolved/Done" task list and left unnassigned. If something needs to be corrected or changed you will get your task back with the feedback info what needs to be done to consider the task as Done.

Once all components are done and tested it's a high time to test everything one more time as all. Please note that it's your responsibility to check that first and only after your internal testing you can ask the QA team for complex testing.

There are few things which you should pay attention to:

- Responsive view - please make sure that you have checked all pages with mobile device emulator in your browser or with tools like Resizer
- Compatibility with PSD project - please compare page by page, element by element compatibility of your work with original PSD design
- Performance - run the Google PageSpeed Insights and implement all required optimization fixes
- Demo Page Flow - make sure that all links, buttons, CTA and navigation items are adressed correctly and will direct user somewhere, where he will be able to take another action and go to another page or element. Don't leave "#" as a link NOWHERE. Always add link to other subpage, section or element within the demo page. All buttons with "BUY NOW", "GET THE THEME", "PURCHASE THE THEME NOW" you need to add 'Optimus Buy Link", which looks like the following: http://optimus-prime.createit.pl/OPTIMUS-THEME-SLUG/buy 'OPTIMUS-THEME-SLUG' can be provided by Team Manager or Support.

4.4 QA Package

When testing is done and theme zip package is created you need to verify whether it works correctly.

1. First, please setup test environment, which is a separate, clean, wordpress site.

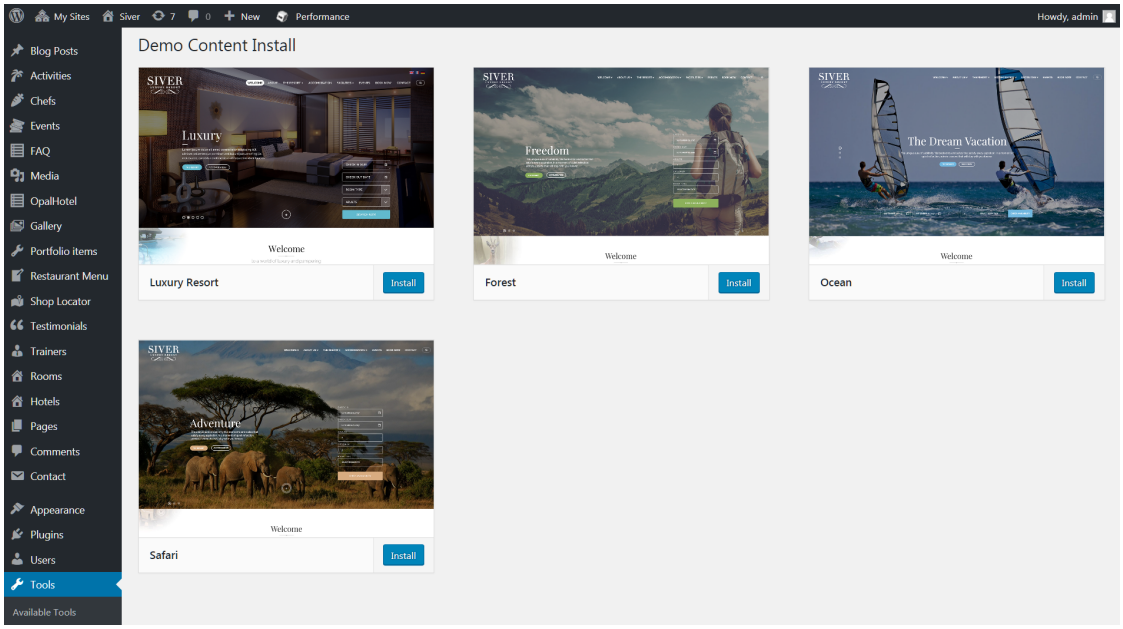
2. Then install the theme package via admin panel by uploading it in *Appearance* > *Themes*. Do not checkout the theme from GIT repository for testing purposes, because it won't verify whether the theme can be installed via admin panel, which is the most common way of installing the theme by customers.
3. Follow the documentation and displayed informations about installing recommended plugins, etc.
4. Import the Demo Content.
5. Test frontend:
    - Check if all images are imported correctly (if we have any 404 in Console - this needs to be fixed)
    - Check if all styles are loaded correctly and your test site looks exactly the same as demo.
    - Check if navigation menu is setup and displayed correctly.
    - Check responsiveness.
    - Check if WooCommerce/Contact Form 7/any other plugins related components are working properly.
    - Check if you are able to install and activate all plugins attached to the package (make sure that you don't have any of these plugins installed on your test environment first)

CHAPTER 5

Upload & Reject

5.1 Demo Content

When users install a theme, they should have the ability to install a demo content (settings, pages, images, colors) which comes with the theme. Below, there is a screenshot of demo content install screen from Siver theme which uses framework built-in extensions “Backups” and “Demo Content”.



In this theme, there are 4 demo contents to choose from. Usually, one demo content is put inside a package (local) and the others are downloaded from external server (remote) for the sake of the package size which should be less than 100MB.

5.1.1 Local demo content

Local demo content resides inside a theme package, but not in theme files on git repository. In this section, there will be two topics:

1. How to set local demo content in a package
2. How to move content from p(h)inky to demo site

Set local demo content

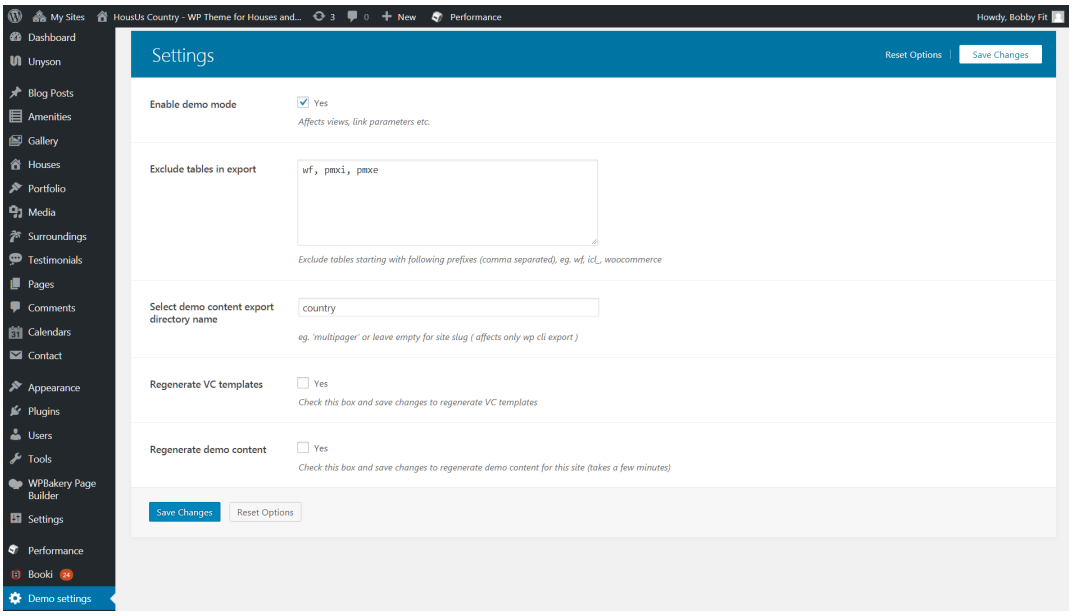
To set a local demo content:

1. Choose a name for your demo content, eg. forest-onepager, example-multipager, default
2. Create a folder inside *theme/demo-content* directory, eg. *wp-content/themes/projectname/theme/demo-content/example-onepager*
3. There, create (or copy from boilerplate) files: *manifest.php*, *screenshot.png*. You should have this structure now:

```
wp-content/
├-themes/
  ├-projectname/
    ├-theme/
    │  |--demo-content/                # Put all demo content manifest and␣
↪screenshot. The content itself should be autogenerated.
    │  │  |--example-onepager/         # Local demo content directory, demo files␣
↪will be generated here
    │  │  │  |--manifest.php           # Demo content name, image, preview link
    │  │  │  |--screenshot.png         # Thumbnail made by graphic designer
```

4. Configure demo content name and preview link in *manifest.php*, as in the example below:

5. Gid add and commit files. That's all. Don't put content files here. The build will add them to the package.
6. **If at this point you don't have content on demo site yet, first proceed with "Move content from p(h)inky to demo site" section below, then go back here**
7. Go to your theme demo page admin (if it's a multisite, choose the site which content you wish to export)
8. Install and activate your theme demo plugin (if not already)
9. Go to *Demo settings* menu screen. It should look like this:

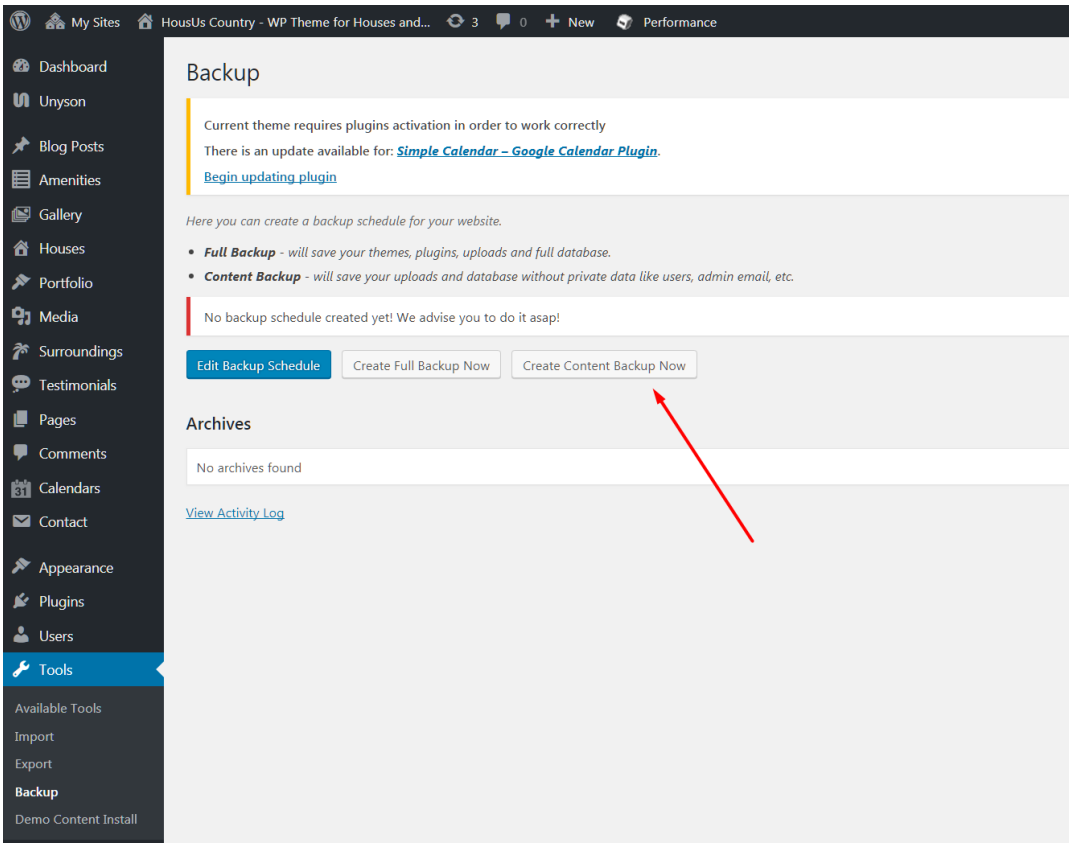


10. Enter your *demo content export directory name* and press save (you can leave the other options intact)
11. That's all. If your build is properly configured, the demo content will be regenerated with each package build.

Move content from p(h)inky to demo site

If your demo site has yet no content, you will need to export it from your p(h)inky and send to demo via git. Follow these steps:

1. Set local demo content as described above (the one you wish to move to demo)
2. Go to your p(h)inky admin (to the site where you have content you wish to move)
3. Make sure you have enabled "Backups" extension in Unyson menu
4. Go to to Tools -> Backup and press "Create Content Backup Now" as in the screenshot



5. Once it finishes exporting, download the backup zip
6. Extract contents of the zip to your local demo content folder, so you have this structure

Structure: .. code-block:

```
wp-content/
        ├-themes/
            ├-projectname/
                ├-theme/
                |   |--demo-content/
                |   |   |--example-onepager/
                |   |   |   |--f/                       # Demo content images
                |   |   |   |--database.json            # Demo content settings
                |   |   |   |--manifest.php
                |   |   |   |--screenshot.png
```

Next:

1. Git add new files and commit them.
2. Ask the project manager to update the demo (and not to generate the package)
3. Choose your newly added demo content to install (wp-admin -> Tools -> Demo Content Install)
4. After several minutes it should be done
5. Delete demo content files (*database.json* and *f* folder) from your theme files and commit (they are meant to be autogenerated)
6. That's all

5.1.2 Remote demo content

For sake of package size, most of demo contents are put remotely on a theme demo site server.

First, you need to set up remote demo content download path.

1. Edit file *wp-content/plugins/theme-plugin/extensions/backups/config.php*
2. Find the variable $cfg[‘remote_demo_url’] (or create it at the end) and set a proper path, ie.

```
/** Remote demo content download link */
$cfg['remote_demo_url'] = 'http://THEMEURL/wp-content/plugins/THEME-DEMO-
↪PLUGIN/vendor/bumblebee/theme-demo-plugin-core/extensions/ct-demo-
↪content/includes/remote-demo-content/download.php';

/** For example
$cfg['remote_demo_url'] = 'http://housus.themeplayers.net/wp-content/
↪plugins/housus-demo-plugin/vendor/bumblebee/theme-demo-plugin-core/
↪extensions/ct-demo-content/includes/remote-demo-content/download.php';
*/
```

3. Edit file *wp-content/plugins/theme-demo-plugin/config.php*.
4. Find the variable $cfg[‘remote_demo_content_directory’] and set proper path, ie.

```
/** Remote demo content download link */
$cfg['remote_demo_content_directory'] = '/themes/PROJECTNAME/themeforest/
↪demo-content';

/** For example
$cfg['remote_demo_content_directory'] = '/themes/siver/themeforest/demo-
↪content';
*/
```

5. Commit.

Then, create a remote demo content:

1. Set local demo content as described above
2. Change the name of *manifest.php* to *remote.php* (indicating this demo content will be a remote one)
3. Add this demo content to remote demo contents list in build.properties file
4. Commit
5. That’s all

5.2 Upload Process

When your theme is complete and demo page is working, there’s one more thing left to do before the upload. That is to set up a zip package containing theme contents with proper text domain, compiled styles, generated demo content, promo materials, documentation, languages etc.

5.2.1 Building a package

Packages are automatically built by Phing tasks. All you need to do is configure them in build.xml file. Then, ask a project manager to build a package.

Check project structure

Please follow the boilerplate structure: Theme structure

In particular, make sure you do have the following directories:

```
wp-content/
├-themes/
  ├-projectname/
    ├-build/                          # There should be build.xml and build.
↪properties files already there
    ├-docs/                           # Put pdf documentation here (or␣
↪create a .gitkeep file if needed)
    ├-theme/
    │  |--vendor/                     # Commerial plugins zip files (if␣
↪required by the theme)
    │  |--languages/                  # Leave empty for autogenerated .pot␣
↪file (create a .gitkeep file if needed)
    │  |--demo-content/               # Put all demo content manifest and␣
↪screenshot. The content itself should be autogenerated.
    │  │  |--example-multipager/      # Local demo content directory, demo␣
↪files will be generated here
    │  │  │  |--manifest.php          # Demo content name, image, preview␣
↪link
    │  │  │  |--screenshot.png        # Screenshot
    │  │  |--example-remote-content/  # Remote demo content directory,␣
↪downloaded outside of the package
    │  │  │  |--remote.php            # Demo content name, image, preview␣
↪link (same as manifest.php)
    │  │  │  |--screenshot.png        # Screenshot
    │  │  |--example-onepager/
    │  │  |--...
```

Basic structure can be downloaded here: Theme boilerplate

Set up paths in build.properties

In `build/build.properties` file set up paths according to your project name and structure. Below, you can find an example from boilerplate theme with comments:

```
<!-- Base variables -->

# base directory, path relative to 'build' directory
base.dir = ./..

# project name = project directory = final text domain (no special chars are allowed)
base.projectName = boilerplatetheme

# project plugin slug = plugin directory
base.pluginName = theme-plugin

# project demo plugin slug = demo plugin directory
base.demoPluginName = theme-demo-plugin

# path to theme files
base.html = ${base.dir}/theme

# path to plugin files
```

```
base.pluginSrc = ${base.dir}/../../plugins/${base.pluginName}

# destination path to plugin zip
base.pluginDst = ${base.html}/vendor/${base.pluginName}

# path to demo plugin files
base.demoPluginSrc = ${base.dir}/../../plugins/${base.demoPluginName}

# theme demo website (for wp-cli tasks)
base.url = http://boilerplate.themeplayers.net

# theme flavors slugs (for wp-cli tasks)
base.flavors = flavor1, flavor2, flavor3

<!-- SCSS Compiler related -->

# path to compiled main css
base.css.path = assets/css/style.css

# path to main scss
base.scss.path = assets/sass/style.scss

# final css format
build.scss.formatter = compressed

<!-- The path to the publish directory. -->

# publisher's name (no need to change)
build.publishName = themeforest

# directory to hold all packages (no need to change)
build.dir = ${base.dir}/${build.publishName}

# temp directory for packing (no need to change)
build.tmp = ${build.dir}/tmp

# theme package name (no need to change)
build.zipNameFile = ${base.projectName}.zip

# theme package destination path (no need to change)
build.zipName = ${build.tmp}/${build.zipNameFile}

# theme package with promo materials destination path (no need to change)
build.zipNameFull = ${build.dir}/${base.projectName}_all.zip

# packing directory (no need to change)
build.html = ${build.tmp}/${base.projectName}

# theme flavors which will be available as remote demo content (for wp-cli tasks)
build.remotes = flavor1, flavor2

# remote demo content destination directory (for wp-cli tasks)
build.remotesDestination = ${build.dir}/demo-content

<!-- Documentation -->

# path to docs html directory on demo (no need to change)
base.doc = ${base.dir}/../../../documentation
```

```
# path to docs pdf inside package (no need to change)
build.doc = ${build.tmp}/Documentation

# slug of the docs file (without an extension)
build.docSlug = boilerplate

# release of the docs file (eg. latest, master)
build.docRelease = latest

# documentation pdf download link
build.docUrlPdf = https://media.readthedocs.org/pdf/${build.docSlug}/${build.
↪docRelease}/${build.docSlug}.pdf

# documentation html download link
build.docUrlHtml = https://media.readthedocs.org/htmlzip/${build.docSlug}/${build.
↪docRelease}/${build.docSlug}.zip
```

Set up tasks in build.xml

Once properties properly set, build.xml tasks need only little changes. Here are some sections higlighted, and below you can find a whole file example

Tasks sets

These are main two task sets. One for creating a package, and another one for updating demo website. Here, you can add or remove tasks defined below in `build.xml`, however the default set up should be enough

```
<target name="envato"
        depends="base.cleanup, base.update, build.delete, build.plugin, build.
↪demoPlugin, build.createDir, build.vctemplates, build.generateDemoContent, build.
↪deleteThumbnails, build.moveRemotes, copy.all, build.removeDemo, build.
↪changeTextDomain, build.pot, build.doc, build.pack, build.cleanup">
    <echo message="Envato product version completed."/>
</target>

<target name="demo"
        depends="base.update, build.plugin, build.demoPlugin, build.vctemplates">
    <echo message="Envato demo completed."/>
</target>
```

build.scss

`build.scss` task calls `ct_fw_ext_sass_compiler_cli_compile_save` function located in demo plugin. Make sure you have configured SCSS compiler in your project. Or if you don't use it, remove the task from task sets above.

```
<target name="build.scss">
    <echo>Compiling SCSS to CSS...</echo>
    <exec
            command="wp eval 'ct_fw_ext_sass_compiler_cli_compile_save();'"
    />
</target>
```

build.generateDemoContent

`build.generateDemoContent` task calls `ct_fw_ext_backups_do_cli_backup` function located in demo plugin. The –url parameter needs to point to a particular site (if multisite install) to generate content from, so you will need to modify the base.flavors variable in build.properties. Read more about WP-CLI: WP-CLI global parameters

Also make sure Demo Plugin settings (demo content directory in particular) are properly set on demo website.

```
<target name="build.generateDemoContent">
    <echo>Generating demo content...</echo>
    <exec
            command="wp eval --user=1 --url=${base.url} 'fw_ext_ct_demo_content_do_
↪cli_backup();'"
            outputProperty="execOutput"
    />
    <echo msg="Generated output: ${execOutput}"/>
    <foreach list="${base.flavors}" target="build.flavorDemoContent" param="flavorSlug
↪" />
</target>

<target name="build.flavorDemoContent">
    <echo msg="Generating demo content for flavor: ${flavorSlug}" />
    <exec
            command="wp eval --user=1 --url=${base.url}/${flavorSlug} 'fw_ext_ct_demo_
↪content_do_cli_backup();'"
            outputProperty="execOutput"
    />
    <echo msg="Generated output: ${execOutput}"/>
</target>
```

An example build.xml file

```
<?xml version="1.0"?>

<project name="boilerplate" default="envato">

<import file="lib/build.common.xml"/>

<property file="build.properties"/>
<property name="build.jsPath" value="${build.html}/${base.js.path}"/>
<property name="build.cssPath" value="${build.html}/${base.css.path}"/>

<taskdef name="lineend" classname="lib.converter.ctLineEndingConverter"/>
<taskdef name="jsobfuscate" classname="lib.obfuscator.JsObfuscatorTask"/>
<taskdef name="frameadd" classname="lib.frame.themewoodmen.
↪ctThemeforestThemewoodmenFrame"/>
<taskdef name="switcher" classname="lib.switcher.ctColorSwitcherTask"/>

<target name="envato"
        depends="base.cleanup, base.update, build.delete, build.plugin, build.
↪demoPlugin, build.createDir, build.vctemplates, build.generateDemoContent, build.
↪deleteThumbnails, build.moveRemotes, copy.all, build.removeDemo, build.
↪changeTextDomain, build.pot, build.doc, build.pack, build.cleanup">
    <echo message="Envato product version completed."/>
</target>
```

```
<target name="demo"
        depends="base.update, build.plugin, build.demoPlugin, build.vctemplates">
    <echo message="Envato demo completed."/>
</target>

<target name="build.plugin">
    <echo>git reset --hard</echo>
    <exec command="git reset --hard" dir="${base.pluginSrc}"/>
    <echo>git pull</echo>
    <exec command="git pull" dir="${base.pluginSrc}"/>
    <echo>composer update</echo>
    <exec command="composer update" dir="${base.pluginSrc}"/>
    <delete file="${base.html}/vendor/${base.pluginName}.zip"/>
    <delete file="${base.pluginDst}/${base.pluginName}.zip"/>
    <delete dir="${base.pluginDst}"/>
    <copy todir="${base.pluginDst}/${base.pluginName}">
        <fileset dir="${base.pluginSrc}">
            <exclude name="**/*.svn/"/>
            <exclude name="**/*.git/"/>
            <exclude name="**/*.git*"/>
            <exclude name="**/*.idea/"/>
            <exclude name="**/*.htaccess"/>
            <exclude name="**/composer.*"/>
        </fileset>
    </copy>
    <zip destfile="${base.pluginDst}/${base.pluginName}.zip" description="dump zip to␣
↪tmp">
        <fileset dir="${base.pluginDst}"/>
    </zip>
    <delete dir="${base.pluginDst}/${base.pluginName}"/>
</target>

<target name="build.demoPlugin">
    <echo>git reset --hard</echo>
    <exec command="git reset --hard" dir="${base.demoPluginSrc}"/>
    <echo>git pull</echo>
    <exec command="git pull" dir="${base.demoPluginSrc}"/>
</target>

<target name="build.update">
    <echo>git pull ${base.dir}</echo>
    <exec command="git reset --hard" dir="${base.dir}" checkreturn="true"/>
    <exec command="git pull" dir="${base.dir}"/>
</target>

<target name="build.removeDemo">
    <reflexive description="Strip demo content if required">
        <fileset dir="${build.html}" includes="**/*.php"/>
        <filterchain description="cleans demo code">
            <replaceregexp>
                <regexp pattern="//\s*demo\s*[\s\S]+?//\s*end demo" replace=""/>
            </replaceregexp>
        </filterchain>
    </reflexive>
</target>

<target name="build.changeTextDomain">
```

```
    <reflexive description="Change text domain to project specific">
        <fileset dir="${build.html}" includes="**/*.php"/>
        <filterchain description="switches text domain">
            <replaceregexp>
                <regexp pattern="ct_theme" replace="${base.projectName}"/>
            </replaceregexp>
        </filterchain>
    </reflexive>
</target>

<target name="build.eol">
    <lineend path="${build.html}/${base.css.path}/style.css"/>
</target>

<target name="build.scss">
    <echo>Compiling SCSS to CSS...</echo>
    <exec
            command="wp eval 'fw_ext_ct_demo_content_cli_compile_save();'"
            outputProperty="execOutput"
    />
    <echo msg="Generated output: ${execOutput}"/>
    <foreach list="${base.flavors}" target="build.flavorScss" param="flavorSlug" />
</target>

<target name="build.flavorScss">
    <echo msg="Compiling SCSS for flavor: ${flavorSlug}" />
    <exec
            command="wp eval --user=1 --url=${base.url}/${flavorSlug} 'fw_ext_ct_demo_
↪content_cli_compile_save();'"
            outputProperty="execOutput"
    />
    <echo msg="Generated output: ${execOutput}"/>
</target>

<target name="build.generateDemoContent">
    <echo>Generating demo content...</echo>
    <exec
            command="wp eval --user=1 --url=${base.url} 'fw_ext_ct_demo_content_do_
↪cli_backup();'"
            outputProperty="execOutput"
    />
    <echo msg="Generated output: ${execOutput}"/>
    <foreach list="${base.flavors}" target="build.flavorDemoContent" param="flavorSlug
↪" />
</target>

<target name="build.flavorDemoContent">
    <echo msg="Generating demo content for flavor: ${flavorSlug}" />
    <exec
            command="wp eval --user=1 --url=${base.url}/${flavorSlug} 'fw_ext_ct_demo_
↪content_do_cli_backup();'"
            outputProperty="execOutput"
    />
    <echo msg="Generated output: ${execOutput}"/>
</target>

<target name="build.moveRemotes">
    <echo msg="Moving remotes: ${build.remotes} to destination: ${build.
↪remotesDestination}" />
```

```
    <foreach list="${build.remotes}" target="build.moveSingleRemote" param="flavorSlug
↪" />
</target>

<target name="build.moveSingleRemote">
    <zip destfile="${base.html}/demo-content/${flavorSlug}/${flavorSlug}.zip"␣
↪description="pack flavor content">
        <fileset dir="${base.html}/demo-content/${flavorSlug}"/>
    </zip>
    <copy todir="${build.remotesDestination}/${flavorSlug}">
        <fileset dir="${base.html}/demo-content/${flavorSlug}">
            <include name="*.zip" />
            <include name="*.php" />
            <include name="*.png" />
        </fileset>
    </copy>
    <delete includeemptydirs="true" description="remove remote demo files from theme␣
↪but leave manifest and screenshot">
        <fileset dir="${base.html}/demo-content/${flavorSlug}">
            <exclude name="remote.php" />
            <exclude name="manifest.php" />
            <exclude name="screenshot.*" />
        </fileset>
    </delete>
</target>

<target name="js.obfuscate">
    <jsobfuscate targetdir="${build.html}/${base.js.path}">
        <fileset dir="${build.html}/${base.js.path}">
            <include name="ct*.js"/>
        </fileset>
    </jsobfuscate>
</target>

<target name="base.cleanup">
    <echo>Deleting vendor zip files...</echo>
    <delete>
        <fileset dir="${base.html}/vendor">
            <include name="*.zip"/> <!-- delete old style vendor files-->
        </fileset>
    </delete>
    <echo>Deleting wrong dir for demo content...</echo>
    <delete dir="${base.html}/demo-content/test-dir" />
</target>

<!-- Delete the Publish directory to start from scratch -->
<target name="build.delete">
    <echo>build.delete</echo>
    <delete file="${build.zipNameFull}"/>
    <delete file="${build.dir}/${build.zipNameFile}"/>
    <delete dir="${build.tmp}"/>
</target>
<target name="build.cleanup">
    <delete dir="${build.tmp}"/>
</target>

<!-- Create the Build Directory -->
<target name="build.createDir">
```

```
    <mkdir dir="${build.tmp}"/>
</target>

<target name="copy.all">
    <!--Main HTML-->
    <copy todir="${build.html}">
        <fileset dir="${base.html}">
            <exclude name="npm/"/>
            <exclude name="**/*.svn/"/>
            <exclude name="**/*.git/"/>
            <exclude name="**/*.git*"/>
            <exclude name="**/*.idea/"/>
            <exclude name="**/*.htaccess"/>
            <exclude name="**/composer.*"/>
        </fileset>
    </copy>
</target>

<target name="build.pack">
    <copy todir="${build.tmp}/packing/${base.projectName}">
        <fileset dir="${build.html}"/>
    </copy>
    <zip destfile="${build.zipName}" description="dump zip to tmp">
        <fileset dir="${build.tmp}/packing/"/>
    </zip>
    <delete dir="${build.tmp}/${base.projectName}" description="remove theme"/>
    <delete dir="${build.tmp}/packing" description="remove temp zip dir"/>
    <zip destfile="${build.zipNameFull}" description="packs everything (with theme␣
↪zip)">
        <fileset dir="${build.tmp}"/>
    </zip>
    <copy todir="${build.dir}">
        <fileset dir="${build.tmp}">
            <include name="${build.zipNameFile}"/>
        </fileset>
    </copy>
</target>

<target name="build.pot" description="generate po i18n file">

    <delete includeemptydirs="false">
        <fileset dir="${build.html}/languages">
        </fileset>
    </delete>

    <echo message="php lib/pot/makepot.php wp-theme ${build.html} ${build.html}/
↪languages/${base.projectName}.pot"/>
    <exec command="php lib/pot/makepot.php wp-theme ${build.html} ${build.html}/
↪languages/${base.projectName}.pot"
          description="generate po file. LANG dir must exist"/>
</target>

<target name="build.vctemplates" description="re generate vc templates for every page
↪">

    <echo>Generating VC templates...</echo>
    <exec
            command="wp eval --user=1 --url=${base.url} 'fw_ext_ct_demo_content_
↪update_vc_templates();'"
```

```
    />
    <foreach list="${base.flavors}" target="build.flavorVctemplates" param="flavorSlug
↪" />

</target>

<target name="build.flavorVctemplates">
    <echo msg="Generating VC templates for flavor: ${flavorSlug}" />
    <exec
            command="wp eval --user=1 --url=${base.url}/${flavorSlug} 'fw_ext_ct_demo_
↪content_update_vc_templates();'"
    />
</target>

<target name="build.deleteThumbnails" description="delete all custom image sizes in␣
↪demo content">

    <exec
            command="find ${base.html}/demo-content -type f -regex '.+-[0-9]+x[0-9]+\.
↪jpg' -o -regex '.+-[0-9]+x[0-9]+\.jpeg' -o -regex '.+-[0-9]+x[0-9]+\.png' -o -regex
↪'.+-[0-9]+x[0-9]+\.gif' | wc -l"
            outputProperty="filesToDelete"
    />
    <echo msg="Deleting ${filesToDelete} thumbnails..."/>

    <exec
            command="find ${base.html}/demo-content -type f -regex '.+-[0-9]+x[0-9]+\.
↪jpg' -delete"
    />

    <exec
            command="find ${base.html}/demo-content -type f -regex '.+-[0-9]+x[0-9]+\.
↪jpeg' -delete"
    />

    <exec
            command="find ${base.html}/demo-content -type f -regex '.+-[0-9]+x[0-9]+\.
↪png' -delete"
    />

    <exec
            command="find ${base.html}/demo-content -type f -regex '.+-[0-9]+x[0-9]+\.
↪gif' -delete"
    />

</target>

<target name="build.doc">

    <!-- pdf docs for the package -->
    <mkdir dir="${build.doc}"/>
    <httpget url="${build.docUrlPdf}" dir="${build.doc}" filename="Documentation.pdf"/
↪>

    <!-- html docs for the demo -->
    <mkdir dir="${base.doc}"/>
    <httpget url="${build.docUrlHtml}" dir="${base.doc}" filename="${build.docSlug}-$
↪{build.docRelease}.zip"
```

```
            description="download docs"/>
    <unzip todir="${base.doc}" file="${base.doc}/${build.docSlug}-${build.docRelease}.
↪zip"
           description="unzip docs"/>
    <copy todir="${base.doc}" overwrite="true">
        <fileset dir="${base.doc}/${build.docSlug}-${build.docRelease}">
            <include name="**/*.*"/>
        </fileset>
    </copy>
    <delete dir="${base.doc}/${build.docSlug}-${build.docRelease}" />
    <delete file="${base.doc}/${build.docSlug}-${build.docRelease}.zip" description=
↪"deleting doc zip"/>

</target>
```

</project>

Generate Package

To locally test the package build, download phing tool, and run `phing` command in build directory. Some task won't work here, eg. wp cli based tasks, but the zip file should be built inside *themeforest* directory. When succeeded, commit and push changes in build.xml. Ask a project manager to update demo website and generate the package.

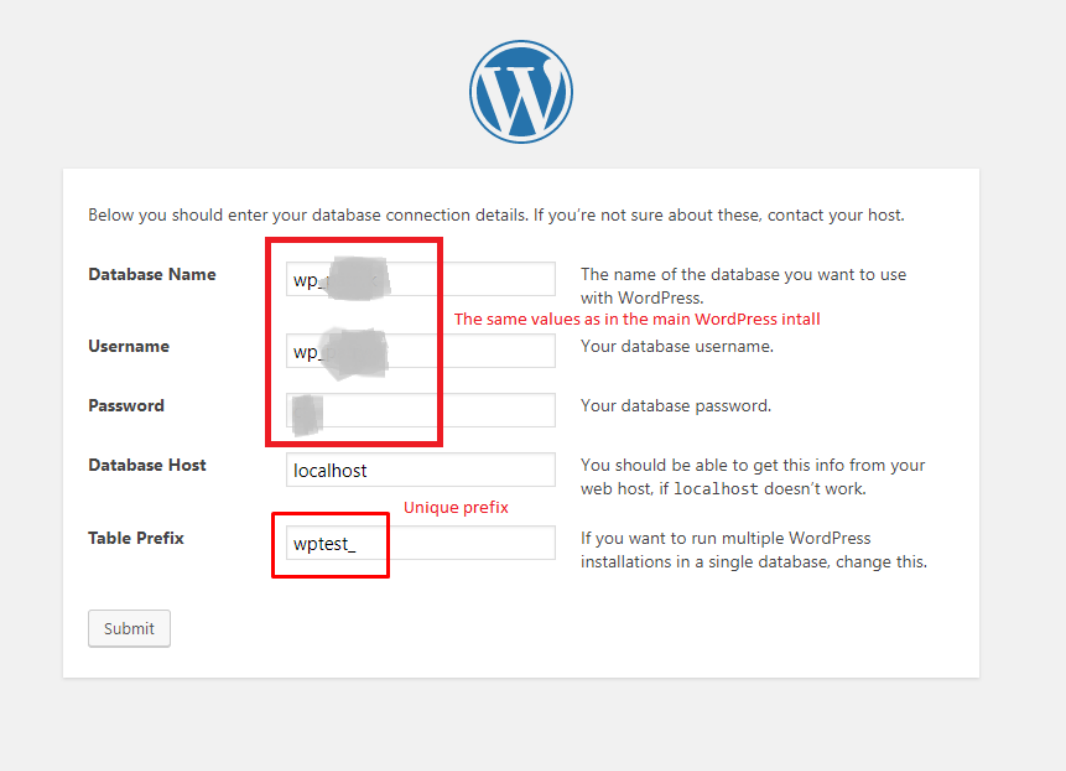
Note: In order for changes in build.xml to take place you may need to run build twice (build.xml is pulled from git after tasks run init)

5.2.2 Package QA

Test Wordpress install

If you don't have it already, create a fresh wordpress install just for testing packages. It can be a single site installation in a subdirectory of your current wordpress.

1. Download wordpress from wordpress.org/download
2. Unpack contents to a new subdirectory of your phinky main wordpress directory, eg. phinky.createit/www/ctXX/wp/**wptest**
3. Run wordpress install. See screenshot below. Important: select unique table prefix, eg. **wptest_**



4. Log in to your new install url: http://ctXX.phinky.createit/wptest/

Package install

1. Make sure you have a fresh wordpress test installation. If not, reset tables using WordPress Reset plugin, delete all plugins, themes, uploads
2. Install the package using wp-admin theme install button "Upload theme"
3. Check everything (Theme Check, Unit Test, general display etc.) on Check list

5.2.3 Upload

If everything checks, let know the project manager the package is ready to be uploaded for a review.

5.3 Rejects Procedure

You can assume even after thorough QA process your theme will be rejected by reviewers. It is necessary to fix all errors according to their instructions (even when they don't make sense) and reupload the fixed theme.

Note: You can find example reject reasons here: Reject Reasons

1. Fix everything according to reviewers' instructions
2. Describe your fixes shortly and attach screenshots of fixes for each point
3. Generate package

4. Install the package on a clean WordPress site using wp-admin theme install feature
5. Check if fixes work
6. Check if demo website is still working properly
7. Check if demo test site is displaying properly (see creating test site)
8. Check everything else (Theme Check, code errors, general display etc.) on Check list
9. Reupload

Important: Every reject task is high priority and should be closed the same day it's created.

CHAPTER 6

Bumblebee

6.1 Getting Started

Unyson is a framework for WordPress that facilitates development of a theme. This framework was created from the ground up by the team behind ThemeFuse from the desire to empower developers to build outstanding WordPress themes fast and easy. Bumblebee is an extensions package by createIT built on top of Unyson. This documentation is heavily modified by createIT to ensure all custom extensions are well documented.

Note: This documentation assumes you have a working knowledge of WordPress. If you haven't, please start by reading WordPress Documentation.

6.1.1 License

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software. Unyson inherits the General Public License (GPL) from WordPress.

6.2 Convention

These are project guidelines regarding coding and structure.

6.2.1 General

The framework was built following some rules to ensure compatibility between components and to provide an easier way for developers to work together. Here are some starting rules to keep in mind:

- The code should work on **php 5.2.4**, like WordPress Minimum Requirements says. Don't use php 5.3+ features, because some hosting providers don't have php 5.3+ installed on the servers.

- Follow WordPress Coding Standards.

> **Note:** If you already have some code written with spaces indentation (that does not follow WordPress Coding Standards), use this RegExp to replace spaces with tabs:
>
> `(?<=^\s*) {4}` replace with `\t`

6.2.2 Prefixes

In both theme and plugin everything is prefixed to prevent naming conflicts and to give a meaning to functions, classes and methods names. See the guidelines below.

Important: Prefix everything! Lack of proper prefixes is a frequent theme reject reason. Read more: Reject Reasons

Theme

In theme development, a unique and more than 2 letters prefix is required by ThemeForest reviewers, therefore the prefix to use in a theme is `fw_ct_bee_` for variables, functions, classes, and `fw-ct-bee-` for styles, image sizes, db options etc.

- Functions and classes should be prefixed with:
  - `fw_ct_bee_` for functions
  - `FW_Ct_Bee` for classes

```
function fw_ct_bee_head() {
    // ...
}

class FW_Ct_Bee_Pagination
{
    // ...
}
```

- Private functions and classes should be prefixed the same way due to ThemeForest standards
- Functions used for hooks should be prefixed with:
  - `fw_ct_bee_action_` for `add_action()`
  - `fw_ct_bee_filter_` for `add_filter()`

```
/**
 * @internal
 */
function fw_ct_bee_filter_the_content($content) {
    // ...

    return $content;
}
add_filter('the_content', 'fw_ct_bee_filter_the_content');

/**
 * @internal
```

```
 */
function fw_ct_bee_action_init() {
    // ...
}
add_action('init', 'fw_ct_bee_action_init');
```

- Filters and actions should be prefixed with `fw_ct_bee_`.

```
$data = apply_filters('fw_ct_bee_whatever', $data);

do_action('fw_ct_bee_whatever');
```

Extensions

- Public functions and classes should be prefixed with:
  - `fw_ext_<extension-name>_` for functions
  - `FW_Ext_<extension-name>_` for classes
- Private functions and classes should be prefixed the same way due to ThemeForest standards
- Functions used for hooks should be prefixed with:
  - `fw_ext_<extension-name>_action_` for `add_action()`
  - `fw_ext_<extension-name>_filter_` for `add_filter()`

  For e.g. if extension name is `demo`:

```
/**
 * @internal
 */
function fw_ext_demo_filter_the_content($content) {
    // ...

    return $content;
}
add_filter('the_content', 'fw_ext_demo_filter_the_content');

/**
 * @internal
 */
function fw_ext_demo_action_init() {
    // ...
}
add_action('init', 'fw_ext_demo_action_init');
```

- Filters and actions should be prefixed with `'fw_ext_<extension-name>_'`.

  For e.g. if extension name is `demo`:

```
$data = apply_filters('fw_ext_demo_whatever', $data);

do_action('fw_ext_demo_whatever');
```

Unyson Core

- Public functions and classes should be prefixed with:
  - `fw_` for functions
  - `FW_` for classes

```
function fw_useful_function() {
    // ...
}

class FW_Useful_Class {
    // ...
}
```

Note: A **Public function** is meant to be used by anyone. Usually it's a helper function that does something useful.

- Private functions and classes should be prefixed with:
  - `_fw_` for functions
  - `_FW_` for classes

```
/**
 * @internal
 */
function _fw_private_function() {
    // ...
}

/**
 * @internal
 */
class _FW_Private_Class
{
    // ...
}
```

Note: A **private function** is used somewhere internally. Don't forget to use the @internal tag in your PhpDoc in order to make it clear to other developers that this is a private function. It will also remove the function from your documentation (if you are using an automatic documentation generator)

- Functions and methods used for hooks should be prefixed with:
  - `_action_` for `add_action()`
  - `_filter_` for `add_filter()`

```
/**
 * @internal
 */
function _action_init_something() {
    // ...
```

```
}
add_action('init', '_action_init_something');
```

Important: Be sure the function name is unique enough in order to minimize the chances to be defined by someone else. Do not use too simple function names like `_action_init`.

```
class FW_Example
{
    public function __construct()
    {
        add_filter('the_content', array($this, '_filter_the_content'));
    }

    /**
     * @internal
     */
    public function _filter_the_content($content) {
        // ...

        return $content;
    }
}
```

- Filters and actions should be prefixed with `'fw_'`.

```
$data = apply_filters('fw_whatever', $data);

do_action('fw_whatever');
```

6.2.3 Escaping & Translations

In the theme, every output needs to be translatable and properly escaped due to security reasons. Introductory reading:

- https://codex.wordpress.org/Validating_Sanitizing_and_Escaping_User_Data
- https://vip.wordpress.com/2014/06/20/the-importance-of-escaping-all-the-things/

Escaping and translation functions

- wp_kses
- esc_html
- esc_html__
- esc_html_e

- esc_js
- esc_attr
- esc_attr__
- esc_attr_e
- esc_sql
- esc_textarea
- esc_url
- esc_url_raw
- sanitize_email

What needs to be escaped

- All php variables following echo should be escaped

```
echo $var;                                  // wrong
echo esc_html( $var );                      // ok
```

- All php functions following echo should be escaped

```
echo home_url( '/' );                       // wrong
echo esc_url( home_url( '/' ) );            // ok
```

- All translations should be escaped

```
echo __( 'Hello', 'ct_theme' );             // wrong
echo esc_html__( 'Hello', 'ct_theme' );     // ok

_e( 'Hello', 'ct_theme' );                  // wrong
esc_html_e( 'Hello', 'ct_theme' );          // ok
```

- All attributes should be escaped

```
<li id="accordion-section-<?php echo $this->id; ?>" class="<?php echo
↪$classes ?>">                              // wrong
<li id="accordion-section-<?php echo esc_attr( $this->id ); ?>" class="<?
↪php echo esc_attr( $classes ); ?>">    // ok
```

- Variables should **not** and can not be translated

```
echo esc_html__( $var );                    // wrong, generates a␣
↪notice
echo esc_html__( $var, 'ct_theme' );        // wrong
echo esc_html( $var );                      // ok
```

Variables containing HTML

When outputting a variable which contains HTML you can do one of the following:

- Use *wp_kses* or *wp_kses_post* function to define allowed HTML tags

```php
$var = 'Hello <b>world</b>';

echo $var;                                          // wrong
echo esc_html( $var );                              // ok, but <b> tag
↪is lost
echo wp_kses( $var, array( 'b' ) );                 // ok
```

Double check

Make sure you have everything escaped before publishing a theme. In order to double check, search the project for:

- `echo $`
- `__(` or better a regex: `\W__\(`
- `_e(`
- `href=`
- `src=`
- `home_url(`
- `permalink(`

6.2.4 Directory Structure

We've organized the files and folders in order to be easy to understand and use. What follows is the directory and file structure of an Unyson theme:

```
themes/
├─parent-theme/
│  └─framework-customizations/
│     ├─extensions/
│     │  ├─extension-name/
│     │  └─...
│     └─theme/
│        ├─manifest.php   # Theme details: title, description, version, dependencies,
↪etc.
│        ├─config.php     # Theme specific configuration
│        └─options/
│           ├─settings.php # Theme settings options
│           ├─customizer.php # Customizer options
│           ├─posts/       # Post types options
│           │  ├─post.php
│           │  ├─testimonial.php
│           │  ├─{post-type}.php
│           │  └─...
│           └─taxonomies/  # Taxonomy terms options
│              ├─category.php
│              ├─post_tag.php
│              ├─{taxonomy}.php
│              └─...
└─child-theme/
   └─framework-customizations/
      └─... # same as in then parent theme, but here you can overwrite specific files
↪from the parent theme
```

Let's take a closer look at each directory and file, and understand how it works.

- `framework-customizations/theme/` - Contains options, views, helpers, and all bunch of theme stuff, we'll take a closer look at every file below.
- `framework-customizations/theme/manifest.php` - Contains an array with information about theme, accessible through `fw()->theme->manifest->get('key');`. More details about the theme manifest.
- `framework-customizations/theme/config.php` - Theme configuration array, accessible through `fw()->theme->get_config('key');`. Here are the default values.

```
$cfg = array(
    // Theme Settings form ajax submit
    'settings_form_ajax_submit' => true,
    // Theme Settings side tabs
    'settings_form_side_tabs' => true,
);
```

- `framework-customizations/theme/options/` - A directory containing option files: post types, taxonomies, customizer and theme settings page options. The framework will automatically pick them, display in admin pages and save the values in the database. Also you can add custom options files in it, for e.g. `framework-customizations/theme/options/my-options.php` and access them through `fw()->theme->get_options('my-options')`. Use the `fw_get_db_..._option()` functions to get the settings, customizer, posts and taxonomies options values from the database.

  For e.g. you can add options in Customizer in two steps:

  1. Create `{theme}/framework-customizations/theme/options/customizer.php`

```
$options = array(
    'section_1' => array(
        'title' => __('Unyson Section', '{domain}'),
        'options' => array(

            'option_1' => array(
                'type' => 'text',
                'value' => 'Default Value',
                'label' => __('Unyson Option', '{domain}'),
                'desc' => __('Option Description', '{domain}'),
            ),

        ),
    ),
);
```

  2. Use option value in template

```
$value = fw_get_db_customizer_option('option_1');
```

- `framework-customizations/extensions/` - Contains customizations for the framework extensions. You can overwrite options, views and configuration files of the extensions located in the framework or custom locations like other plugins. You can also store there theme extensions and create sub-extensions for extensions located in the framework or custom locations. Extension is identified by its relative path, for e.g. an extension can be located in:

  – Framework `wp-content/plugins/unyson/framework/extensions/{extension-name}`

  - Plugin `wp-content/plugins/whaterever-plugin/custom-dir/extensions/{extension-name}`

that extension can be customized in `{theme}/framework-customizations/extensions/{extension-name}`. Also you can create a sub-extension in `{theme}/framework-customizations/extensions/{extension-name}/extensions/{sub-extension-name}`.

You can also create a `framework-customizations/` directory in the child theme. There you can do the same things as in parent theme, and also you can overwrite some files from the parent theme, like options and configuration files. Keep in mind that some files from the child theme are included before the parent theme files (or the other way around, it depends on the case) to give you the ability to customize some parent theme behavior.

6.3 Options

6.3.1 Introduction

Introduction

Options are intended for creating form fields representing different kind of data e.g. rich and plain text, icons, media content, fonts and more. With options you can easily create tabs, boxes and form inputs for the admin pages. You just build an array and it will be transformed to html. On form submit, values will be saved into the database, and you will be able to access them anywhere you want using `fw_get_db_..._option()` helper functions.

For advanced users, this is an easy way to create form inputs and use them for various purposes. The simplest options array looks something like this:

```
$options = array(
    'option_id' => array(
        'type' => 'text'
    )
);
```

This will generate a text input. The array key is used as option id, it should be unique. Values in the database will be stored as `array('option_id' => 'value')`.

Note: The only required parameter for any option is `type`.

All options have some base parameters:

- `label` *(string)* Label
- `desc` *(string)* Description
- `value` *(mixed)* Default value

- `attr` *(array)* HTML attributes *(some options will place these attributes in input, other in wrapper div)*
- `help` *(string|array)* Additional info about option. This will generate an  next to option that will show the text in a tip popup.

Some options can have additional (optional) parameters. A better customized option will look like this:

```
$options = array(
    'option_id' => array(
        'type'  => 'text',
        'value' => 'Default value',
        'label' => __('Option Label', '{domain}'),
        'desc'  => __('Option Description', '{domain}'),
        'attr'  => array('class' => 'custom-class', 'data-foo' => 'bar'),
        'help'  => __('Some html that will appear in tip popup', '{domain}'),
    )
);
```

You can test the above array by creating any of the below options file and placing the array in it.

Options files

These are the main places where options are used:

- **Theme Settings Page**: Loads the options from `{theme}/framework-customizations/theme/options/settings.php`
- **Customizer Page**: Loads the options from `{theme}/framework-customizations/theme/options/customizer.php`
- **Post Add/Edit Page**: Loads the options from `{theme}/framework-customizations/theme/options/posts/{$post_type}.php`
- **Taxonomy Term Edit Page**: Loads the options from `{theme}/framework-customizations/theme/options/taxonomies/{$taxonomy}.php`

Containers

Options that have no value and contain other options in the `options` parameter are containers. If an option has the `options` parameter, it is considered a container.

The simplest container option array looks as in the below example and will generate an empty metabox without title:

```
$options = array(
    array(
        'type'    => 'box',
        'options' => array()
    )
);
```

Note: Like options, containers have a minimum set of required parameters: `type` and `options`. The `type` parameter in Customizer options is optional and it's not used (has no effect).

There are 4 built-in container types:

- `box` - WordPress metabox

```
'box_id' => array(
    'type' => 'box',
    'options' => array(
        'option_id'  => array( 'type' => 'text' ),
    ),
    'title' => __('Box Title', '{domain}'),
    'attr' => array('class' => 'custom-class', 'data-foo' => 'bar'),

    /**
     * When used in Post Options on the first array level
     * the ``box`` container accepts additional parameters
     */
    //'context' => 'normal|advanced|side',
    //'priority' => 'default|high|core|low',
),
```

- `tab` - One tab *(Tabs from the same array level will be collected and rendered as multiple tabs)*

```
'tab_id' => array(
    'type' => 'tab',
    'options' => array(
        'option_id'  => array( 'type' => 'text' ),
    ),
    'title' => __('Tab Title', '{domain}'),
    'attr' => array('class' => 'custom-class', 'data-foo' => 'bar'),
),
'tab_id_2' => array(
    'type' => 'tab',
    'options' => array(
        'option_id_2'  => array( 'type' => 'text' ),
    ),
    'title' => __('Tab Title #2', '{domain}'),
    'attr' => array('class' => 'custom-class', 'data-foo' => 'bar'),
),
```

- `group` - Group options into a wrapper div. Has no design. Usually used to show/hide a group of options from javascript

```
'group_id' => array(
    'type' => 'group',
    'options' => array(
        'option_id'  => array( 'type' => 'text' ),
    ),
    'attr' => array('class' => 'custom-class', 'data-foo' => 'bar'),
),
```

- `popup` - A button, when clicked it will open a modal with options

```
'popup_id' => array(
    'type' => 'popup',
    'options' => array(
        'option_id'  => array( 'type' => 'text' ),
    ),
    'title' => __('Button and Popup Title', '{domain}'),
    'attr' => array('class' => 'custom-class', 'data-foo' => 'bar'),
    'modal-size' => 'small', // small, medium, large
    'desc' => __('Button Description', '{domain}'),
),
```

Restrictions

Here are some restrictions to keep in mind:

- `attr` parameter from **Post Options** first level `box` containers, is not used. Because boxes are added with add_meta_box() which has no parameter for specifying attributes.

Note: There are no restrictions (except Customizer) for what options are contained in the `options` parameter. It's possible to create multi level options: boxes inside boxes, tabs inside boxes, tabs inside tabs, and so on.

6.3.2 Theme Integration

With options you can easy create admin forms and use the values in frontend. Let's to this!

Customizer Options

1. Create `{theme}/framework-customizations/theme/options/customizer.php` with the following contents:

```
<?php if (!defined( 'FW' )) die('Forbidden');

$options = array(
    'body-color' => array(
        'type' => 'color-picker',
        'label' => __('Body Color', '{domain}'),
        'value' => '#ADFF2F',
    ),
);
```

2. Add in `{theme}/functions.php`

```
function _action_theme_wp_print_styles() {
    if (!defined('FW')) return; // prevent fatal error when the framework is not␣
↪active

    $option_value = fw_get_db_customizer_option('body-color');

    echo '<style type="text/css">'
         . 'body { '
         . 'border: 30px solid '. esc_html($option_value) .'; '
         . '}'
         . '</style>';
}
add_action('wp_print_styles', '_action_theme_wp_print_styles');
```

3. Go to menu **Appearance > Customize**, find the **Body Color** option and change it.

Hint: You can enable *Live Preview* for customizer options.

Settings Options

1. Create `{theme}/framework-customizations/theme/options/settings.php` with the following contents:

```
<?php if (!defined( 'FW' )) die('Forbidden');

$options = array(
    'body-color' => array(
        'type' => 'color-picker',
        'label' => __('Body Color', '{domain}'),
        'value' => '#ADFF2F',
    ),
);
```

2. Add in `{theme}/functions.php`

```
function _action_theme_wp_print_styles() {
    if (!defined('FW')) return; // prevent fatal error when the framework is not␣
↪active

    $option_value = fw_get_db_settings_option('body-color');

    echo '<style type="text/css">'
         . 'body { '
         . 'border: 30px solid '. esc_html($option_value) .'; '
         . '}'
         . '</style>';
}
add_action('wp_print_styles', '_action_theme_wp_print_styles');
```

3. Go to menu **Appearance > Theme Settings**, find the **Body Color** option, change it and press Save.
4. Go to frontend and see the changes.

Post Options

1. Create `{theme}/framework-customizations/theme/options/posts/post.php` with the following contents:

```
<?php if (!defined( 'FW' )) die('Forbidden');

$options = array(
    'main' => array(
        'type' => 'box',
        'title' => __('Testing Options', '{domain}'),
        'options' => array(
            'body-color' => array(
                'type' => 'color-picker',
                'label' => __('Body Color', '{domain}'),
                'value' => '#ADFF2F',
            ),
        ),
    ),
);
```

2. Add in `{theme}/functions.php`

```
function _action_theme_wp_print_styles() {
    if (!defined('FW')) return; // prevent fatal error when the framework is not␣
↪active

    global $post;

    if (!$post || $post->post_type != 'post') {
        return;
    }

    $option_value = fw_get_db_post_option($post->ID, 'body-color');

    echo '<style type="text/css">'
        . 'body { '
        . 'border: 30px solid '. esc_html($option_value) .'; '
        . '}'
        . '</style>';
}
add_action('wp_print_styles', '_action_theme_wp_print_styles');
```

3. Create a new Post, find **Body Color** option, change it and save the post.
4. Open the post in frontend and see the changes.

6.3.3 Customizer

Introduction

Customizer Options `{theme}/framework-customizations/theme/options/customizer.php` are turned into Customizer elements (panels, sections and controls).

The customizer elements have a strict structure which also applies to options array structure:

- Containers can be nested only 2 levels
  - `container > option` is turned into `section > control`
  - `container > container > option` is turned into `panel > section > control`
  - `container > container > container > option` will not work `panel > section > ERROR`
- Containers must contain only options or only containers, because a panel can't contain both sections and controls.

Examples

Try the below arrays in `{theme}/framework-customizations/theme/options/customizer.php`.

- Create a Section

```
$options = array(
    'section_1' => array(
        'title' => __('Unyson Section', '{domain}'),
        'options' => array(

            'option_1' => array(
                'type' => 'text',
                'value' => 'Default Value',
                'label' => __('Unyson Option', '{domain}'),
                'desc' => __('Option Description', '{domain}'),
            ),

        ),
    ),
);
```

- Create a Panel with Sections

```
$options = array(
    'panel_1' => array(
        'title' => __('Unyson Panel', '{domain}'),
        'options' => array(

            'section_1' => array(
                'title' => __('Unyson Section #1', '{domain}'),
                'options' => array(

                    'option_1' => array(
                        'type' => 'text',
                        'value' => 'Default Value',
                        'label' => __('Unyson Option #1', '{domain}'),
                        'desc' => __('Option Description', '{domain}'),
                    ),

                ),
            ),

            'section_2' => array(
                'title' => __('Unyson Section #2', '{domain}'),
                'options' => array(

                    'option_2' => array(
                        'type' => 'text',
                        'value' => 'Default Value',
                        'label' => __('Unyson Option #2', '{domain}'),
                        'desc' => __('Option Description', '{domain}'),
                    ),
                    'option_3' => array(
                        'type' => 'text',
                        'value' => 'Default Value',
                        'label' => __('Unyson Option #3', '{domain}'),
                        'desc' => __('Option Description', '{domain}'),
                    ),

                ),
            ),
```

```
        ),
    ),
);
```

- Get option database/saved value in template

```
$value = fw_get_db_customizer_option('option_1');
```

Additional Arguments

- Control arguments can be set in `wp-customizer-args` option parameter.

```
$options = array(
    'section_1' => array(
        'title' => __('Unyson Section', '{domain}'),
        'options' => array(

            'option_1' => array(
                'type' => 'text',
                'value' => 'Default Value',
                'label' => __('Unyson Option', '{domain}'),
                'desc' => __('Option Description', '{domain}'),

                'wp-customizer-args' => array(
                    'priority' => 3,
                ),
            ),

        ),
    ),
);
```

- Setting arguments can be set in `wp-customizer-setting-args` option parameter.

```
$options = array(
    'section_1' => array(
        'title' => __('Unyson Section', '{domain}'),
        'options' => array(

            'option_1' => array(
                'type' => 'text',
                'value' => 'Default Value',
                'label' => __('Unyson Option', '{domain}'),
                'desc' => __('Option Description', '{domain}'),

                'wp-customizer-setting-args' => array(
                    'capability' => 'edit_posts',
                ),
            ),

        ),
    ),
);
```

Live Preview

In background, customizer options are converted into customizer elements, so they follow default WordPress behavior and implementing a live preview can be done using the default WordPress solution.

1. Change the setting transport and enqueue the javascript

```
// file: {theme}/inc/hooks.php

if (defined('FW')):
    /**
     * @param WP_Customize_Manager $wp_customize
     * @internal
     */
    function _action_customizer_live_fw_options($wp_customize) {
        if ($wp_customize->get_setting('fw_options[OPTION_ID]')) {
            $wp_customize->get_setting('fw_options[OPTION_ID]')->
↪transport = 'postMessage';

            add_action( 'customize_preview_init', '_action_customizer_
↪live_fw_options_preview' );
        }
    }
    add_action('customize_register', '_action_customizer_live_fw_options
↪');

    /**
     * @internal
     */
    function _action_customizer_live_fw_options_preview() {
        wp_enqueue_script(
            'mytheme-customizer',
            get_template_directory_uri() .'/assets/js/theme-customizer.js
↪',
            array( 'jquery','customize-preview' ),
            fw()->theme->manifest->get_version(),
            true
        );
    }
endif;
```

2. Handle the change in javascript

```
// file: {theme}/assets/js/theme-customizer.js

( function( $ ) {
    wp.customize( 'fw_options[OPTION_ID]', function( value ) {
        value.bind( function( newval ) {
            /**
             * An array of collected html inputs
             * [{'name':'input[name]','value':'input value'}]
             * or
             * [{'name':'input[name]','value':'input value'},{'name':
↪'input[name]','value':'input value'},...]
             */
            newval = JSON.parse(newval);

            $( 'h1' ).text( newval[0].value );
        } );
```

```
    } );
} )( jQuery );
```

Note: The value comes in `[{'name':'input[name]','value':'input value'}]` format, because the customizer form is not submitted as a regular form. A control can store its value only inside a single input which has some special attributes (instead of `name="..."`) and it is monitored for changes by the Customizer script to trigger the preview update. Because of that, the framework options collect all their inputs values and store them in that special input (here is an advanced explanation).

6.3.4 Option Types

Every option has `type` as a required parameter. Its value should be an existing registered option type.

HTML

All option types must have `.fw-option-type-{type}` class on main/wrapper html element.

CSS

If the option type has css, all rules must be prefixed with `.fw-option-type-{type}` class:

```
/* correct */
.fw-option-type-demo .some-class {
    color: blue;
}

/* wrong */
.some-class {
    color: blue;
}
```

Tip: This is done to prevent css conflicts.

Javascript

All javascript must stick to `.fw-option-type-{type}` class and work only within the main/wrapper element (no events attached to the body). If the option type has custom javascript events, those events must be triggered on the main element.

```
$someInnerElement.closest('.fw-option-type-demo')
    .trigger('fw:option-type:demo:custom-event', {some: 'data'});
```

If it's specified in the documentation that an option type has custom events, it means that you can attach event listeners on the elements with `.fw-option-type-{type}` class (not on body or `fwEvents`).

Caution: Do not confuse `.fw-option-type-{type}` with `.fw-backend-option-type-{type}` class which is used internally by the framework and should not be used in option type scripts.

6.3.5 Built-in Option Types

Here is a complete list of all built-in option types with all available parameters for each option.

Text

Regular text input.

```
array(
    'type'  => 'text',
    'value' => 'default value',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
)
```

Textarea

Regular textarea.

```
array(
    'type'  => 'textarea',
    'value' => 'default value',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
)
```

Checkbox

Single checkbox.

```
array(
    'type'  => 'checkbox',
    'value' => true, // checked/unchecked
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'text'  => __('Yes', '{domain}'),
)
```

Checkboxes

A list of checkboxes.

```
array(
    'type'  => 'checkboxes',
    'value' => array(
        'choice-1' => false,
        'choice-2' => true,
    ),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'choices' => array( // Note: Avoid bool or int keys http://bit.ly/1cQgVzk
        'choice-1' => __('Choice 1', '{domain}'),
        'choice-2' => __('Choice 2', '{domain}'),
        'choice-3' => __('Choice 3', '{domain}'),
    ),
    // Display choices inline instead of list
    'inline' => false,
)
```

Radio

A list of radio buttons.

```
array(
    'type'  => 'radio',
    'value' => 'choice-3',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'choices' => array( // Note: Avoid bool or int keys http://bit.ly/1cQgVzk
        'choice-1' => __('Choice 1', '{domain}'),
        'choice-2' => __('Choice 2', '{domain}'),
        'choice-3' => __('Choice 3', '{domain}'),
    ),
    // Display choices inline instead of list
    'inline' => false,
)
```

Select

Regular select.

```
array(
    'type'  => 'select',
    'value' => 'choice-3',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'choices' => array(
        '' => '---',
        'choice-1' => __('Choice 1', '{domain}'),
        'choice-2' => array(
            'text' => __('Choice 2', '{domain}'),
```

```
            'attr' => array('data-foo' => 'bar'),
        ),
        array( // optgroup
            'attr'    => array('label' => __('Group 1', '{domain}')),
            'choices' => array(
                'choice-3' => __('Choice 3', '{domain}'),
                // ...
            ),
        ),
    ),
    /**
     * Allow save not existing choices
     * Useful when you use the select to populate it dynamically from js
     */
    'no-validate' => false,
)
```

Select Multiple

Select with multiple values.

```
array(
    'type'  => 'select-multiple',
    'value' => array( 'choice-1', 'choice-3' ),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'choices' => array(
        '' => '---',
        'choice-1' => __('Choice 1', '{domain}'),
        'choice-2' => array(
            'text' => __('Choice 2', '{domain}'),
            'attr' => array('data-foo' => 'bar'),
        ),
        array( // optgroup
            'attr'    => array('label' => __('Group 1', '{domain}')),
            'choices' => array(
                'choice-3' => __('Choice 3', '{domain}'),
                // ...
            ),
        ),
    ),
)
```

Multi-Select

Select multiple choices from different sources: posts, taxonomies, users or a custom array.

```
array(
    'type'  => 'multi-select',
    'value' => array( 'choice-1', 'choice-3' ),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
```

```
    'help'  => __('Help tip', '{domain}'),
    /**
     * Set population method
     * Are available: 'posts', 'taxonomy', 'users', 'array'
     */
    'population' => 'array',
    /**
     * Set post types, taxonomies, user roles to search for
     *
     * 'population' => 'posts'
     * 'source' => 'page',
     *
     * 'population' => 'taxonomy'
     * 'source' => 'category',
     *
     * 'population' => 'users'
     * 'source' => array( 'editor', 'subscriber', 'author' ),
     *
     * 'population' => 'array'
     * 'source' => '' // will populate with 'choices' array
     */
    'source' => '',
    /**
     * Set the number of posts/users/taxonomies that multi-select will be prepopulated
     * Or set the value to false in order to disable this functionality.
     */
    'prepopulate' => 10,
    /**
     * An array with the available choices
     * Used only when 'population' => 'array'
     */
    'choices' => array(
        'choice-1' => __('Choice 1', '{domain}'),
        'choice-2' => __('Choice 2', '{domain}'),
        'choice-3' => __('Choice 3', '{domain}'),
    ),
    /**
     * Set maximum items number that can be selected
     */
    'limit' => 100,
)
```

Visual Composer Integration

Tip: This is a Bumblebee feature.

When converting 'multi-upload' option type to VC 'autocomplete' param, the following settings are set by default:

```
vc_map( array(
    'type'        => 'autocomplete',
    'param_name'  => ...
    'description' => ...
    'class'       => ...
    'group'       => ...
```

```
    'dependency'  => ...
    'heading'     => ...
    'settings'    => array( // can be overwritten by $options['settings']
        'groups'         => false, // not compatible with multi-select
        'multiple'       => true,  // can be changed by 'limit' option == 1
        'min_length'     => 1,
        'unique_values'  => true,
        'display_inline' => true,
        'delay'          => 500,
        'auto_focus'     => true,
    ),
);
```

'settings' array can be changed by passing 'settings' array in the option, eg.:

```
'pages'  => array(
    'type'       => 'multi-select',
    'label'      => esc_html__( "Select pages", 'ct_theme' ),
    'tab'        => esc_html__( "Pages", 'ct_theme' ),
    'limit'      => 1,
    'population' => 'posts',
    'source'     => 'page',
    'settings' => array(
        'groups'         => false, // not compatible with multi-select
        'multiple'       => false,
        'min_length'     => 3,
        'unique_values'  => false,
        'display_inline' => false,
        'delay'          => 3000,
        'auto_focus'     => false,
    ),
),
```

Read more about 'autocomplete' param at github

Switch

Switch between two choices.

```
array(
    'type'  => 'switch',
    'value' => 'hello',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'left-choice' => array(
        'value' => 'goodbye',
        'label' => __('Goodbye', '{domain}'),
    ),
    'right-choice' => array(
        'value' => 'hello',
        'label' => __('Hello', '{domain}'),
    ),
)
```

Custom Events

`fw:option-type:switch:change` - Value was changed.

Note: Switch value in html is json encoded to prevent issues with boolean values, so before using the html value in javascript do `value = JSON.parse(value);`

Color Picker

Pick a color.

```
array(
    'type'  => 'color-picker',
    'value' => '#FF0000',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    // palette colors array
    'palettes' => array( '#ba4e4e', '#0ce9ed', '#941940' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
)
```

RGBA Color Picker

Pick a `rgba()` color.

```
array(
    'type'  => 'rgba-color-picker',
    'value' => 'rgba(255,0,0,0.5)',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    // palette colors array
    'palettes' => array( '#ba4e4e', '#0ce9ed', '#941940' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
)
```

Gradient

Pick gradient colors.

```
array(
    'type'  => 'gradient',
    'value' => array(
        'primary'   => '#FF0000',
        'secondary' => '#0000FF',
    )
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
)
```

Image Picker

Pick an image.

```
array(
    'type'  => 'image-picker',
    'value' => 'image-2',
    'attr'  => array(
        'class'    => 'custom-class',
        'data-foo' => 'bar',
    ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'choices' => array(
        'value-1' => get_template_directory_uri() .'/images/thumbnail.png',
        'value-2' => array(
            // (required) url for thumbnail
            'small' => get_template_directory_uri() .'/images/thumbnail.png',
            // (optional) url for large image that will appear in tooltip
            'large' => get_template_directory_uri() .'/images/preview.png',
            // (optional) choice extra data for js, available in custom events
            'data' => array(...)
        ),
        'value-3' => array(
            // (required) url for thumbnail
            'small' => array(
                'src' => get_template_directory_uri() .'/images/thumbnail.png',
                'height' => 70
            ),
            // (optional) url for large image that will appear in tooltip
            'large' => array(
                'src' => get_template_directory_uri() .'/images/preview.png',
                'height' => 400
            ),
            // (optional) choice extra data for js, available in custom events
            'data' => array(...)
        ),
    ),
    'blank' => true, // (optional) if true, images can be deselected
)
```

Custom Events

`fw:option-type:image-picker:clicked` - A thumbnail was clicked.

`fw:option-type:image-picker:changed` - Value was changed.

Background Image

Choose background image.

```
array(
    'type'  => 'background-image',
    'value' => 'bg-1',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
```

```
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'choices' => array(
        'none' => array(
            'icon' => get_template_directory_uri() . '/images/bg/bg-0.jpg',
            'css'  => array(
                'background-image' => 'none'
            ),
        ),
        'bg-1' => array(
            'icon'  => get_template_directory_uri() . '/images/bg/bg-1.jpg',
            'css'  => array(
                'background-image'  => 'url("' . get_template_directory_uri() . '/
↪images/bg-1.png' . '")',
                'background-repeat' => 'repeat',
            ),
        ),
        'bg-2' => array(
            'icon' => get_template_directory_uri() . '/images/bg/bg-2.jpg',
            'css'  => array(
                'background-image'  => 'url("' . get_template_directory_uri() . '/
↪images/bg-2.png' . '")',
                'background-repeat' => 'repeat-y'
            ),
        )
    )
)
```

Date Picker

Pick a date in calendar.

```
array(
    'type'  => 'date-picker',
    'value' => '',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'monday-first' => true, // The week will begin with Monday; for Sunday, set to␣
↪false
    'min-date' => date('d-m-Y'), // By default minimum date will be current day. Set␣
↪a date in format d-m-Y as a start date
    'max-date' => null, // By default there is not maximum date. Set a date in format␣
↪d-m-Y as a start date
)
```

Datetime Picker

Pick a datetime in calendar.

```
array(
    'type'  => 'datetime-picker',
    'value' => '',
```

```
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'datetime-picker' => array(
        'format'        => 'Y/m/d H:i', // Format datetime.
        'maxDate'       => false,  // By default there is not maximum date , set a␣
↪date in the datetime format.
        'minDate'       => false,  // By default minimum date will be current day,␣
↪set a date in the datetime format.
        'timepicker'    => true,   // Show timepicker.
        'datepicker'    => true,   // Show datepicker.
        'defaultTime'   => '12:00' // If the input value is empty, timepicker will␣
↪set time use defaultTime.
    ),
)
```

Datetime Range

Set a datetime range.

```
array(
    'type'  => 'datetime-range',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'datetime-pickers' => array(
    'from' => array(
        'minDate' => '1970/01/01', // By default minimum date will be current day,␣
↪set a date in the datetime format.
        'maxDate' => '2038/01/19', // By default there is not maximum date , set a␣
↪date in the datetime format.
        'format'  => 'Y/m/d H:i',  // Format datetime.
        'timepicker'  => true,     // Show timepicker.
        'datepicker'  => true,     // Show datepicker.
        ),
    'to' => array(
        'minDate' => '1970/01/01', // By default minimum date will be current day,␣
↪set a date in the datetime format.
        'maxDate' => '2038/01/19', // By default there is not maximum date , set a␣
↪date in the datetime format.
        'format'  => 'Y/m/d H:i',  // Format datetime.
        'timepicker'  => true,     // Show timepicker.
        'datepicker'  => true,     // Show datepicker.
        )
    ),
    'value' => array(
        'from' => '',
        'to' => ''
    )
)
```

Icon v2

```
array(
    'type'  => 'icon-v2',

    /**
     * small | medium | large | sauron
     * Yes, sauron. Definitely try it. Great one.
     */
    'preview_size' => 'medium',

    /**
     * small | medium | large
     */
    'modal_size' => 'medium',

    /**
     * There's no point in configuring value from code here.
     *
     * I'll document the result you get in the frontend here:
     * 'value' => array(
     *   'type' => 'icon-font', // icon-font | custom-upload
     *
     *   // ONLY IF icon-font
     *   'icon-class' => '',
     *   'icon-class-without-root' => false,
     *   'pack-name' => false,
     *   'pack-css-uri' => false
     *
     *   // ONLY IF custom-upload
     *   // 'attachment-id' => false,
     *   // 'url' => false
     * ),
     */

    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
)
```

Default value is not really supported, because of the complexity of the data that this option type holds.

The second version of the first Icon option type. It was improved a lot in terms of both UI and extensibility. The user will be able to filter through a list of icon packs and also upload his own icon. The result value will contain `type` field and it will contain the type of the selected content. It can be `icon-font` or `custom-upload`. You'll also get favorite icon functionallity which will work out of the box.

Note: You'll have to enable *SVG* uploads by yourself, with a hook in your theme.

By default, we have just 6 icon packs enabled and served with Unyson itself.

- Font Awesome
- Entypo
- Linecons

- Linearicons
- Typicons
- Unycons

Note: By default, none of this packs will be enqueued in the frontend of your theme.

You should call this in order to enqueue them: `fw()->backend->option_type('icon-v2')->packs_loader->enqueue`

Configure Icon Packs

Icon V2 is easily extensible with a couple of filters you can hook into. First, you may want to configure which of the *already* registered packs we should display into the picker.

```
function _custom_packs_list($current_packs) {
    /**
     * $current_packs is an array of pack names.
     * You should return which one you would like to show in the picker.
     */
    return array('font-awesome', 'unycon');
}

add_filter('fw:option_type:icon-v2:filter_packs', '_custom_packs_list');
```

Note: That's a global hook which changes behavior for **all** pickers. Configuring packs per picker is not available and will **not** be implemented later. If you have some particular use case for this, please fill an issue.

Add Icon Pack

Long story short, you can add more packs by filtering on `fw:option_type:icon-v2:packs` filter. Simplest example, all of the keys are required:

```
add_filter('fw:option_type:icon-v2:packs', '_add_my_pack');

function _add_my_pack($default_packs) {
    /**
     * No fear. Defaults packs will be merged in back. You can't remove them.
     * Changing some flags for them is allowed.
     */
    return array(
      'my_pack' => array(
        'name' => 'my_pack', // same as key
        'title' => 'My Cool Pack',
        'css_class_prefix' => 'my-pack',
        'css_file' => 'path_to_css_file',
        'css_file_uri' => 'network_accessible_url'
      )
    )
}
```

And this will just work for most of the cases. You don't need to specify which icons specifically to show inside the picker. All of them will be showed, by default. In fact, there's some magick going on that will extract all of your icons and show them up. I'll try to make it clear below.

Computing icons list

By default, when you register an icon pack it's icons will be extracted from the css file automatically, so that you don't have to maintain a long array of icons for each pack. Instead we do some trick instead. We look into the css file for each pack and look for patterns that look like this:

```
.`css_class_prefix`-some-icon:before {
    content: '\266a';
}
```

`css_class_prefix` there refers to the `css_class_prefix` option you specified for your icon pack.

```
// Those will be considered an icon
.my-pack-some-icon:before { content: '\266a'; }
.my-pack.my-pack-some-icon:before { content: '\266a'; }
.my-pack.my-pack-some-icon:after { content: '\266a'; }

// This one won't
.my-pack.my-pack-some-icon:after { color: red; }
```

Generally speaking, that's what an icon pack CSS file consist of:

- `@font-face` rules
- icon generations – we try hard to get just them
- some other general purpose helpers – they're encountered not that often

You can also completely stop this mechanism for one pack by specifying an array of icons for the `icons` option. A more complete pack definition can be found here.

Upload

Single file upload.

```
array(
    'type'  => 'upload',
    'value' => array(
        /*
        'attachment_id' => '9',
        'url' => '//site.com/wp-content/uploads/2014/02/whatever.jpg'
        */
        // if value is set in code, it is not considered and not used
        // because there is no sense to set hardcode attachment_id
    ),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    /**
     * If set to `true`, the option will allow to upload only images, and display a␣
↪thumb of the selected one.
     * If set to `false`, the option will allow to upload any file from the media␣
↪library.
```

```
     */
    'images_only' => true,
    /**
     * An array with allowed files extensions what will filter the media library and␣
↪the upload files.
     */
    'files_ext' => array( 'doc', 'pdf', 'zip' ),
    /**
     * An array with extra mime types that is not in the default array with mime␣
↪types from the javascript Plupload library.
     * The format is: array( '<mime-type>, <ext1> <ext2> <ext2>' ).
     * For example: you set rar format to filter, but the filter ignore it , than you␣
↪must set
     * the array with the next structure array( '.rar, rar' ) and it will solve the␣
↪problem.
     */
    'extra_mime_types' => array( 'audio/x-aiff, aif aiff' )
)
```

Custom Events

`fw:option-type:upload:change` - The value was changed.

`fw:option-type:upload:clear` - The value was cleared (the selected item is removed).

Multi-Upload

Upload multiple files.

```
array(
    'type'  => 'multi-upload',
    'value' => array(
        /*
        array(
            'attachment_id' => '9',
            'url' => '//site.com/wp-content/uploads/2014/02/whatever.jpg'
        ),
        ...
        */
        // if value is set in code, it is not considered and not used
        // because there is no sense to set hardcode attachment_id
    ),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    /**
     * If set to `true`, the option will allow to upload only images, and display a␣
↪thumb of the selected one.
     * If set to `false`, the option will allow to upload any file from the media␣
↪library.
     */
    'images_only' => true,
    /**
     * An array with allowed files extensions what will filter the media library and␣
↪the upload files.
```

```
     */
    'files_ext' => array( 'doc', 'pdf', 'zip' ),
    /**
     * An array with extra mime types that is not in the default array with mime␣
↪types from the javascript Plupload library.
     * The format is: array( '<mime-type>, <ext1> <ext2> <ext2>' ).
     * For example: you set rar format to filter, but the filter ignore it , than you␣
↪must set
     * the array with the next structure array( '.rar, rar' ) and it will solve the␣
↪problem.
     */
    'extra_mime_types' => array( 'audio/x-aiff, aif aiff' )
)
```

Custom Events

`fw:option-type:multi-upload:change` - The value was changed.

`fw:option-type:multi-upload:clear` - The value is cleared (all the selected items are removed).

`fw:option-type:multi-upload:remove` - A thumb (selected item) is removed. Triggered only when `images_only` is set to `true`.

Slider

Drag the handle to select a numeric value.

```
array(
    'type'  => 'slider',
    'value' => 33,
    'properties' => array(
        /*
        'min' => 0,
        'max' => 100,
        'step' => 1, // Set slider step. Always > 0. Could be fractional.
        */
    ),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
)
```

Range Slider

Drag the handles to set a numeric value range.

```
array(
    'type'  => 'range-slider',
    'value' => array(
        'from' => 10,
        'to'   => 33,
    ),
    'properties' => array(
```

```
        /*
        'min' => 0,
        'max' => 100,
        'step' => 1, // Set slider step. Always > 0. Could be fractional.
        */
    ),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
)
```

Popup

Popup with options.

```
array(
    'type' => 'popup',
    'value' => array(
        'option_1' => 'value 1',
        'option_2' => 'value 2',
    ),
    'label' => __('Popup', '{domain}'),
    'desc'  => __('Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do␣
↪eiusmod tempor incididunt ut labore et dolore magna aliqua.', '{domain}'),
    'popup-title' => __('Popup Title', '{domain}'),
    'button' => __('Edit', '{domain}'),
    'popup-title' => null,
    'size' => 'small', // small, medium, large
    'popup-options' => array(
        'option_1' => array(
            'label' => __('Text', '{domain}'),
            'type' => 'text',
            'value' => 'Demo text value',
            'desc' => __('Lorem ipsum dolor sit amet, consectetur adipisicing elit,␣
↪sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.', '{domain}'),
            'help' => sprintf("%s \n\n'\"<br/><br/>\n\n <b>%s</b>",
                __('Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do␣
↪eiusmod tempor incididunt ut labore et dolore magna aliqua.', '{domain}'),
                __('Sed ut perspiciatis, unde omnis iste natus error sit voluptatem␣
↪accusantium doloremque laudantium', '{domain}')
            ),
        ),
        'option_2' => array(
            'label' => __('Textarea', '{domain}'),
            'type' => 'textarea',
            'value' => 'Demo textarea value',
            'desc' => __('Lorem ipsum dolor sit amet, consectetur adipisicing elit,␣
↪sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.', '{domain}'),
            'help' => sprintf("%s \n\n'\"<br/><br/>\n\n <b>%s</b>",
                __('Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do␣
↪eiusmod tempor incididunt ut labore et dolore magna aliqua.', '{domain}'),
                __('Sed ut perspiciatis, unde omnis iste natus error sit voluptatem␣
↪accusantium doloremque laudantium', '{domain}')
            ),
        ),
```

```
    ),
)
```

Addable Popup

Addable popup with options.

```
array(
    'type' => 'addable-popup',
    'value' => array(
        array(
            'option_1' => 'value 1',
            'option_2' => 'value 2',
        ),
        // ...
    ),
    'label' => __('Addable Popup', '{domain}'),
    'desc'  => __('Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do␣
↪eiusmod tempor incididunt ut labore et dolore magna aliqua.', '{domain}'),
    'template' => '{{- demo_text }}',
    'popup-title' => null,
    'size' => 'small', // small, medium, large
    'limit' => 0, // limit the number of popup`s that can be added
    'add-button-text' => __('Add', '{domain}'),
    'sortable' => true,
    'popup-options' => array(
        'option_1' => array(
            'label' => __('Text', '{domain}'),
            'type' => 'text',
            'value' => 'Demo text value',
            'desc' => __('Lorem ipsum dolor sit amet, consectetur adipisicing elit,␣
↪sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.', '{domain}'),
            'help' => sprintf("%s \n\n'\"<br/><br/>\n\n <b>%s</b>",
                __('Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do␣
↪eiusmod tempor incididunt ut labore et dolore magna aliqua.', '{domain}'),
                __('Sed ut perspiciatis, unde omnis iste natus error sit voluptatem␣
↪accusantium doloremque laudantium', '{domain}')
            ),
        ),
        'option_2' => array(
            'label' => __('Textarea', '{domain}'),
            'type' => 'textarea',
            'value' => 'Demo textarea value',
            'desc' => __('Lorem ipsum dolor sit amet, consectetur adipisicing elit,␣
↪sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.', '{domain}'),
            'help' => sprintf("%s \n\n'\"<br/><br/>\n\n <b>%s</b>",
                __('Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do␣
↪eiusmod tempor incididunt ut labore et dolore magna aliqua.', '{domain}'),
                __('Sed ut perspiciatis, unde omnis iste natus error sit voluptatem␣
↪accusantium doloremque laudantium', '{domain}')
            ),
        ),
    ),
)
```

Addable Option

Create a list of options.

```
array(
    'type'  => 'addable-option',
    'value' => array('Value 1', 'Value 2', 'Value 3'),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'option' => array( 'type' => 'text' ),
    'add-button-text' => __('Add', '{domain}'),
    'sortable' => true,
)
```

Custom Events

`fw:option-type:addable-option:option:init` - New option was added and initialized.

Addable Box

Addable box with options.

```
array(
    'type'  => 'addable-box',
    'value' => array(
        array(
            'option_1' => 'value 1',
            'option_2' => 'value 2',
        ),
        // ...
    ),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'box-options' => array(
        'option_1' => array( 'type' => 'text' ),
        'option_2' => array( 'type' => 'textarea' ),
    ),
    'template' => 'Hello {{- option_1 }}', // box title
    'box-controls' => array( // buttons next to (x) remove box button
        'control-id' => '<small class="dashicons dashicons-smiley"></small>',
    ),
    'limit' => 0, // limit the number of boxes that can be added
    'add-button-text' => __('Add', '{domain}'),
    'sortable' => true,
)
```

Custom Events

`fw:option-type:addable-box:box:init` - Box was initialized. Triggered for each existing box after page load, or when a box was added.

`fw:option-type:addable-box:control:click` - A custom control was clicked.

Visual Composer Integration

Tip: This is a Bumblebee feature.

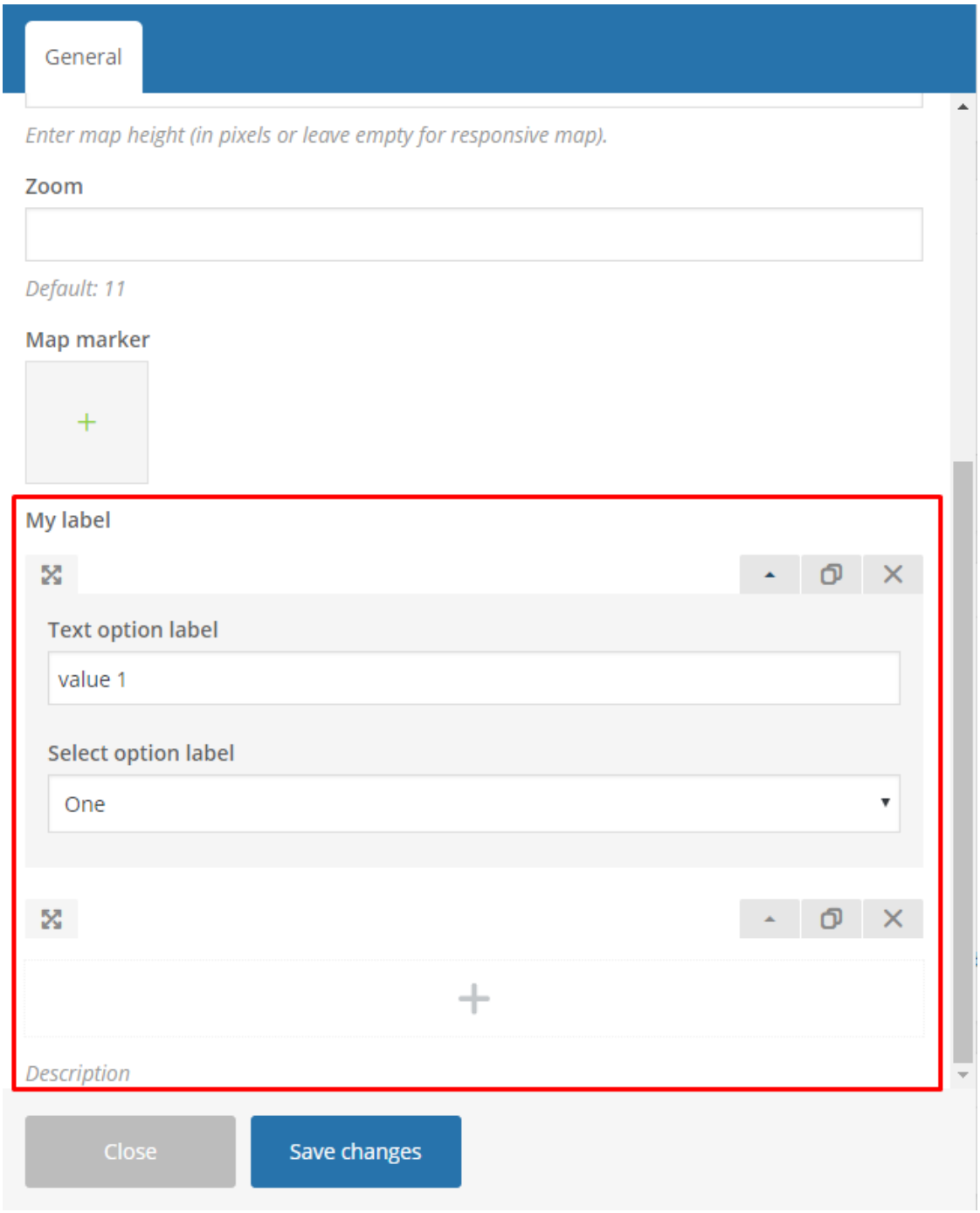
When Visual Composer extension is active, you can use 'addable-box' option type in VC shortcodes, which then will be converted to VC 'param_group' param type.

Please note the following options will not take effect on VC: *attr, template, box-controls, control-id, limit, add-button-text, sortable*

Example input from options.php:

```
'my_box'    => array(
        'type'        => 'addable-box',
        'value'       => array(     # preset values
                array(              # each array will create another set of values
                        'option_1' => 'value 1',
                        'option_2' => 'value 2',
                ),
                array(
                        'option_1' => 'another value 1',
                        'option_2' => 'another value 2',
                ),
        ),
        'tab'         => esc_html__( "General", 'ct_theme' ),
        'label'       => esc_html__( 'My label', 'ct_theme' ),
        'desc'        => esc_html__( 'Description', 'ct_theme' ),
        'box-options' => array(     # inner options
                'option_1' => array(
                        'type'  => 'text',
                        'label' => esc_html__("Text option label", 'ct_theme' ),
                ),
                'option_2' => array(
                        'type' => 'select',
                        'label' => esc_html__('Select option label', 'ct_theme' ),
                        'choices' => array(
                                'one'  => esc_html__( 'One', 'ct_theme' ),
                                'two'  => esc_html__( 'Two', 'ct_theme' ),
                        )
                ),
        ),
),
```

The option will render as follows in admin:



In view file just use a foreach to go through values:

```
<?php

foreach( $atts['my_box'] as $number => $value_set ) :
    echo $value_set['option_1'];
    echo $value_set['option_2'];
endforeach;

?>
```

Typography v2

Choose font family, style, weight, size, line-height, letter-spacing and color.

```
array(
    'type' => 'typography-v2',
    'value' => array(
        'family' => 'Amarante',
        // For standard fonts, instead of subset and variation you should set 'style'␣
↪and 'weight'.
        // 'style' => 'italic',
        // 'weight' => 700,
        'subset' => 'latin-ext',
        'variation' => 'regular',
        'size' => 14,
        'line-height' => 13,
        'letter-spacing' => -2,
        'color' => '#0000ff'
    ),
    'components' => array(
        'family'         => true,
        // 'style', 'weight', 'subset', 'variation' will appear and disappear along␣
↪with 'family'
        'size'           => true,
        'line-height'    => true,
        'letter-spacing' => true,
        'color'          => true
    ),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
)
```

WP Editor

Textarea with the WordPress Editor like the one you use on the blog posts edit pages.

```
array(
    'type'  => 'wp-editor',
    'value' => 'default value',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'size' => 'small', // small, large
    'editor_height' => 400,
    'wpautop' => true,
    'editor_type' => false, // tinymce, html

    /**
     * Also available
     * https://github.com/WordPress/WordPress/blob/4.4.2/wp-includes/class-wp-editor.
↪php#L80-L94
     */
)
```

Multi-Picker

Pick a choice, then complete options related to that choice.

The `picker` parameter holds a valid option type with choices. Supported option types are `select`, `radio`, `image-picker` and `switch`.

```
array(
    'type'  => 'multi-picker',
    'label' => false,
    'desc'  => false,
    'value' => array(
        /**
         * '<custom-key>' => 'default-choice'
         */
        'gadget' => 'phone',

        /**
         * These are the choices and their values,
         * they are available after option was saved to database
         */
        'laptop' => array(
            'price' => '123',
            'webcam' => false
        ),
        'phone' => array(
            'price' => '456',
            'memory' => '32'
        )
    ),
    'picker' => array(
        // '<custom-key>' => option
        'gadget' => array(
            'label'   => __('Choose device', '{domain}'),
            'type'    => 'select', // or 'short-select'
            'choices' => array(
                'phone'  => __('Phone', '{domain}'),
                'laptop' => __('Laptop', '{domain}')
            ),
            'desc'    => __('Description', '{domain}'),
            'help'    => __('Help tip', '{domain}'),
        )
    ),
    /*
    'picker' => array(
        // '<custom-key>' => option
        'gadget' => array(
            'label'   => __('Choose device', '{domain}'),
            'type'    => 'radio',
            'choices' => array(
                'phone'  => __('Phone', '{domain}'),
                'laptop' => __('Laptop', '{domain}')
            ),
            'desc'    => __('Description', '{domain}'),
            'help'    => __('Help tip', '{domain}'),
        )
    ),
    */
    /*
```

```
    'picker' => array(
        // '<custom-key>' => option
        'gadget' => array(
            'label'   => __('Choose device', '{domain}'),
            'type'    => 'image-picker',
            'choices' => array(
                'phone'  => 'http://placekitten.com/70/70',
                'laptop' => 'http://placekitten.com/71/70'
            ),
            'desc'    => __('Description', '{domain}'),
            'help'    => __('Help tip', '{domain}'),
        )
    ),
    */
    /*
    picker => array(
        // '<custom-key>' => option
        'gadget' => array(
            'label' => __('Choose device', '{domain}'),
            'type'  => 'switch',
            'right-choice' => array(
                'value' => 'laptop',
                'label' => __('Laptop', '{domain}')
            ),
            'left-choice' => array(
                'value' => 'phone',
                'label' => __('Phone', '{domain}')
            ),
            'desc' => __('Description', '{domain}'),
            'help' => __('Help tip', '{domain}'),
        )
    ),
    */
    'choices' => array(
        'phone' => array(
            'price' => array(
                'type'  => 'text',
                'label' => __('Price', '{domain}'),
            ),
            'memory' => array(
                'type'  => 'select',
                'label' => __('Memory', '{domain}'),
                'choices' => array(
                    '16' => __('16Gb', '{domain}'),
                    '32' => __('32Gb', '{domain}'),
                    '64' => __('64Gb', '{domain}'),
                )
            )
        ),
        'laptop' => array(
            'price' => array(
                'type'  => 'text',
                'label' => __('Price', '{domain}'),
            ),
            'webcam' => array(
                'type'  => 'switch',
                'label' => __('Webcam', '{domain}'),
            )
```

```
        ),
    ),
    /**
     * (optional) if is true, the borders between choice options will be shown
     */
    'show_borders' => false,
)
```

Get database option value

```
$value = fw_get_db_..._option(
    'option_id/'. fw_get_db_..._option('option_id/'. 'gadget')
);
```

Add support for new option type in picker

If you want to use in `picker` an option type that is not supported by default (is not present in the examples above), follow the steps below. In this example, is added support for `icon` option type *(it is not practical, just for demonstration purposes)*.

1. Add in `{theme}/inc/hooks.php`

```
/**
 * Generate array( 'choice_id' => array( Choice Options ) )
 * @internal
 * @param array $choices
 * @param array $data
 * @return array
 */
function _filter_theme_option_type_multi_picker_choices_icon($choices,
↪$data) {
    $choices = $data['option']['choices'];

    // maybe check and remove invalid choices ...

    return $choices;
}
add_filter(
    'fw_option_type_multi_picker_choices:icon',
    '_filter_theme_option_type_multi_picker_choices_icon',
    10, 2
);

/**
 * @internal
 */
function _admin_theme_multi_picker_custom_picker_scripts() {
    wp_enqueue_script(
        'multi-picker-custom-pickers',
        get_template_directory_uri() . '/js/multi-picker-custom-pickers.js
↪',
        array('fw-events'),
        false,
        true
```

```
    );
}
add_action(
    'admin_enqueue_scripts',
    '_admin_theme_multi_picker_custom_picker_scripts'
);
```

2. Add in `{theme}/js/multi-picker-custom-pickers.js`

```
fwEvents.on('fw:option-type:multi-picker:init:icon', function(data){
    data.$pickerGroup.find('.fw-option-type-icon > input[type="hidden"]').
↪on('change', function() {
        data.chooseGroup(
            this.value // this is `choice_id` from the `fw_option_type_
↪multi_picker_choices:{type}` filter (above)
        );
    }).trigger('change');
});
```

3. Add in `{theme}/framework-customizations/theme/options/settings.php`

```
$options = array(

'demo_multi_picker_icon' => array(
    'type'          => 'multi-picker',
    'label'         => false,
    'desc'          => false,
    'picker'        => array(
        'gadget' => array(
            'label'   => __( 'Multi Picker: Icon', 'unyson' ),
            'type'    => 'icon',
        )
    ),
    'choices' => array(
        'fa fa-btc'  => array(
            'price'  => array(
                'label' => __( 'Price', 'unyson' ),
                'type'  => 'slider',
                'value' => 70,
            ),
        ),
        'fa fa-viacoin' => array(
            'price'  => array(
                'label' => __( 'Price', 'unyson' ),
                'type'  => 'slider',
                'value' => 30
            ),
        ),
    ),
),

);
```

4. Open **Theme Settings** page and pick the Bitcoin or Viacoin icon.

Map

Google maps location.

```
array(
    'type'  => 'map',
    'value' => array(
        'coordinates' => array(
            'lat'   => -34,
            'lng'   => 150,
        )
    ),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
)
```

Multi

Group any options database values under a single array key. This option has no design, inner options will look the same as other options (it's like the `group` container).

```
// database value structure

'option_type_multi_id' => array(
    'inner_option_1' => ...
    'inner_option_2' => ...
)
```

```
array(
    'type'  => 'multi',
    'value' => array(
        'option-1' => 'value 1',
        'option-2' => 'value 2',
    ),
    'attr'  => array(
        'class' => 'custom-class',
        'data-foo' => 'bar',
        /*
        // Add this class to display inner options separators
        'class' => 'fw-option-type-multi-show-borders',
        */
    ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'inner-options' => array(
        'option_1' => array( 'type' => 'text' ),
        'option_2' => array( 'type' => 'textarea' ),
    )
)
```

Important: The parameter that contains options is named `inner-options` not `options` otherwise this will be

treated as a container option.

Hidden

Simple hidden input.

```
array(
    'type'  => 'hidden',
    'value' => 'default value',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
)
```

Tip: The hidden input is not visible, so parameters like `label`, `desc` and `help` have no sense here.

HTML

If you want to display a custom piece of html, use this option type.

Note: This option type has a value stored in a hidden input. Advanced users can create some javascript functionality in html and store the value in that hidden input.

```
array(
    'type'  => 'html',
    'value' => 'default hidden value',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'html'  => 'My <b>custom</b> <em>HTML</em>',
)
```

Note: There are `html-fixed` and `html-full` option types as well. They are the same as `html` but has **fixed** and **full** *option width*.

Password

Regular password input.

```
array(
    'type'  => 'password',
    'value' => 'default value',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
)
```

Oembed

Generate oembed preview of the inserted link, for more details see Embeds in WordPress.

```
array(
    'type'  => 'oembed',
    'value' => 'https://vimeo.com/113078377',
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
    'preview' => array(
        'width'  => 300, // optional, if you want to set the fixed width to iframe
        'height' => 300, // optional, if you want to set the fixed height to iframe
        /**
         * if is set to false it will force to fit the dimensions,
         * because some widgets return iframe with aspect ratio and ignore applied␣
↪dimensions
         */
        'keep_ratio' => true
    )
)
```

Typography

Choose font family, size, style and color.

```
array(
    'type'  => 'typography',
    'value' => array(
        'family' => 'Arial',
        'size'   => 12,
        'style'  => '400',
        'color'  => '#000000'
    ),
   'components' => array(
        'family' => true,
        'size'   => true,
        'color'  => true
    ),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
    'help'  => __('Help tip', '{domain}'),
)
```

Icon

Choose a FontAwesome icon.

```
array(
    'type'  => 'icon',
    'value' => 'fa-smile-o',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', '{domain}'),
    'desc'  => __('Description', '{domain}'),
```

```
    'help'  => __('Help tip', '{domain}'),
)
```

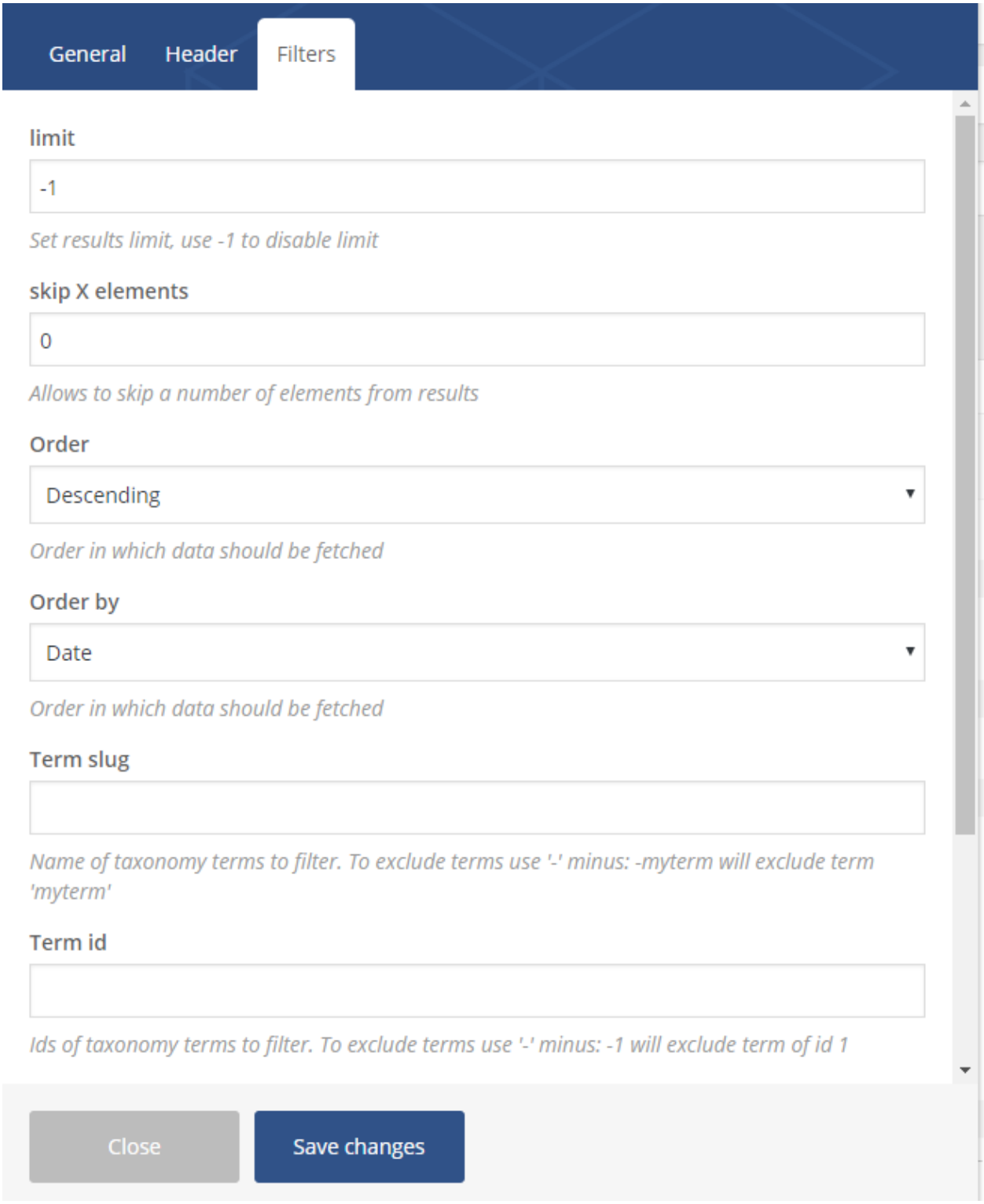
Queryable

Adds automatic filters for custom post types. Useful for creating shortcodes which allow users to easily filter collection (ex. select first 5 team members ordered by date descending)

Tip: This element belongs to Bumblebee Package.

```
'query' => array(
    'type' => 'queryable',
    'bee_query_post_type' => 'post',         # default: 'page'
    'bee_query_taxonomy' => 'custom_tax',    # default: 'category'
    'tab' => esc_html__( 'Filters', 'ct_theme' ),
),
```

Result in VC Editor:



Use in the shortcode view:

```
$posts_array = bee_query_posts( $atts );
```

Shortcode

Allows to add options from one shortcode inside another (by default into a separate tab). See description below for detailed usage.

Tip: This element belongs to Bumblebee Package.

```
$options = array(
    'my_related_shortcode' => //this name will be needed later inside view to render
↪it's content
            array(
                    'type' => 'shortcode',
                    'shortcode' => 'shortcode_name',
                    'tab' => __('My Shortcode', '{domain}'),
                    'options' => array() //we can override any shortcode option by
↪specifying options array('label'=>__("My new label",'ct_theme))
    )
)
```

Note: These type of shortcodes are called `Related Shortcodes`.

Above code will render a separate tab `My Shortcode` with all the options from `shortcode_name`. To easily render related shortcode, please use this snippet inside shortcode view:

```
<?php echo bee_shortcode_embed('my_related_shortcode', $atts, $tag, $content); ?>
```

Please note that `$atts`, `$tag` and `$content` are injected by BumbleBee automatically into the view. All you need to do is just adjust `my_related_shortcode` name which is taken from `$options` array.

Design

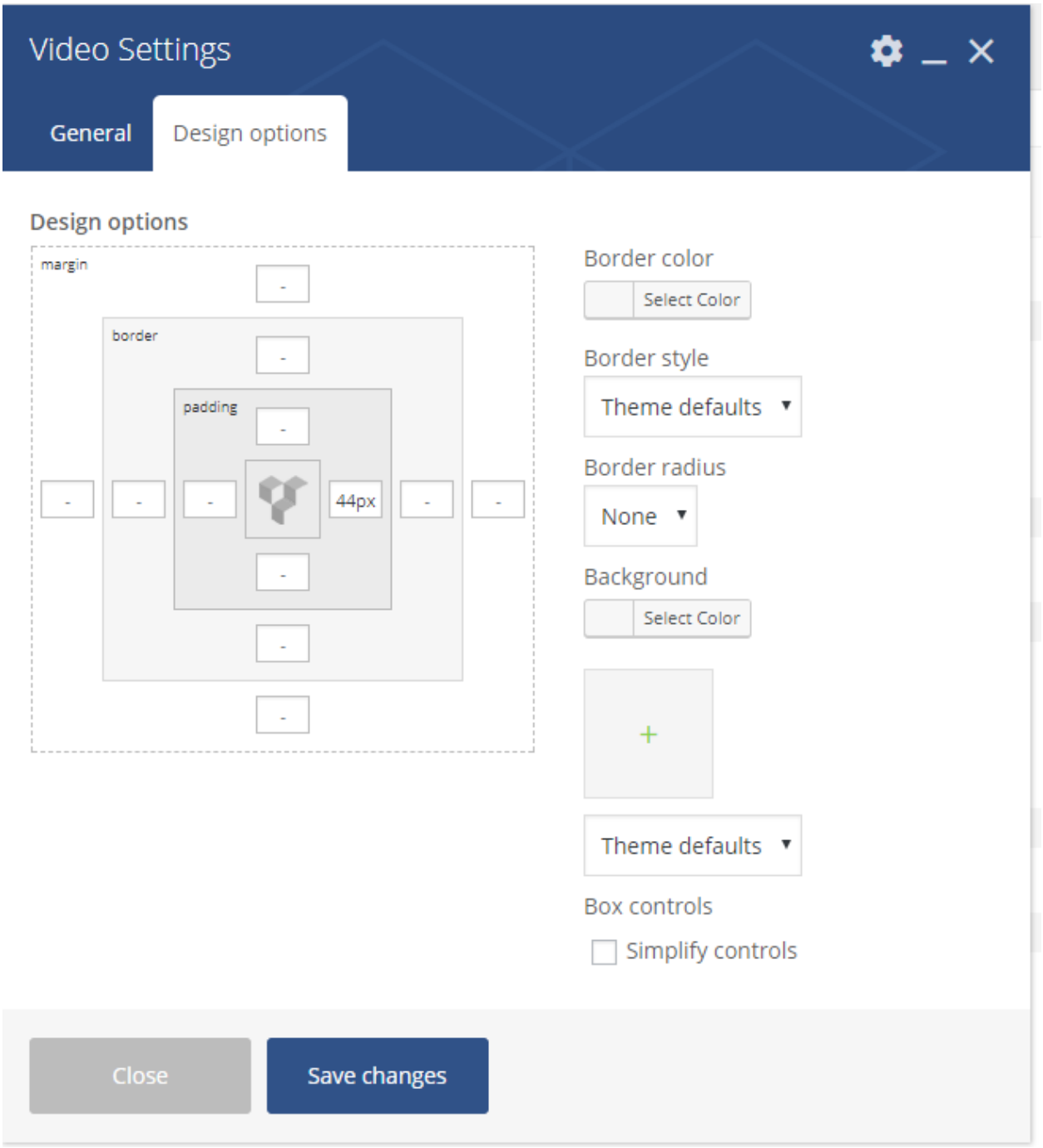
There's a simple way to add Visual Composer's CSS Editor to your shortcode. Option type `design` will be mapped to `css_editor` param.

Note: This feature requires the following Bumblebee extensions: Shortcodes, Visual Composer

Example:

```
'design' => array(
        'type'  => 'design',
        'tab'   => esc_html__( 'Design options', 'ct_theme' ),
        'label' => esc_html__( 'Design options', 'ct_theme' ), // optional
),
```

Result:



The view attribute will return a css style you can add to a shortcode container. Example:

```
<div class="<?php echo esc_attr( $atts['design'] ); ?>">

</div>
```

Some option types have custom javascript events. The events are triggered on elements with `.fw-option-type-{type}` class. Some events send data that can be accessed this way:

```
jQuery('.fw-option-type-demo#fw-option-demo')
    .on('fw:option-type:demo:custom-event', function(event, data){
```

```
        console.log(data);
    });
```

6.3.6 Create Option Type

To define a new option type, create a class that extends the base option type class and register it.

Note: It doesn't matter where you place your new option type. If you use the Theme Includes directory structure, place it in the `{theme}/inc/includes/option-types/my-option/` directory and include it on `fw_option_types_init` action:

```
// file: {theme}/inc/hooks.php

/** @internal */
function _action_theme_include_custom_option_types() {
    require_once dirname(__FILE__) . '/includes/option-types/new/class-fw-option-type-
↪new.php';
}
add_action('fw_option_types_init', '_action_theme_include_custom_option_types');
```

```
class FW_Option_Type_New extends FW_Option_Type
{
    public function get_type()
    {
        return 'new';
    }

    /**
     * @internal
     */
    protected function _enqueue_static($id, $option, $data)
    {
        $uri = get_template_directory_uri() .'/inc/includes/option-types/'. $this->
↪get_type() .'/static';

        wp_enqueue_style(
            'fw-option-'. $this->get_type(),
            $uri .'/css/styles.css'
        );

        wp_enqueue_script(
            'fw-option-'. $this->get_type(),
            $uri .'/js/scripts.js',
            array('fw-events', 'jquery')
        );
    }

    /**
     * @internal
     */
    protected function _render($id, $option, $data)
    {
        /**
         * $data['value'] contains correct value returned by the _get_value_from_
↪input()
```

```
         * You decide how to use it in html
         */
        $option['attr']['value'] = (string)$data['value'];

        /**
         * $option['attr'] contains all attributes.
         *
         * Main (wrapper) option html element should have "id" and "class" attribute.
         *
         * All option types should have in main element the class "fw-option-type-{
↪$type}".
         * Every javascript and css in that option should use that class.
         *
         * Remaining attributes you can:
         *  1. use them all in main element (if option itself has no input elements)
         *  2. use them in input element (if option has input element that contains␣
↪option value)
         *
         * In this case you will use second option.
         */

        $wrapper_attr = array(
            'id'    => $option['attr']['id'],
            'class' => $option['attr']['class'],
        );

        unset(
            $option['attr']['id'],
            $option['attr']['class']
        );

        $html  = '<div '. fw_attr_to_html($wrapper_attr) .'>';
        $html .= '<input '. fw_attr_to_html($option['attr']) .' type="text" />';
        $html .= '<button type="button" class="button">'. __('Clear text', '{domain}
↪') .'</button>';
        $html .= '</div>';

        return $html;
    }

    /**
     * @internal
     */
    protected function _get_value_from_input($option, $input_value)
    {
        /**
         * In this method you receive $input_value (from form submit or whatever)
         * and must return correct and safe value that will be stored in database.
         *
         * $input_value can be null.
         * In this case you should return default value from $option['value']
         */

        if (is_null($input_value)) {
            $input_value = $option['value'];
        }

        return (string)$input_value;
```

```
    }

    /**
     * @internal
     */
    protected function _get_defaults()
    {
        /**
         * These are default parameters that will be merged with option array.
         * They makes possible that any option has
         * only one required parameter array('type' => 'new').
         */

        return array(
            'value' => ''
        );
    }
}

FW_Option_Type::register('FW_Option_Type_New');
```

```
/**
 * Prefix (namespace) all css rules with ".fw-option-type-{$option_type}"
 * This guarantees that there will be no conflicts with other styles.
 */

.fw-option-type-new input {
    background-color: green;
    color: white;
}

.fw-option-type-new button {
    display: block;
}
```

```
jQuery(document).ready(function ($) {
    var optionTypeClass = '.fw-option-type-new';

    /**
     * Listen to special event that is triggered for uninitialized elements
     */
    fwEvents.on('fw:options:init', function (data) {
        /**
         * data.$elements are jQuery selected elements
         * that contains options html that needs to be initialized
         *
         * Find uninitialized options by main class
         */
        var $options = data.$elements.find(optionTypeClass +':not(.initialized)');

        /**
         * Listen for button click and clear input value
         */
        $options.on('click', 'button', function(){
            $(this).closest(optionTypeClass).find('input').val('');
        });
```

```
        /**
         * After everything has done, mark options as initialized
         */
        $options.addClass('initialized');
    });
});
```

Option Width

There are three width types:

- **auto** - dynamically adapted to the contents of the option.
- **fixed** - fixed size (it doesn't matter what size, it's just fixed).
- **full** - full available width (100%).

Every option has its own width type specified in `FW_Option_Type::_get_backend_width_type()`.

6.3.7 Custom Save Location

Introduction

By default *(post, settings, customizer, term and other)* options are saved all in one place in database, in a wp_option or any other location. In some cases you need an option to be saved in a separate wp_option or post meta or some custom location in database, for example Mailer settings option-type is saved in one wp_option and the same value is used in all Contact Forms. This custom saving behavior is achieved via the `fw-storage` option parameter, it has the following structure:

```
'option_id' => array(
    'type' => 'any-type',

    'fw-storage' => array(
        'type' => 'valid-storage-type',
        // additional parameters of the used storage type
    ),
)
```

When options are saved and loaded from database all their `fw-storage` parameters are processed.

Predefined types

Here are some examples with default storage types:

1. To save an option in a separate wp_option:

```
'demo-option-id' => array(
    'type' => 'text',

    'fw-storage' => array(
        'type' => 'wp-option',
        'wp-option' => 'demo_wp_option',
    ),
)
```

Add the above option array in post, settings, customizer or term options, save the form and check the database `wp_options` table for an option named `demo_wp_option`.

Additional parameters can be found here.

2. To save an option in a separate post meta:

```
'demo-option-id' => array(
    'type' => 'text',

    'fw-storage' => array(
        'type' => 'post-meta',
        'post-meta' => 'demo_post_meta',
    ),
)
```

Add the above option array in **post options**, edit a post and check the database `wp_postmeta` table for a meta named `demo_post_meta`.

Additional parameters can be found here.

Custom Types

It's possible to register new storage types.

1. Create your class in a separate file and extend the `FW_Option_Storage_Type` class.

   Let's say the file is `{theme}/class-fw-option-storage-type-demo.php`.

```
class FW_Option_Storage_Type_Demo extends FW_Option_Storage_Type {
    public function get_type() {
        return 'demo';
    }

    protected function _load($id, array $option, $value, array $params) {
        if ($param_id = $this->get_parameter_value($id, $option,
↪$params)) {
            // the parameter was specified
            return $this->load_value($param_id);
        } else {
            // do nothing, return the current value
            return $value;
        }
    }

    protected function _save($id, array $option, $value, array $params) {
        if ($param_id = $this->get_parameter_value($id, $option,
↪$params)) {
            $this->save_value($param_id, $value);
```

```
            // do not return current value to prevent duplicate and␣
↪useless memory usage
            // return empty default option-type value
            return fw()->backend->option_type($option['type'])->get_value_
↪from_input(
                array('type' => $option['type']), null
            );
        } else {
            // do nothing, return the current value
            return $value;
        }
    }

    /**
     * Check and extract the identification parameter
     * @param string $id
     * @param array $option
     * @param array $params
     * @return string|bool
     */
    private function get_parameter_value($id, $option, $params) {
        if (isset($option['fw-storage']['demo-id'])) {
            return $option['fw-storage']['demo-id'];
        } else {
            return false;
        }
    }

    private function load_value($param_id) {
        // Load the value from your custom location (a wp_option, a␣
↪custom table, etc.)
        $value = 'Hello World'; // ...

        return $value;
    }

    private function save_value($param_id, $value) {
        // Save the value to your custom location...
    }
}
```

Note: The class implementation is simplified just to give you an idea of how it works. For a complete implementation inspect the predefined types.

2. Register your custom type. Add in `{theme}/functions.php`:

```
add_action(
    'fw:option-storage-types:register',
    '_action_theme_custom_fw_storage_types'
);
function _action_theme_custom_fw_storage_types($register) {
    require_once dirname(__FILE__) .'/class-fw-option-storage-type-demo.
↪php';
    $register->register(new FW_Option_Storage_Type_Demo());
}
```

3. Use your custom type in any option:

```
'some-option-id' => array(
    'type' => 'text',

    'fw-storage' => array(
        'type' => 'demo', // Must match FW_Option_Storage_Type_Demo::get_
↪type()
        'demo-id' => 'lorem-ipsum', // This is used by FW_Option_Storage_
↪Type_Demo::get_parameter_value()
    ),
)
```

6.4 Built-in Extensions

6.4.1 Introduction

The Unyson framework comes with the following built-in extensions:

- *Shortcodes*
- *Slider*
- *Mega Menu*
- *Sidebars*
- *Portfolio*
- *Backup & Demo Content*
- *Forms*
- *Breadcrumbs*
- *SEO*
- *Events*
- *Social*
- *Builder*
- *Feedback*
- *Learning*
- *Translation*
- *WordPress Shortcodes*

6.4.2 Shortcodes

The shortcodes extension makes possible the easy creation of WordPress Shortcodes and their optional integration with the framework's page builder.

Built-in shortcodes

Unyson comes with a set of built-in shortcodes like `accordion`, `button`, `map`, `testimonials` and others. All shortcodes are located in `{some-extension}/shortcodes/` but the vast majority of them are located in the shortcodes extension (`framework/extensions/shortcodes/shortcodes`). They can be modified by *overwriting* or *disabled*

Overwriting shortcodes

Some shortcode files can be overwritten (meaning that the files can be swapped). This permits shortcode customization. The files that can be overwritten are *config file*, *options file*, *static file* and *view file*.

There are three places where the shortcode files are searched until found: child theme (if active), parent theme and framework.

- If the shortcode is built-in (declared in the framework) the files will be first looked up in the child theme (if active), after that in the parent theme, and in the end the framework will search in the shortcode's declared path
- If the shortcode is declared in the parent theme the files will be first searched in the child theme (if active) and then in the parent theme (where it was declared)
- If the shortcode is declared in the child theme the files will be searched only at it's declared path

For a better understanding let's look at an example: Imagine that there is a shortcode `demo` located in the shortcodes extension (`framework/extensions/shortcodes/shortcodes/demo`). When the framework loads it's

files (`options.php` for this example) it will follow these simple steps:

1. If a child theme is active it will first look in `{your-child-theme}/framework-customizations/extensions/shortcodes/shortcodes/demo/options.php`
2. If it did not find the file in the child theme it will search in `{your-parent-theme}/framework-customizations/extensions/shortcodes/shortcodes/demo/options.php`
3. If it did not find the file in the parent theme it will search at the shortcode's declared path `framework/extensions/shortcodes/shortcodes/demo/options.php`

Disabling shortcodes

A shortcode can be disabled via the `fw_ext_shortcodes_disable_shortcodes` filter. A good place to put the code for the disabling would be in `{your-theme}/framework-customizations/extensions/shortcodes/hooks.php`. It should look something like the following:

```
<?php if (!defined('FW')) die('Forbidden');

function _filter_theme_disable_default_shortcodes($to_disable) {
    $to_disable[] = 'accordion';
    $to_disable[] = 'button';

    return $to_disable;
}
add_filter('fw_ext_shortcodes_disable_shortcodes', '_filter_theme_disable_default_
↪shortcodes');
```

Creating a new shortcode

If *overwriting* a built-in shortcode does not suit your needs then you might want to create a new shortcode. For that you will first have to decide where to place it:

- If you are developing a unyson extension and you want to offer some functionality from the extension via a shortcode you should create the it at `framework-customizations/extensions/{your-extension}/shortcodes/{your-shortcode}`. One such example from the built-in extensions is the slider extension and it's shortcode.
- If the shortcode that you want to create is not extension specific but more generalist (like the `button`, `tabs` ones are) than you should place it the shortcodes extensions (`framework-customizations/extensions/shortcodes/shortcodes/{your-shortcodes}`).

Directory structure

```
{shortcode-name}
├──class-fw-shortcode-{shortcode-name}.php # optional
├──config.php # optional
├──options.php # optional
├──static.php # optional
├──static # optional
│   ├──css # you can put your css files here
│   ├──img
│   │   └──page_builder.png # used as the page builder icon
│   └──js # you can put your js files here
```

```
└──views
    └──view.php
```

Attention: The directory name of the shortcode folder will become it's tag, hyphens will be replaced with underscores. This means that if you name the shortcode `demo-shortcode` it will be transformed into `[demo_shortcode]`.

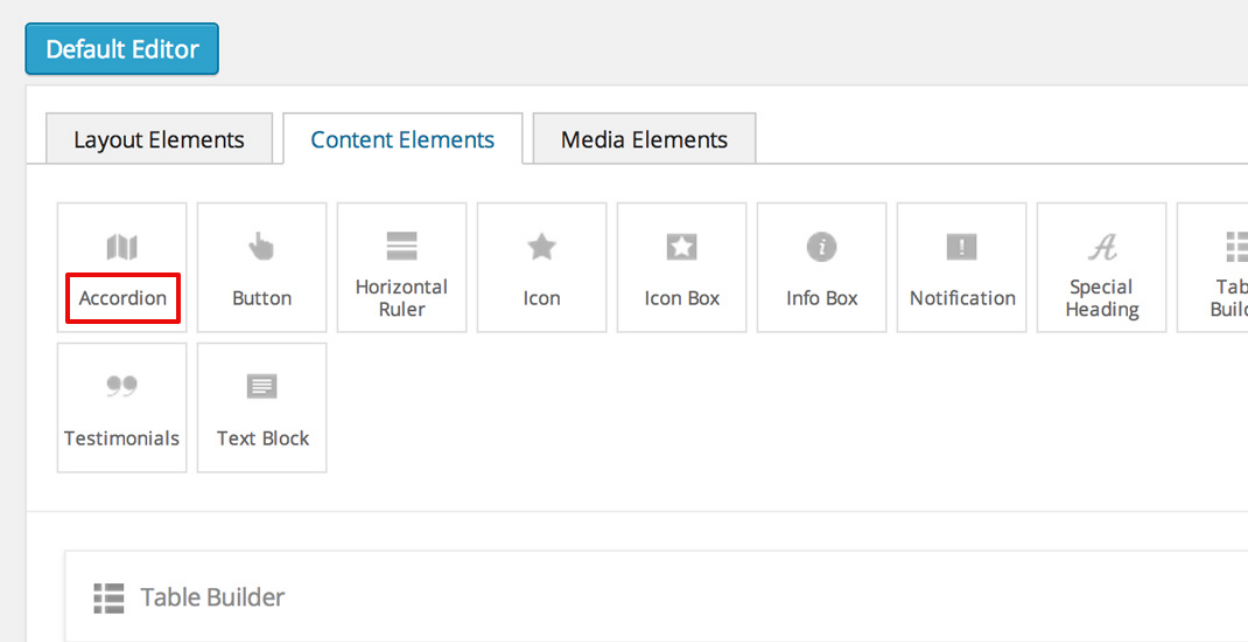
Config File

The shortcode configuration is a file named `config.php` placed inside the root directory of the shortcode. It contains an array that must be stored in a `$cfg` variable and is typically used to provide configurations for the visual page builder.

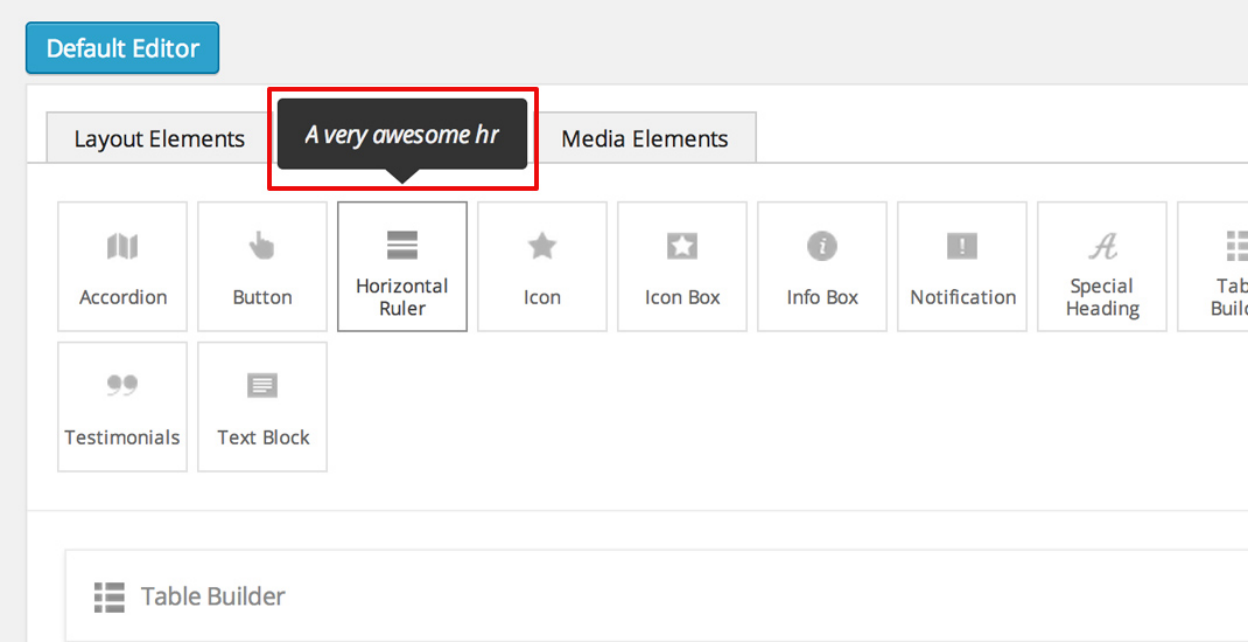
```
$cfg = array(
    'page_builder' => array(
        'title'         => __('Demo Shortcode', '{domain}'),
        'description'   => __('Demo shortcode description', '{domain}'),
        'tab'           => __('Demo Elements', '{domain}'),
        'popup_size'    => 'small', // can be large, medium or small

        /*
        // Icon examples
        // Note: By default is loaded {your-shortcode}/static/img/page_builder.png
        'icon' => 'http://.../image-16x16.png', // background color should be #8c8c8c
        'icon' => 'dashicons dashicons-admin-site',
        'icon' => 'unycon unycon-crown',
        'icon' => 'fa fa-btc',
        */

        /*
        // Title Template examples
        //
        // Syntax:
        // * {{- variable }} - Output with html escape
        // * {{= variable }} - Output raw (without html escape)
        // * {{ if (execute.any(javascript, code)) { console.log('Hi'); } }}
        //
        // Available variables:
        // * title - shortcode title (from config)
        // * o - an object with all shortcode options values
        'title_template' => '{{- title }} Lorem {{- o.option_id }} ipsum {{= o[
↪"option-id"] }}',
        'title_template' => '{{- title }}: {{- o.label }}{{ if (o.target == "_blank")
↪{ }} <span class="dashicons dashicons-external"></span>{{ } }}',
        */
    )
);
```

For the shortcode to appear in the page builder the config array contains a special `page_builder` key that holds an array with the following data:

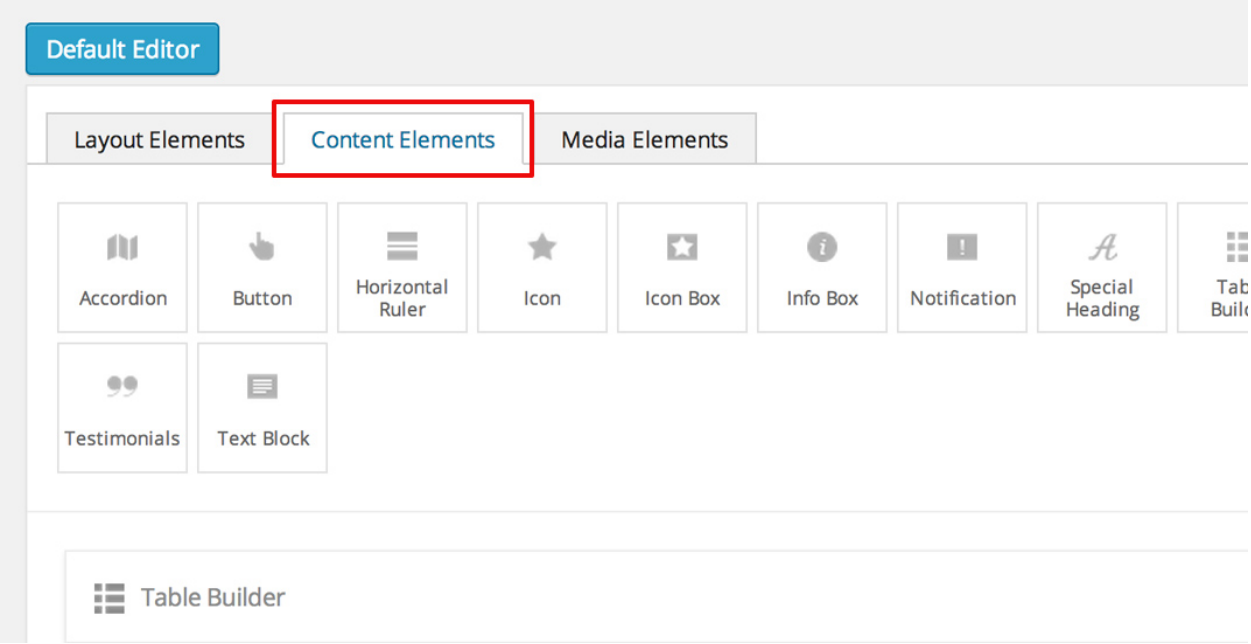
- `title` - the title that will appear in the shortcode box.



- `description` - the text that will be shown in a tooltip when hovering the shortcode box.

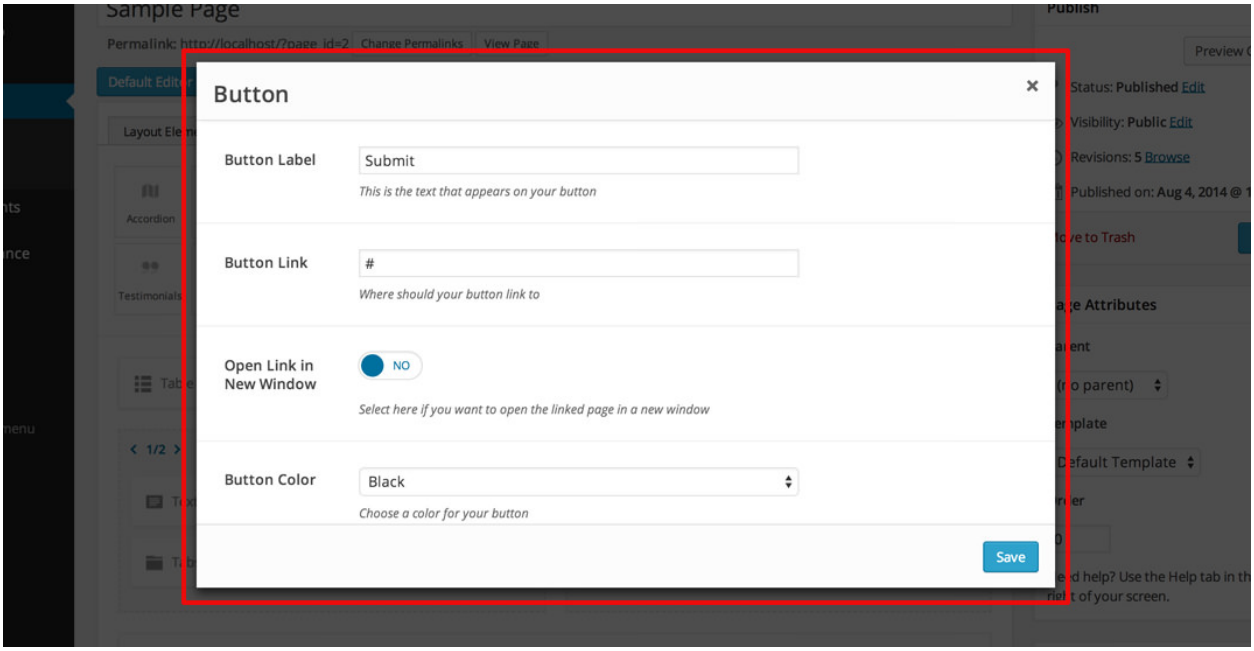


- `tab` - the builder tab in which the shortcode box will appear.



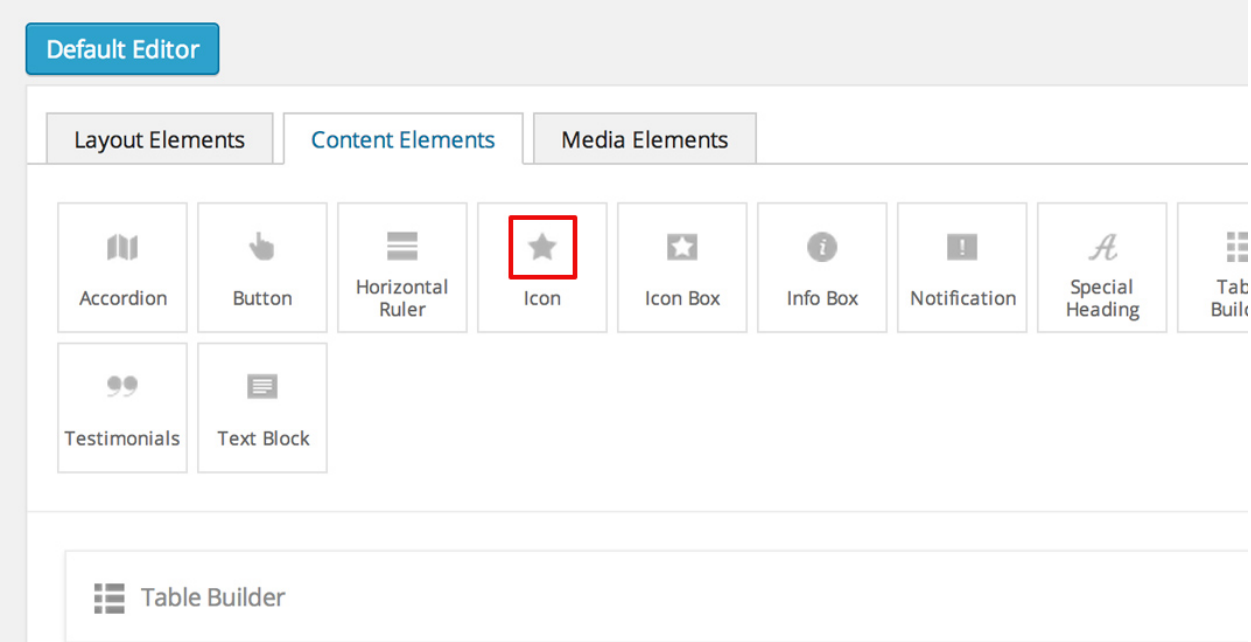
- `popup_size` - the size of the popup in which the *shortcode options* will be displayed.

  Allowed values are `large | medium | small`. This parameter is optional and the default is set to `small`.



Builder icon

To set an icon for the shortcode box, put an image named `page_builder.png` inside `{your-shortcode}/static/img/` directory. The image should have the size of 16x16 px.

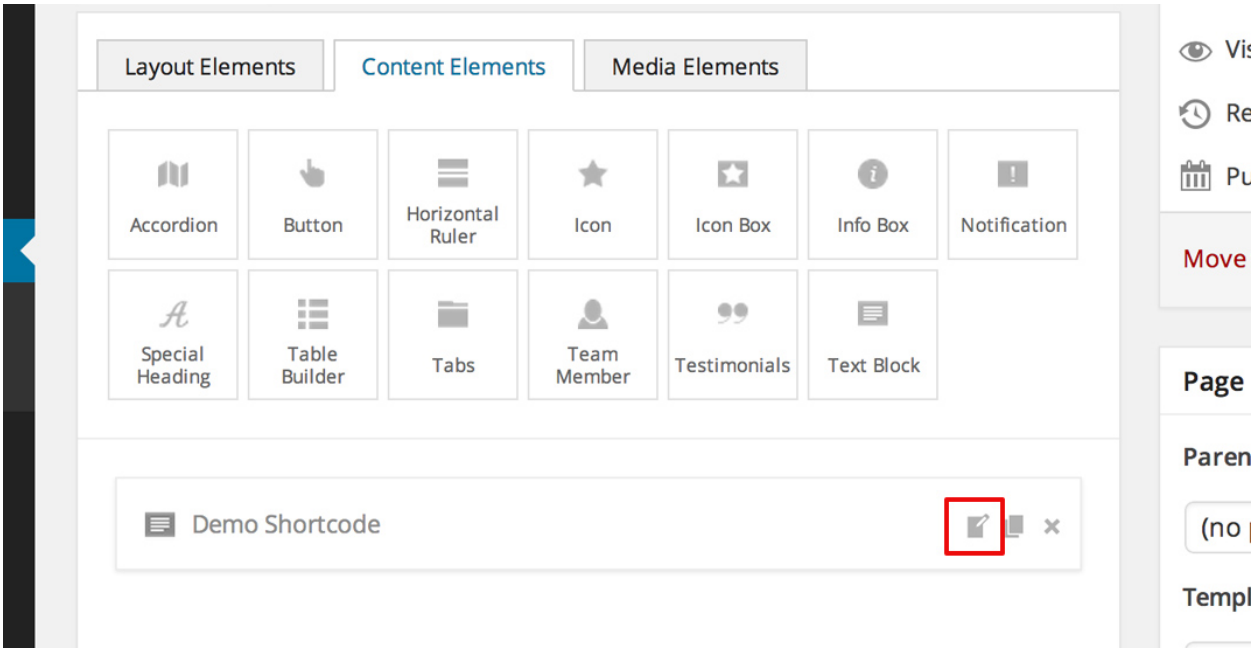


Options file

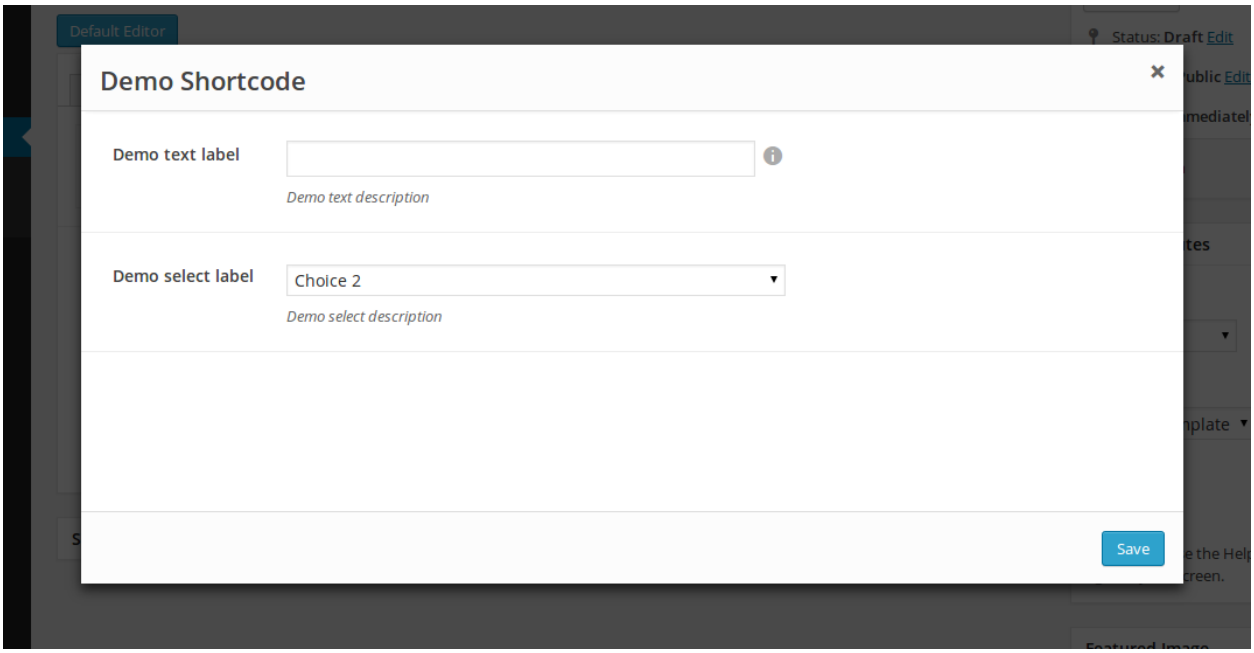
The shortcode directory can contain a file named `options.php` with correctly formed options:

```
$options = array(
    'demo_text'   => array(
        'label'   => __('Demo text label', '{domain}'),
        'desc'    => __('Demo text description', '{domain}'),
        'help'    => __('Demo text help', '{domain}'),
        'type'    => 'text'
    ),
    'demo_select' => array(
        'label'   => __('Demo select label', '{domain}'),
        'desc'    => __('Demo select description', '{domain}'),
        'type'    => 'select',
        'choices' => array(
            'c1'  => __('Choice 1', '{domain}'),
            'c2'  => __('Choice 2', '{domain}'),
            'c3'  => __('Choice 3', '{domain}')
        ),
        'value'   => 'c2'
    )
);
```

If it does, then it will have an icon when dragged into the builder's canvas area, indicating that the shortcode can be edited:



When clicking either the edit icon or the shortcode itself, a modal window will open containing the declared options:



The saved options values will be passed into the *view file*.

Default view file

By default, when WordPress wants to render a shortcode built into the framework, it will serve the html from the default view file located in `{your-shortcode}/views/view.php`. **3 variables** are passes into the view file : `$atts`, `$content` and `$tag`.

Tip: More information can be found in the *cookbook section*.

Static file

A shortcode can have a `static.php` file that is included when the shortcode is rendered. It is meant for enqueuing static files. Here is an example of a basic *static.php* file:

```
<?php if (!defined('FW')) die('Forbidden');

// find the uri to the shortcode folder
$uri = fw_get_template_customizations_directory_uri('/extensions/shortcodes/
↪shortcodes/demo-shortcode');
wp_enqueue_style(
    'fw-shortcode-demo-shortcode',
    $uri . '/static/css/styles.css'
);
wp_enqueue_script(
    'fw-shortcode-demo-shortcode',
    $uri . '/static/js/scripts.js'
);
```

If you want to include custom styles and scripts for a existing shortcode, overwrite the `static.php` file by creating `framework-customizations/extensions/shortcodes/shortcodes/demo-shortcode/static.php`.

Attention: All of the above is valid only in the case that the `_render` method from the *class file* was not overwritten.

Class file

When creating a shortcode folder with all the required files, the framework makes an instance of `FW_Shortcode` to ensure the correct default functionality, some of which default functionality can be overwritten by creating a class in the shortcode directory that extends `FW_Shortcode`.

Note: The class file must respect the following naming convention: `class-fw-shortcode-{your-shortcode-folder-name}.php`.

The class inside the class file must respect the following naming convention: `FW_Shortcode_{Your_Shortcode_Folder_Name}`.

Replace the hyphens with underscores in the class name.

Note: The framework replaces hyphens with underscores when registering the shortcode, so `your-shortcode` will be transformed to `[your_shortcode]`.

So in order to create a class for the `[demo_shortcode]` shortcode, we need to create a file `demo-shortcode/class-fw-shortcode-demo-shortcode.php` and within the file create a class that extends `FW_Shortcode`:

```
class FW_Shortcode_Demo_Shortcode extends FW_Shortcode
{
    // ...
}
```

The new class inherits some usefull methods like:

- `get_tag()` - returns the shortcode's tag.
- `get_declared_path($rel_path = '')` - returns the path to where the shortcode folder was declared.
- `get_declared_URI($rel_path = '')` - returns the uri to where shortcode folder was declared.
- `locate_path($rel_path = '')` - searches a rel path given as an argument first in child theme then in parent theme and last in framework. Returns the found path or false if not found. See *overwriting* for more details.
- `locate_URI($rel_path = '')` - does the same as *locate_path* with uris.
- `get_config($key = null)` - returns the shortcode's whole *overwritten* config array, or just a particular key of it's given as an argument.
- `get_options()` - returns the shortcode's *overwritten* options array, if there is any.

The methods that are most prone to be overwritten are:

- `_init()` - is called when the `FW_Shortcode` instance for the shortcode is created. Useful for loading other php files (custom option types, libraries, etc.).
- `_render($atts, $content, $tag)` - returns the html that will be displayed when the shortcode will be executed by WordPress. Useful for changing the default behavior with a custom one.

Tip: More information about this can be found in the *cookbook section*.

Cookbook

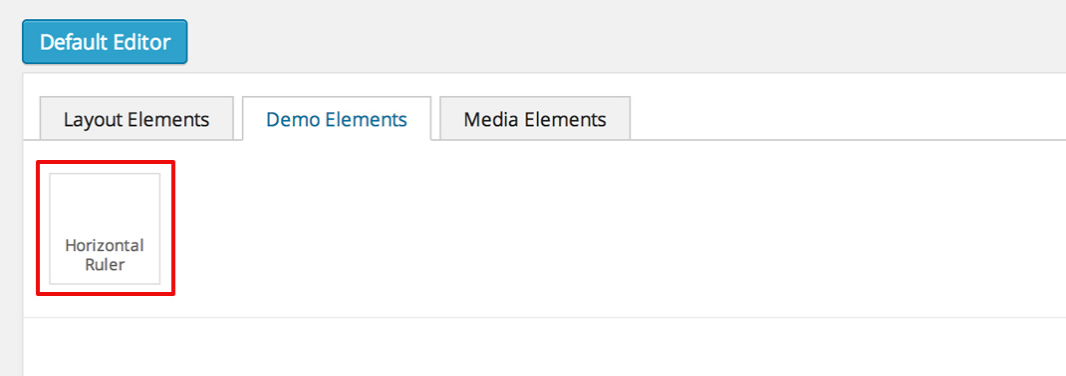
Creating a simple shortcode

This example will go through creating the `[hr]` (horizontal ruler) shortcode in a few simple steps:

1. Create a `hr` folder in `framework-customizations/extensions/shortcodes/shortcodes/`.
2. Create a *config file* inside that folder:

```
<?php if (!defined('FW')) die('Forbidden');

$cfg = array(
    'page_builder' => array(
        'title'       => __('Horizontal Ruler', '{domain}'),
        'description' => __('Creates a \'hr\' html tag', '{domain}'),
        'tab'         => __('Demo Elements', '{domain}'),
    )
);
```

Note: At this point the shortcode should appear in the **Demo Elements** tab of the layout builder as shown below:



Tip: To add an icon to the shortcode see the *icon section*.

3. Create a views folder and the *view file* inside it:

```
<?php if (!defined('FW')) die('Forbidden'); ?>

<hr>
```

The `[hr]` shorcode is completed! The directory structure of the shortcode is as shown below:

```
framework-customizations/
└─theme/
  └─shortcodes/
    └─hr/
      ├─config.php
      └─views/
        └─view.php
```

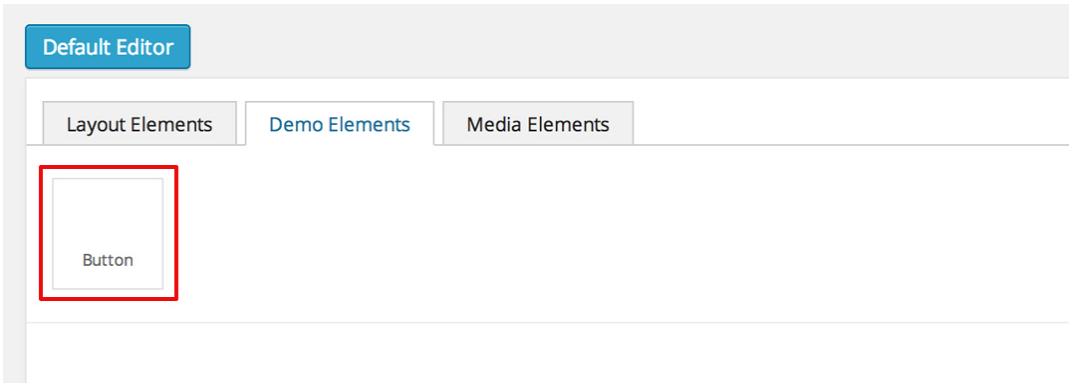
Creating a shortcode with options

This example will go through creating the `[button]` shortcode.

1. Create a `button` folder in `framework-customizations/extensions/shortcodes/shortcodes/`
2. Create a *config file* inside that folder:

```
<?php if (!defined('FW')) die('Forbidden');

$cfg = array(
    'page_builder' => array(
        'title'         => __('Button', '{domain}'),
        'description'   => __('Creates a button with choosable label,␣
↪size and style', '{domain}'),
        'tab'           => __('Demo Elements', '{domain}'),
    )
);
```

Note: At this point the shortcode should appear in the **Demo Elements** tab of the layout builder as shown below:

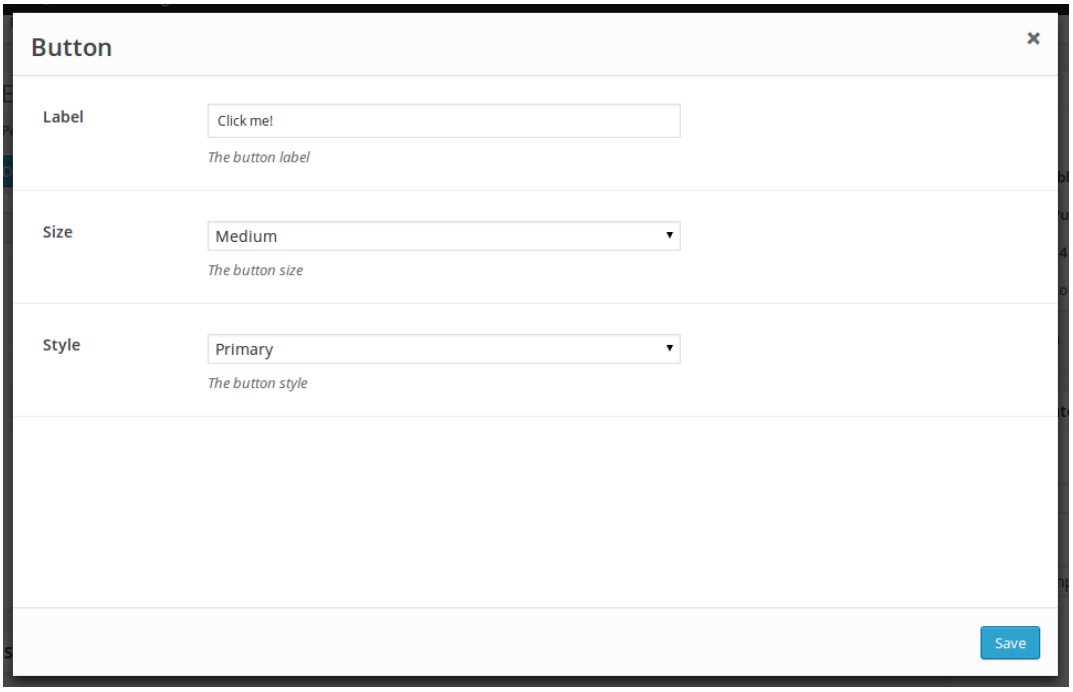


Tip: To add an icon to the shortcode see the *icon section*.

3. Create an *options file* with the options for **label**, **size** and **style**:

```
<?php if (!defined('FW')) die('Forbidden');

$options = array(
    'label' => array(
        'label'   => __('Label', '{domain}'),
        'desc'    => __('The button label', '{domain}'),
        'type'    => 'text',
        'value'   => __('Click me!', '{domain}')
    ),
    'size' => array(
        'label'   => __('Size', '{domain}'),
        'desc'    => __('The button size', '{domain}'),
        'type'    => 'select',
        'choices' => array(
            'big'    => __('Big', '{domain}'),
            'medium' => __('Medium', '{domain}'),
            'small'  => __('Small', '{domain}')
        ),
        'value'   => 'medium'
    ),
    'style' => array(
        'label'   => __('Style', '{domain}'),
        'desc'    => __('The button style', '{domain}'),
        'type'    => 'select',
        'choices' => array(
            'primary'   => __('Primary', '{domain}'),
            'secondary' => __('Secondary', '{domain}')
        )
    )
);
```

Now, when clicking the shortcode inside the canvas area of the layout builder a pop-up window containting the options will appear:



4. Create a views folder and the *view file* inside it. Make use of the `$atts` variable that is avaialble inside the view, it contains all the options values that the user has selected in the pop-up:

```
<?php if (!defined('FW')) die('Forbidden'); ?>

<button class="button button-<?php echo $atts['size']; ?> button-<?php␣
↪echo $atts['style']; ?>">
    <?php echo $atts['label']; ?>
</button>
```

Tip: For more information about the view variables check out the *default view section*.

The `[button]` shorcode is completed! The directory structure of the shortcode is as shown below:

```
framework-customizations/
└─theme/
  └─shortcodes/
    └─button/
      ├─config.php
      ├─options.php
      └─views/
        └─view.php
```

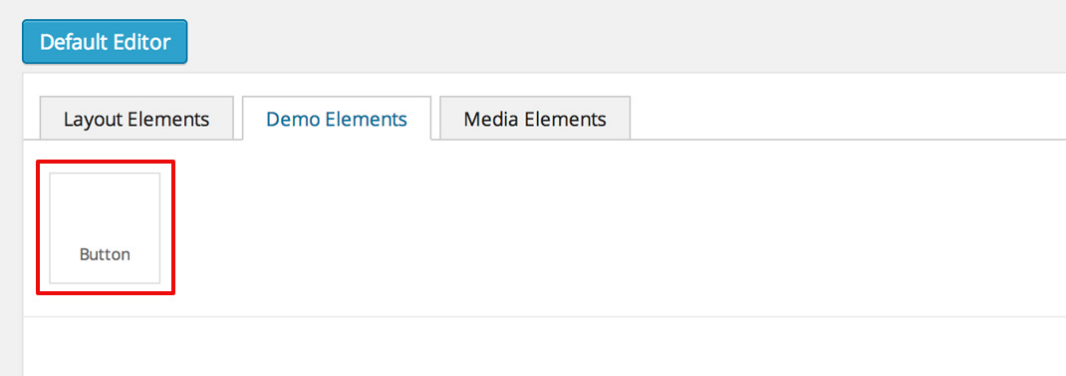
Creating a shortcode with related shortcodes

This example will go through creating the `[icon_box]` shortcode. This shortcode will use other `shortcodes` (aka `related shortcodes`) to reuse available components and speed up development time.

1. Create a `icon_box` folder in `framework-customizations/extensions/shortcodes/shortcodes/`
2. Create a *config file* inside that folder:

```
<?php if (!defined('FW')) die('Forbidden');

$cfg = array(
    'page_builder' => array(
        'title'         => __('Button', '{domain}'),
        'description'   => __('Creates a button with choosable label,
↪size and style', '{domain}'),
        'tab'           => __('Demo Elements', '{domain}'),
    )
);
```

Note: At this point the shortcode should appear in the **Demo Elements** tab of the layout builder as shown below:



Tip: To add an icon to the shortcode see the *icon section*.

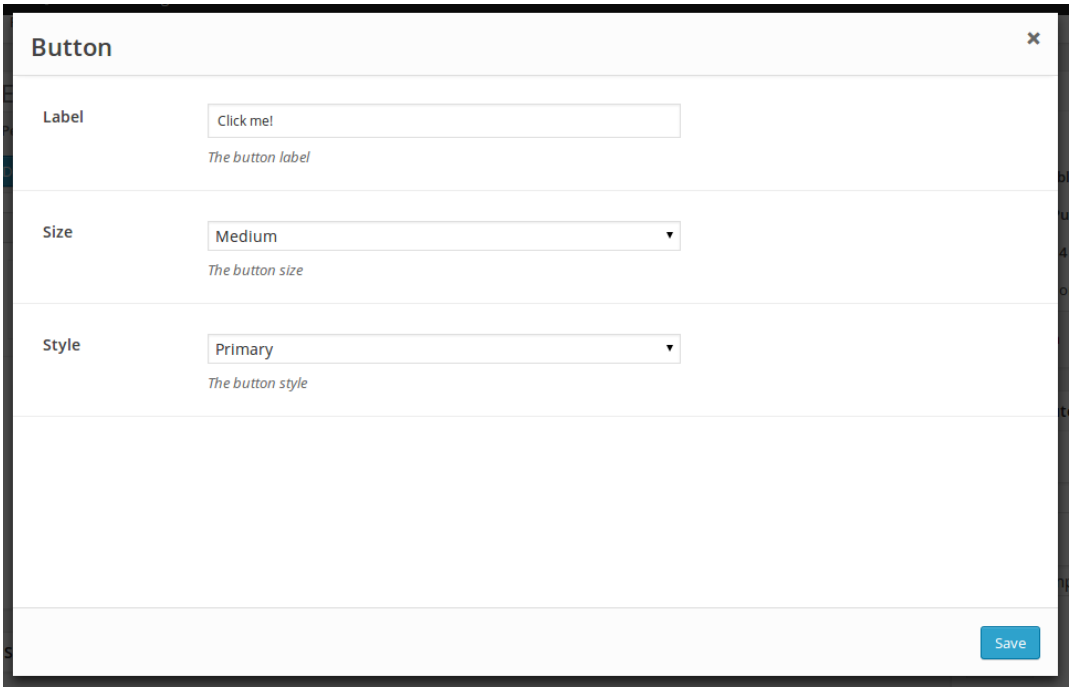
3. Create an *options file* with the options for **header**, **content**, **icon** and **button**:

```
<?php if (!defined('FW')) die('Forbidden');

$options = array(
    'header' => array(
        'label'   => __('Header', '{domain}'),
        'desc'    => __('The box header', '{domain}'),
        'type'    => 'text',
    ),
    'content' => array(
        'label'   => __('Box content', '{domain}'),
        'type'    => 'textarea',
    ),
    'my_icon' => array(
        'label'     => __('Icon', '{domain}'),
        'type'      => 'shortcode',
        'shortcode' => 'ct_icon'
        'tab'       => 'Icon tab label'
    ),
    'my_button' => array(
        'label'     => __('Button', '{domain}'),
        'type'      => 'shortcode',
        'shortcode' => 'button'
        'tab'       => 'More button'
```

```
    ),
);
```

Please note that we used special param type `shortcode`. This allows us to render `related shortcode` params inside a `tab` with specified label.

Now, when clicking the shortcode inside the canvas area of the layout builder a pop-up window containting the options will appear:



4. Create a views folder and the *view file* inside it. Make use of the `$atts` variable that is avaialble inside the view, it contains all the options values that the user has selected in the pop-up:

```
<?php if (!defined('FW')) die('Forbidden'); ?>

<div class="ct-icon-box">
    <h1><?php echo $atts['header']?></h1>

    <?php echo bee_shortcode_embed('my_icon',$atts, $tag, $content)?>

    <div class="content">
        <?php echo $content ?>
    </div>

    <?php echo bee_shortcode_embed('my_button',$atts, $tag, $content)?>

</div>
```

Tip: `bee_shortcode_embed` is a Bee extension. For more information about the view variables check out the *default view section*.

The `[button]` shorcode is completed! The directory structure of the shortcode is as shown below:

```
framework-customizations/
└─theme/
  └─shortcodes/
    └─button/
      ├─config.php
      ├─options.php
      └─views/
        └─view.php
```

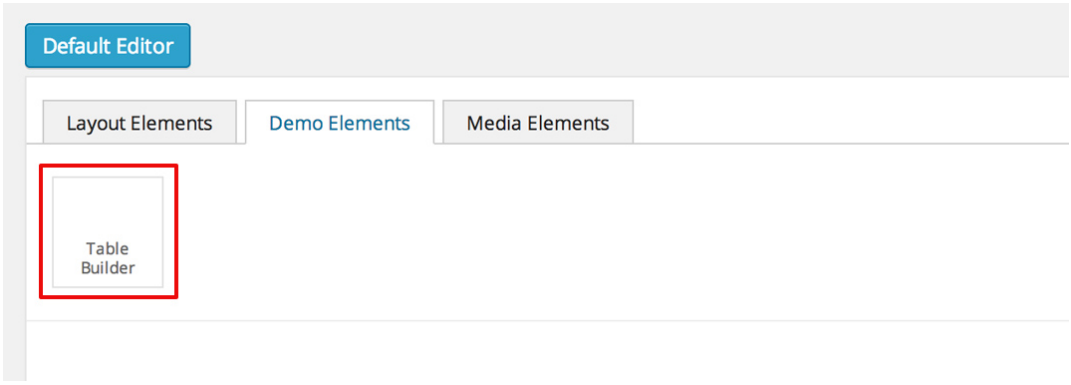
Creating an advanced shortcode with a custom class

This ex will go through creating a `[table_builder]` shortcode, it will make use of it's own custom option type:

1. Create a `table-builder` folder in `framework-customizations/extensions/shortcodes/shortcodes/`.
2. Create *a config file* inside that folder:

```php
<?php if (!defined('FW')) die('Forbidden');

$cfg = array(
    'page_builder' => array(
        'title'        => __('Table Builder', '{domain}'),
        'description' => __('Creates custom tables', '{domain}'),
        'tab'          => __('Demo Elements', '{domain}'),
        'popup_size'   => 'large'
    )
);
```

Note: At this point the shortcode should appear in the **Demo Elements** tab of the layout builder as shown below:



Tip: To add an icon to the shortcode see the *icon section*

---

3. A custom option type is needed for the shortcode to be created, because the ones that exist in the framework do not suit our needs.
   (a) Create an includes folder and a `table-builder` option type inside it.

(b) Create a *custom class* for the shortcode and override the `_init()` method, to load the custom option type class file.

```
<?php if (!defined('FW')) die('Forbidden');

class FW_Shortcode_Table_Builder extends FW_Shortcode
{
    /**
     * @internal
     */
    public function _init()
    {
        if (is_admin()) {
            $this->load_option_type();
        }
    }

    private function load_option_type()
    {
        require $this->locate_path('/includes/fw-option-type-
↪table-builder/class-fw-option-type-table-builder.php');
    }

    // ...

}
```

(c) Create an *options file* with the custom option type:

```
<?php if (!defined('FW')) die('Forbidden');

$options = array(
    'table' => array(
        'type'  => 'table-builder',
        'label' => false,
        'desc'  => false,
    )
);
```

Note: At this point, when clicking the shortcode inside the canvas area of the layout builder a pop-up window containting the options will appear:

4. Create the *view file* and make use of the custom option type's value.

The `[table_builder]` shorcode is completed! The directory structure of the shortcode is as shown below:

```
framework-customizations/
└─theme/
  └─shortcodes/
    └─table-builder/
      ├─class-fw-shortcode-table-builder.php
      ├─config.php
      ├─options.php
      ├─views/
      │ └─view.php
      └─includes/
        └─fw-option-type-table-builder/
          ├─class-fw-option-type-table-builder.php
          ├─static/
          └─views/
```

Enqueue shortcode dynamic css in page head

When the shortcode has options that affect its css, you can populate the `style="..."` attribute in `view.php`:

```
// file:: {theme}/framework-customizations/extensions/shortcodes/shortcodes/{name}/
↪views/view.php

<p style="color: <?php echo esc_attr($atts['color']) ?>;" >Hello, World!</p>
```

A better solution would be to assign shortcode an unique id and enqueue in head css for that id.

1. Add a hidden option that will generate an unique id

```
// file: {theme}/framework-customizations/extensions/shortcodes/
↪shortcodes/{name}/options.php

$options = array(
    'id'    => array( 'type' => 'unique' ),
    'color' => array( 'type' => 'color-picker' ),
```

```
    ...
);
```

2. Output the id in view

```
// file: {theme}/framework-customizations/extensions/shortcodes/
↪shortcodes/{name}/views/view.php

<p id="shortcode-<?php echo esc_attr($atts['id']); ?>" >Hello, World!</p>
```

3. Enqueue the main style

```
// file: {theme}/framework-customizations/extensions/shortcodes/
↪shortcodes/{name}/static.php

wp_enqueue_style(
    'theme-shortcode-{name}',
    fw_ext('shortcodes')->locate_URI('/shortcodes/{name}/static/css/
↪styles.css')
);
```

4. Enqueue the dynamic style

```
// file: {theme}/framework-customizations/extensions/shortcodes/
↪shortcodes/{name}/static.php

...

if (!function_exists('_action_theme_shortcode_{name}_enqueue_dynamic_css
↪')):

/**
 * @internal
 * @param array $data
 */
function _action_theme_shortcode_{name}_enqueue_dynamic_css($data) {
    $shortcode = '{name}';
    $atts = shortcode_parse_atts( $data['atts_string'] );
    $atts = fw_ext_shortcodes_decode_attr($atts, $shortcode, $data['post
↪']->ID);

    wp_add_inline_style(
        'theme-shortcode-'. $shortcode,
        '#shortcode-'. $atts['id'] .' { '.
            'color: '. $atts['color'] .';'.
        ' } '
    );
}
add_action(
    'fw_ext_shortcodes_enqueue_static:{name}',
    '_action_theme_shortcode_{name}_enqueue_dynamic_css'
);

endif;
```

Visual Composer integration

Note: This feature is part of Bumblebee package

Unyson shortcodes are automatically translated to Visual Composer shortcodes by Visual Composer extension which is part of Bumblebee package. Visual Composer extension has to be activated in Unyson admin panel. There are a few things to remember concerning Visual Composer params.

Note: Read more about Visual Composer params at https://wpbakery.atlassian.net/wiki/pages/viewpage.action?pageId=524332

Option specific settings

Option specific settings are set in options.php shortcode file.

- dependency

You can easily define dependency between options as in example below:

```
// independent option, no dependency here
    'icons_number' => array(
            'type'    => 'select',
            'choices' => array(
                    '1' => '1',
                    '2' => '2',
                    '3' => '3',
                    '4' => '4',
            ),
            'value'   => '4',
            'label'   => __( 'Number of icons' ),
            'tab'     => __( 'General', 'ct_theme' ),
    ),

// dependent option
    'column_2_title' => array(
            'label'      => __( 'Title:', 'ct_theme' ),
            'default'    => '',
            'type'       => 'text',
            'tab'        => __( "Column 2", 'ct_theme' ),
            'dependency' => array(
                    'element' => 'icons_number',
                    'value'   => array( '2', '3', '4' ),
            )
    ),
```

Tip: Dependency element name can also be used in connection with 'shortcode' option type as well as in widget context.

Tip: Read more about VC dependency here

- Hide an option in widget

When you mapped a shortcode to a widget using Bee_Widget class and you want selected options not to be visible in widget options, just set 'hide_in_widget' field to true as in the example below:

```
$options[] = array(
        'type'           => 'checkbox',
        'hide_in_widget' => true,
);
```

- Hide an option in VC shortcode

When you want selected options not to be visible in Visual Composer shortcode options editor, just set 'hide_in_vc' field to true as in the example below. This is useful when you have a large shortcode mapped to a widget using Bee_Widget class and you wish some options to be visible only in widgets.

```
$options[] = array(
        'type'       => 'checkbox',
        'hide_in_vc' => true,
);
```

- Weight / Position

There is an easy way to set option position among other options

```
$options[] = array(
        'type'       => 'checkbox',
        'position'   => 0,
);
```

Shortcode specific settings

Shortcode specific settings are set in config.php shortcode file.

- custom_markup, js_view, admin_enqueue_js

In Bumblebee you can use helper functions to call default table view in VC backend editor, see example:

```
$cfg = array(
  'page_builder' => array(
    'title'            => __( 'Team display', 'unyson' ),
    'description'      => __( 'Team display', 'unyson' ),
    'tab'              => __( 'Disrupt', 'fw' ),
    'icon'             => 'fa fa-question-circle-o fa-2x',
    'js_view'          => fw_ext_shortcodes_get_custom_vc_view_name(),
    'custom_markup'    => fw_ext_shortcodes_get_custom_vc_view_markup(),
    'admin_enqueue_js' => fw_ext_shortcodes_get_custom_vc_view_js_uri(),
  )
);
```

Which should result in something like this:

- show_settings_on_create

Set it to false if content element's settings page shouldn't open automatically after adding it to the stage

```
$cfg = array(
  'page_builder' => array(
    'title'                    => __( 'Single Property', 'ct_theme' ),
    'description'              => __( 'Display a property page', 'ct_theme' ),
    'tab'                      => __( 'Estato', 'ct_theme' ),
    'icon'                     => 'fa fa-building-o fa-2x',
    'show_settings_on_create' => false,
  )
);
```

6.4.3 Slider

Adds a sliders module to your website from where you'll be able to create different built in jQuery sliders for your homepage and rest of the pages.

Directory Structure

The slider extension directory has the following structure:

```
slider/
├-...
└-extensions/
  ├-{slider-type}/
  ├-...
  └-{slider-type}/
    ├-class-fw-extension-{slider-type}.php # optional
    ├-config.php
    ├-options/ # optional
    |  ├-options.php
    |  └-...
    ├-static/ # optional
    |  ├-css/
    |  |  ├-auto-enqueued-style.css
    |  |  └-...
    |  ├-img/
    |  |  ├-preview.jpg
    |  |  ├-thumb.jpg
    |  |  └-...
```

```
    │  └─js/
    │    ├─auto-enqueued-script.js
    │    └─...
    └─views/
      ├─{slider-type}.php
      └─...
```

Create a simple slider type

To create simple slider type, create a child extension. In our case the slider type is `bx-slider`, so the child extension directory will be `framework-customizations/extensions/media/extensions/slider/extensions/bx-slider`.

Configuration

The configuration file `config.php` contains the following parameters:

```php
/**
 * Specify available population methods.
 *
 * There are 4 built-in population methods:
 *   'posts'      Populate with posts
 *   'categories' Populate with categories
 *   'tags'       Populate with tags
 *   'custom'     Populate with custom content provided by the user
 */
$cfg['population_methods'] = array('posts', 'categories', 'tags', 'custom');

/**
 * Specify what media types the slider supports.
 *
 * There are 2 media types available:
 *   'image' Supports images
 *   'video' Supports videos
 */
$cfg['multimedia_types'] = array('image');
```

Template

View the file that contains the slider template for frontend, is located in `views/{slider-type}.php`. Here is an example for our `bx-slider`:

```php
<?php if (!defined('FW')) die('Forbidden');
/**
 * @var array $data
 */

$unique_id = 'bx-slider-'. fw_unique_increment();
?>
<?php if (isset($data['slides'])): ?>
    <script type="text/javascript">
        jQuery('document').ready(function () {
            jQuery('#<?php echo $unique_id ?>').bxSlider();
```

```
        });
    </script>
    <ul id="<?php echo $unique_id ?>" class="bxslider">
        <?php foreach ($data['slides'] as $slide): ?>
            <li>
                <?php if ($slide['multimedia_type'] === 'video') : ?>
                    <?php echo fw_oembed_get($slide['src'], $dimensions); ?>
                <?php else: ?>
                    <img src="<?php echo fw_resize($slide['src'], $dimensions['width
↪'], $dimensions['height']); ?>"
                         alt="<?php echo esc_attr($slide['title']) ?>"
                         width="<?php echo esc_attr($dimensions['width']); ?>"
                         height="<?php echo $dimensions['height']; ?>"/>
                <?php endif; ?>
            </li>
        <?php endforeach; ?>
    </ul>
<?php endif; ?>
```

The `$data` variable that is available in view, has the following structure:

```
$data = array(
    'slides' => array(
        array(
            'title' => 'Slide Title',
            'multimedia_type' => 'video|image',
            'src'   => 'Slide src',
            'desc'  => 'Slide Description',
            'extra' => array(
                /**
                 * This array can be empty, it depends on population method
                 * or if user set extra options for population method
                 */
                'extra-slide-key' => 'Extra slide value',
                ...
            )
        ),
        ...
    ),
    'settings' => array(
        'title'             => 'Slider Title',
        'slider_type'       => '{slider-type}',
        'population_method' => 'posts|categories|tags|custom',
        'post_id'           => 10, // ID of the slider (slider is a custom post)
        'extra' => array(
            /**
             * This array can be empty.
             * Or will have something in it
             * if user set custom options for slider in options/options.php
             */
            'extra-slider-key' => 'Extra slider values',
            ...
        )
    )
);
```

Options

Optionally, if your slider have extra options, you can create 2 types of option files within `options/` directory:

- `options.php` - extra options shown after default options on add and edit slider page.
- `{population-method}.php` - extra options for concrete population method, shown after default options on edit slider page.

Static files

Scripts, styles and images are stored in `static/` directory.

- `static/images/` - directory for images. This directory has 2 special images that you should create:
  - `thumb.png` - small image with frontend preview of this slider type. Is displayed on the admin side in Slider Type choices.
  - `preview.png` - a bigger image with frontend preview of this slider type. It is displayed when the user hovers the `thumb.png` in the WordPress admin.
- `static/css/` - directory for styles. They will be automatically enqueued in frontend.
- `static/js/` - directory for scripts. They will be automatically enqueued in frontend.

Note: Styles and scripts are enqueued in alphabetical orders. You cannot set dependencies for them. So if you want for e.g. `c.js` to be enqueued before `b.js`, you must rename it, or prefix it with some number or letter `a-c.js`.

Create advanced slider type

If you want to create an advanced slider with your own extra logic, you must create a class file named `class-fw-extension-{slider-type}.php` within the slider type directory.

In our case the slider type is `bx-slider`, so the class file will be located in `framework-customizations/extensions/media/extensions/slider/extensions/bx-slider/class-fw-extension-bx-slider.php` and will contain:

```
<?php if (!defined('FW')) die('Forbidden');

class FW_Extension_Bx_Slider extends FW_Slider
{
    /**
     * @internal
     */
    public function _init()
    {
    }
}
```

Then you can take a look at the `FW_Slider` methods to learn what are they doing and decide which one you will overwrite.

Frontend render

There are two ways you can display a slider in frontend:

1. **Builder shortcode** - the main slider extension automatically creates a `[slider]` shortcode which is available in builder in the **Media Elements** tab.
2. **Render from code** - the slider extension has a public method that you can use to render a slider on frontend.

```
fw()->extensions->get('slider')->render_slider(10, array(
    'width'  => 300,
    'height' => 200
));
```

6.4.4 Mega Menu

The Mega Menu extension gives the end-user the ability to construct advanced navigation menus.

Important: This extensions is not be visible by default in Unyson Extensions page. To make it appear in that list, you have to:

- Add the extension name in theme manifest

```
$manifest['supported_extensions'] = array(
    'megamenu' => array(),
);
```

- Or set the `WP_DEBUG` constant to `true`

Overview

When it is turned on, it enriches menu with the following:

1. Ability to set an icon for any menu item
2. Ability to group several menu items into columns placed in rows

HTML/CSS

The extension adds the following css classes:

- `.menu-item-has-icon`
- `.menu-item-has-mega-menu`
- `.sub-menu-has-icons`
- `.mega-menu`
- `.mega-menu-row`

- `.mega-menu-col`

The markup will be the following:

```
li.menu-item-has-mega-menu
    div.mega-menu
        ul.mega-menu-row
            li.mega-menu-col
            li.mega-menu-col
            li.mega-menu-col
        ul.mega-menu-row
            li.mega-menu-col
            li.mega-menu-col
            li.mega-menu-col
```

Note: All other standard WordPress classes and HTML remains the same.

Markup Example

```
<ul>
    <li class="menu-item-has-mega-menu menu-item-has-icon">
        <a class="fa fa-exclamation" href="#">Mega Menu 1</a>
        <div class="mega-menu">
            <ul class="sub-menu mega-menu-row">
                <li class="mega-menu-col">
                    <a href="#">Just Links</a>
                    <ul class="sub-menu">
                        <li>
                            <a href="#">Menu Item 1</a>
                        </li>
                        ...
                    </ul>
                </li>
                <li class="mega-menu-col">
                    <a href="#">Links with Icons</a>
                    <ul class="sub-menu sub-menu-has-icons">
                        <li class="menu-item-has-icon">
                            <a class="fa fa-inbox" href="#">Menu Item 1</a>
                            <p>Praesent quis enim euismod, fringilla quam vitae,␣
↪consectetur quam.</p>
                        </li>
                        <li class="menu-item-has-icon">
                            <a class="fa fa-wrench" href="#">Menu Item 2</a>
                        </li>
                        ...
                    </ul>
                </li>
            </ul>
        </div>
    </li>
    <li class="menu-item-has-icon">
        <a class="fa fa-info-circle" href="#">Home</a>
        <ul class="sub-menu sub-menu-has-icons">
            <li class="menu-item-has-icon">
                <a class="fa fa-info-circle" href="#">Page 2</a>
```

```
            </li>
            <li class="menu-item-has-icon">
                <a class="fa fa-info-circle" href="#">Page 3</a>
                <ul class="sub-menu sub-menu-has-icons">
                    <li class="menu-item-has-icon">
                        <a class="fa fa-key" href="#">Page 4</a>
                    </li>
                    <li class="menu-item-has-icon">
                        <a class="fa fa-briefcase" href="#">Page 5</a>
                    </li>
                    <li class="menu-item-has-icon">
                        <a class="fa fa-gavel" href="#">Page 6</a>
                        <ul class="sub-menu sub-menu-has-icons">
                            <li class="menu-item-has-icon">
                                <a class="fa fa-globe" href="#">Page 7</a>
                            </li>
                            <li>
                                <a href="#">Page 8</a>
                            </li>
                        </ul>
                    </li>
                </ul>
            </li>
        </ul>
    </li>
</ul>
```

Changing item/icon markup

By default the icon is added to

```
<a href="..." class="fa fa-...">Menu item</a>
```

If you want to change it to

```
<a href="..."><i class="fa fa-..."></i> Menu item</a>
```

overwrite this view in your theme

```
<?php if (!defined('FW')) die('Forbidden');

// file: {theme}/framework-customizations/extensions/megamenu/views/item-link.php

/**
 * @var WP_Post $item
 * @var string $title
 * @var array $attributes
 * @var object $args
 * @var int $depth
 */

{
    $icon_html = '';

    if (
        fw()->extensions->get('megamenu')->show_icon()
```

```
        &&
        ($icon = fw_ext_mega_menu_get_meta($item, 'icon'))
    ) {
        $icon_html = '<i class="'. $icon .'"></i> ';
    }
}

// Make a menu WordPress way
echo $args->before;
echo fw_html_tag('a', $attributes, $args->link_before . $icon_html . $title . $args->
↪link_after);
echo $args->after;
```

6.4.5 Sidebars

Brings another layer of customization freedom to your website by letting you add more than one sidebar to a page, or different sidebars on different pages.

- *Configuration*
- *Helpers*
- *Filters*

Configuration

```
<?php if (!defined('FW')) die('Forbidden');

// file: framework-customizations/extensions/sidebars/config.php

$cfg = array(
    'sidebar_positions' => array(
        'position-id' => array(
            /**
             * Image from: framework-customizations/extensions/sidebars/images/
             * (required)
             */
            'icon_url' => 'picture.png',
            /**
             * Number of sidebars on page.
             * The maximum number is 4.
             * (optional)
             * (default 0)
             */
            'sidebars_number' => 0
        ),
        // other positions ...
    ),
    /**
     * Array that will be passed to register_sidebar($args)
     * Should be without 'id' and 'name'.
     * Will be used for all dynamic sidebars.
     */
```

```
    'dynamic_sidebar_args' => array(
        'before_widget' => '<div id="%1$s" class="widget %2$s">',
        'after_widget'  => '</div>',
        'before_title'  => '<h3>',
        'after_title'   => '</h3>',
    ),
    /**
     * Render sidebar metabox in post types.
     * By default is set to false.
     * If you want to render sidebar in post types set it to true.
     */
     'show_in_post_types' => false
);
```

Helpers

- `fw_ext_sidebars_show($color)` - display sidebar in frontend. The parameter `$color` is the color of the sidebar selected from the WordPress admin and can be: `blue`, `yellow`, `green` or `red`.
- `fw_ext_sidebars_get_current_position()` - can be called in the frontend to find out current position name. It returns `position-id` from `$cfg['sidebar_positions']`, or null.
- `fw_ext_sidebars_get_current_preset()` - can be called in the frontend to find out the sidebar's settings for current page template.

```
// file: sidebar-content.php

<?php if (!defined('FW')) die('Forbidden');

$current_position = fw_ext_sidebars_current_position_get();

if ($current_position !== 'position-id') {
    echo fw_ext_sidebars_show('green');
}

if ($current_position === 'position-id-2') {
    echo fw_ext_sidebars_show('blue');
}

if ($current_position === 'position-id-3') {
    echo fw_ext_sidebars_show('yellow');
}
```

Filters

- `fw_ext_sidebars_post_types` - use this filter to change/remove post types that are used in extension.

```
/** @internal */
function _filter_remove_post_type_from_sidebars($post_types_list) {
    unset($post_types_list['post_type_name']);

    return $post_types_list;
}
add_filter('fw_ext_sidebars_get_post_types', '_filter_remove_post_type_
↪from_sidebars' );
```

- `fw_ext_sidebars_taxonomies` - use this filter to change/remove taxonomies that are used in extension.

```
/** @internal */
function _filter_remove_taxonomy_from_sidebars($taxonomy_list) {
    unset($taxonomy_list['taxonomy_name']);

    return $taxonomy_list;
}
add_filter('fw_ext_sidebars_get_taxonomies', '_filter_remove_taxonomy_
↪from_sidebars');
```

- `fw_ext_sidebars_conditional_tags` - use this filter to change/remove/add conditional tags that are used in extension.

```
/** @internal */
function _filter_fw_ext_sidebars_add_conditional_tag($conditional_tags) {
    $conditional_tags['is_archive_page_slug'] = array(
        'order_option' => 2, // (optional: default is 1) position in the
↪'Others' lists in backend
        'check_priority' => 'last', // (optional: default is last, can be␣
↪changed to 'first') use it to change priority checking conditional tag
        'name' => __('Portfolio archive', '{domain}'), // conditional tag␣
↪title
        'conditional_tag' => array(
            'callback' => 'is_post_type_archive', // existing callback
            'params' => array('fw-portfolio') //parameters for callback
        )
    );

    return $conditional_tags;
}
add_filter('fw_ext_sidebars_conditional_tags', '_filter_fw_ext_sidebars_
↪add_conditional_tag' );
```

6.4.6 Portfolio

The Portfolio extension allows you to create Portfolio section on your site.

Configuration

In the config.php file, you can set the portfolio Gallery and Featured Image sizes.

```
$cfg['image_sizes'] = array(
    'featured-image' => array(
        'width'  => 227,
        'height' => 142,
        'crop'   => true
```

```
    ),
    'gallery-image'  => array(
        'width'  => 474,
        'height' => 241,
        'crop'   => true
    )
);
```

Also define if the portfolio custom post will support gallery or not.

```
$cfg['has-gallery'] = true;
```

Helpers

- `fw_ext_portfolio_get_gallery_images( $post_id )` - use this function to return all project gallery images.

```
<?php if ( have_posts() ) : ?>
    <?php while ( have_posts() ) : ?>
        <?php $gallery = fw_ext_portfolio_get_gallery_images(); ?>
        <ul class="gallery">
            <?php foreach( $gallery as $image ) : ?>
                <li>
                    <a href="<?php echo get_permalink($image['attachment_
↪id'])?>">
                        <img src="<?php echo $image['url'] ?>" alt=""/>
                    </a>
                </li>
            <?php endforeach ?>
        </ul>
    <?php endwhile ?>
<?php endif ?>
```

> **Note:** If you are in The Loop, the global $post will be used for $post_id

Hooks

- `fw_ext_portfolio_post_slug` - portfolio custom post slug

```
/**
 * @internal
 */
function _filter_custom_portfolio_post_slug($slug) {
    return 'work';
}
add_filter('fw_ext_portfolio_post_slug', '_filter_custom_portfolio_post_
↪slug');
```

- `fw_ext_portfolio_taxonomy_slug` - portfolio taxonomy slug

```
/**
 * @internal
 */
```

```
function _filter_custom_portfolio_tax_slug($slug) {
    return 'works';
}
add_filter('fw_ext_portfolio_taxonomy_slug', '_filter_custom_portfolio_
↪tax_slug');
```

- `fw_ext_projects_post_type_name` - portfolio custom post labels (plural and singular)

```
/**
 * @internal
 */
function _filter_portfolio_labels($labels) {
    $labels = array(
        'singular' => __('Custom Project', '{domain}'),
        'plural'   => __('Custom Projects', '{domain}'),
    );

    return $labels;
}
add_filter('fw_ext_projects_post_type_name', '_filter_portfolio_labels');
```

- `fw_ext_portfolio_category_name` - portfolio taxonomy labels (plural and singular)

```
/**
 * @internal
 */
function portfolio_tax_labels_names( $labels ) {
    $labels = array(
        'singular' => __( 'Custom Category', '{domain}' ),
        'plural'   => __( 'Custom Categories', '{domain}' ),
    );

    return $labels;
}
add_filter( 'fw_ext_portfolio_category_name', 'portfolio_tax_labels_names
↪' );
```

Views

Templates are located in the views/ directory. Here is the list of templates that you can customize:

- `single.php` - Portfolio course single post template. By default is used `single.php` from the theme root directory, you can overwrite it by creating `framework-customizations/extensions/portfolio/views/single.php`.
- `taxonomy.php` - Portfolio category template. By default is used `taxonomy.php` from the theme root directory, you can overwrite it by creating `framework-customizations/extensions/portfolio/views/taxonomy.php`.
- `content.php` - Default portfolio single page template content. It is loaded if the `framework-customizations/extensions/portfolio/views/single.php` doesn't exist and is used `single.php` from the theme root directory. The content of this view is rendered using worpdress the_content filter, when the course single page is loaded.

6.4.7 Backup & Demo Content

This extension lets you create an automated backup schedule, import demo content or even create a demo content archive for migration purposes.

Demo Content Install

The demo content install might be very useful for your clients. After they install the theme, they won't need to configure and create content from scratch but instead they can install the demo content you've created. Their site will look exactly like your theme demo page, and they can just start to modify and adapt the existing content.

Create Demos

In order to create a demo content archive, just create a Content Backup.

Important: If you have contact forms added in pages with the visual page builder, please check the Mailer settings and remove all private credentials you might have inserted there.

The Mailer settings are saved in a wp option which is excluded on content backup but unfortunately the same settings are saved in other places. This problem will be fixed later but until then, you'll need to manually clear the mailer settings for each contact form before content backup.

Tip: Before creating a Content Backup for Demo Install use a plugin to remove post revisions.

The next step is to let your users view and select which demo content to install. This page is created in the WordPress admin under **Tools > Demo Content Install** if the theme has at least one demo content available. The demo content archives can be placed in theme or can be downloaded from a remote server.

Demos bundled in theme

1. Create Content Backup
2. Extract the zip in `{theme}/demo-content/{demo-name}/`
3. Create `{theme}/demo-content/{demo-name}/manifest.php` with the following contents:

```
<?php if (!defined('FW')) die('Forbidden');
/**
 * @var string $uri Demo directory url
 */
```

```
$manifest = array();
$manifest['title'] = __('Awesome Demo', '{domain}');
$manifest['screenshot'] = $uri . '/screenshot.png';
$manifest['preview_link'] = 'https://your-site.com/demo/awesome';
```

4. Go to **Tools > Demo Content Install** menu in the WordPress admin. The demo(s) should be listed on that page.

Demos on remote server

1. Create Content Backup
2. Upload the zip on your server (in any directory you want, for e.g. `your-site.com/demo/`)
3. Upload this download script, let's say in the same directory `your-site.com/demo/`
4. In the same directory with the download script, create a config.php file and add your demos in the following format:

```
//   'demo-id' => '/path/to/demo.zip',
'awesome-demo' => dirname(__FILE__) .'/awesome-demo.zip',
```

5. Register the demo(s) in your theme. Add in {theme}/inc/hooks.php:

```
/**
 * @param FW_Ext_Backups_Demo[] $demos
 * @return FW_Ext_Backups_Demo[]
 */
function _filter_theme_fw_ext_backups_demos($demos) {
    $demos_array = array(
        'your-demo-id' => array(
            'title' => __('Demo Title', '{domain}'),
            'screenshot' => 'https://your-site.com/.../screnshot.png',
            'preview_link' => 'https://your-site.com/demo/your-demo-id',
        ),
        // ...
    );

    $download_url = 'https://your-site.com/path/to/download-script/';

    foreach ($demos_array as $id => $data) {
        $demo = new FW_Ext_Backups_Demo($id, 'piecemeal', array(
            'url' => $download_url,
            'file_id' => $id,
        ));
        $demo->set_title($data['title']);
        $demo->set_screenshot($data['screenshot']);
        $demo->set_preview_link($data['preview_link']);

        $demos[ $demo->get_id() ] = $demo;

        unset($demo);
    }

    return $demos;
}
add_filter('fw:ext:backups-demo:demos', '_filter_theme_fw_ext_backups_
↪demos');
```

6. Go to **Tools > Demo Content Install** menu in the WordPress admin. The demo(s) should be listed on that page.

Hooks

- Filter to exclude wp options on database export

```
function _filter_theme_fw_ext_backups_db_export_exclude_option($exclude,
↪$option_name, $is_full_backup) {
    if (!$is_full_backup) {
        if ($option_name === 'your-private-option') {
            return true;
        }
    }

    return $exclude;
}
add_filter(
    'fw_ext_backups_db_export_exclude_option',
    '_filter_theme_fw_ext_backups_db_export_exclude_option',
    10, 3
);
```

- Filter to exclude wp options on database restore

> **Note:** The current options (if exist) will be wiped out. To keep the current options, use *the following filter*.

```
function _filter_theme_fw_ext_backups_db_restore_exclude_option($exclude,
↪$option_name, $is_full) {
    if (!$is_full) {
        if ($option_name === 'your-special-option') {
            return true;
        }
    }

    return $exclude;
}
add_filter(
    'fw_ext_backups_db_restore_exclude_option',
    '_filter_theme_fw_ext_backups_db_restore_exclude_option',
    10, 3
);
```

- Filter to preserve current wp options values on database restore

```
function _filter_fw_ext_backups_db_restore_keep_options($options, $is_
↪full) {
    if (!$is_full) {
        $options[ 'your-special-option' ] = true;
    }

    return $options;
}
add_filter(
    'fw_ext_backups_db_restore_keep_options',
    '_filter_fw_ext_backups_db_restore_keep_options',
```

```
    10, 2
);
```

- Filter to register a custom directory that contains theme demos (for e.g. a plugin bundled with theme)

```
function _filter_theme_fw_ext_backups_demo_dirs($dirs) {
    $dirs['/path/to/dir-with-theme-demos']
    = 'http://.../uri/to/dir-with-theme-demos';

    return $dirs;
}
add_filter('fw_ext_backups_demo_dirs', '_filter_theme_fw_ext_backups_demo_
↪dirs');
```

6.4.8 Forms

This extension adds the possibility to create a forms (for e.g. a contact form). Use the drag & drop form builder to create any form you'll ever want or need.

Customize Views

- **Form Fields** - Frontend form fields views can be customized in `framework-customizations/extensions/forms/form-builder/items/{item-type}/views/view.php`. All built-in form builder item types can be found in `framework/extensions/forms/includes/option-types/form-builder/items/` directory.

  For e.g. to overwrite the view for item type `text` (which is located in `framework/extensions/forms/includes/option-types/form-builder/items/text`) create `framework-customizations/extensions/forms/form-builder/items/text/views/view.php`.

- **Form Fields Container** - The view for the container that wraps the form fields can be customized in `framework-customizations/extensions/forms/form-builder/views/items.php`.

Contact Form Views

- **Form Content** - The inner contents of the `<form>` can be customized in `framework-customizations/extensions/forms/extensions/contact-forms/views/form.php`.
- **Email Content** - The contents of the email that is sent after an user submitted the contact form can be customized in `framework-customizations/extensions/forms/extensions/contact-forms/views/email.php`.

Create Form Builder Item Type

First, make sure you understand how the base builder works.

The Forms extension have a built-in `form-builder` option type (that can be found in the `framework-customizations/extensions/forms/form-builder/` directory) which is used by the Contact Forms sub-extension. To create an item type for `form-builder` you have to look in its method called `item_type_is_valid()` to see what class you must extend in order to be accepted by the builder.

```
class FW_Option_Type_Form_Builder
{
    ...

    /**
     * @param FW_Option_Type_Builder_Item $item_type_instance
     * @return bool
     */
    protected function item_type_is_valid($item_type_instance)
    {
        return is_subclass_of($item_type_instance, 'FW_Option_Type_Form_Builder_Item
↪');
    }
}
```

So you have to extend the `FW_Option_Type_Form_Builder_Item` class and register it as a builder item type.

Below is explained how to create a simple `Yes/No` question radio input.

Create the `framework-customizations/extensions/forms/includes/builder-items/yes-no` directory.

Create `framework-customizations/extensions/forms/includes/builder-items/yes-no/class-fw-option-type-form-builder-item-yes-no.php` with the following contents:

```
class FW_Option_Type_Form_Builder_Item_Yes_No extends FW_Option_Type_Form_Builder_Item
{
    /**
     * The item type
     * @return string
     */
    public function get_type()
    {
        return 'yes-no';
    }

    /**
     * The boxes that appear on top of the builder and can be dragged down or clicked␣
↪to create items
     * @return array
     */
    public function get_thumbnails()
    {
        return array(
            array(
                'html' =>
                    '<div class="item-type-icon-title">'.
                    '    <div class="item-type-icon"><span class="dashicons dashicons-
↪editor-help"></span></div>'.
                    '    <div class="item-type-title">'. __('Yes/No Question', 'unyson
↪') .'</div>'.
```

```
                    '</div>',
                )
            );
        }

        /**
         * Enqueue item type scripts and styles (in backend)
         */
        public function enqueue_static()
        {
            $uri = fw_get_template_customizations_directory_uri('/extensions/forms/
↪includes/builder-items/yes-no/static');

            wp_enqueue_style(
                'fw-form-builder-item-type-yes-no',
                $uri .'/backend.css',
                array(),
                fw()->theme->manifest->get_version()
            );

            wp_enqueue_script(
                'fw-form-builder-item-type-yes-no',
                $uri .'/backend.js',
                array('fw-events'),
                fw()->theme->manifest->get_version(),
                true
            );

            wp_localize_script(
                'fw-form-builder-item-type-yes-no',
                'fw_form_builder_item_type_yes_no',
                array(
                    'l10n' => array(
                        'item_title'        => __('Yes/No', 'unyson'),
                        'label'             => __('Label', 'unyson'),
                        'toggle_required'   => __('Toggle mandatory field', 'unyson'),
                        'edit'              => __('Edit', 'unyson'),
                        'delete'            => __('Delete', 'unyson'),
                        'edit_label'        => __('Edit Label', 'unyson'),
                        'yes'               => __('Yes', 'unyson'),
                        'no'                => __('No', 'unyson'),
                    ),
                    'options'  => $this->get_options(),
                    'defaults' => array(
                        'type'    => $this->get_type(),
                        'options' => fw_get_options_values_from_input($this->get_
↪options(), array())
                    )
                )
            );

            fw()->backend->enqueue_options_static($this->get_options());
        }

        /**
         * Render item html for frontend form
         *
         * @param array $item Attributes from Backbone JSON
```

```
     * @param null|string|array $input_value Value submitted by the user
     * @return string HTML
     */
    public function frontend_render(array $item, $input_value)
    {
        return '<pre>'. print_r($item, true) .'</pre>';
    }

    /**
     * Validate item on frontend form submit
     *
     * @param array $item Attributes from Backbone JSON
     * @param null|string|array $input_value Value submitted by the user
     * @return null|string Error message
     */
    public function frontend_validate(array $item, $input_value)
    {
        return 'Test error message';
    }

    private function get_options()
    {
        return array(
            array(
                'g1' => array(
                    'type' => 'group',
                    'options' => array(
                        array(
                            'label' => array(
                                'type'  => 'text',
                                'label' => __('Label', 'unyson'),
                                'desc'  => __('The label of the field that will be␣
↪displayed to the users', 'unyson'),
                                'value' => __('Yes/No', 'unyson'),
                            )
                        ),
                        array(
                            'required' => array(
                                'type'  => 'switch',
                                'label' => __('Mandatory Field?', 'unyson'),
                                'desc'  => __('If this field is mandatory for the user
↪', 'unyson'),
                                'value' => true,
                            )
                        ),
                    )
                )
            ),
            array(
                'g2' => array(
                    'type' => 'group',
                    'options' => array(
                        array(
                            'default_value' => array(
                                'type'  => 'radio',
                                'label' => __('Default Value', 'unyson'),
                                'choices' => array(
                                    '' => __('None', 'unyson'),
```

```
                                    'yes' => __('Yes', 'unyson'),
                                    'no' => __('No', 'unyson'),
                                ),
                            )
                        )
                    )
                )
            ),
        );
    }
}
FW_Option_Type_Builder::register_item_type('FW_Option_Type_Form_Builder_Item_Yes_No');
```

Create `framework-customizations/extensions/forms/includes/builder-items/yes-no/static/backend.js`:

```
fwEvents.one('fw-builder:'+ 'form-builder' +':register-items', function(builder){
    var currentItemType = 'yes-no';
    var localized = window['fw_form_builder_item_type_yes_no'];

    var ItemView = builder.classes.ItemView.extend({
        template: _.template(
            '<div class="fw-form-builder-item-style-default fw-form-builder-item-type-
↪'+ currentItemType +'">'+
                '<div class="fw-form-item-controls fw-row">'+
                    '<div class="fw-form-item-controls-left fw-col-xs-8">'+
                        '<div class="fw-form-item-width"></div>'+
                    '</div>'+
                    '<div class="fw-form-item-controls-right fw-col-xs-4 fw-text-right
↪">'+
                        '<div class="fw-form-item-control-buttons">'+
                            '<a class="fw-form-item-control-required dashicons<% if␣
↪(required) { %> required<% } %>" data-hover-tip="<%- toggle_required %>" href="#"␣
↪onclick="return false;" >*</a>'+
                            '<a class="fw-form-item-control-edit dashicons dashicons-
↪edit" data-hover-tip="<%- edit %>" href="#" onclick="return false;" ></a>'+
                            '<a class="fw-form-item-control-remove dashicons␣
↪dashicons-no-alt" data-hover-tip="<%- remove %>" href="#" onclick="return false;" >
↪</a>'+
                        '</div>'+
                    '</div>'+
                '</div>'+
                '<div class="fw-form-item-preview">'+
                    '<div class="fw-form-item-preview-label">'+
                        '<div class="fw-form-item-preview-label-wrapper"><label data-
↪hover-tip="<%- edit_label %>"><%- label %></label> <span <% if (required) { %>class=
↪"required"<% } %>>*</span></div>'+
                        '<div class="fw-form-item-preview-label-edit"><!-- --></div>'+
                    '</div>'+
                    '<div class="fw-form-item-preview-input">'+
                        '<label><input type="radio" disabled <% if (default_value ===␣
↪\'yes\') { %>checked<% } %>> <%- yes %></label><br/>'+
                        '<label><input type="radio" disabled <% if (default_value ===␣
↪\'no\') { %>checked<% } %>> <%- no %></label>'+
                    '</div>'+
                '</div>'+
            '</div>'
        ),
```

```
        events: {
            'click': 'onWrapperClick',
            'click .fw-form-item-control-edit': 'openEdit',
            'click .fw-form-item-control-remove': 'removeItem',
            'click .fw-form-item-control-required': 'toggleRequired',
            'click .fw-form-item-preview .fw-form-item-preview-label label':
↪'openLabelEditor',
            'change .fw-form-item-preview-input input':
↪'updateDefaultValueFromPreviewInput'
        },
        initialize: function() {
            this.defaultInitialize();

            // prepare edit options modal
            {
                this.modal = new fw.OptionsModal({
                    title: localized.l10n.item_title,
                    options: this.model.modalOptions,
                    values: this.model.get('options'),
                    size: 'small'
                });

                this.listenTo(this.modal, 'change:values', function(modal, values) {
                    this.model.set('options', values);
                });

                this.model.on('change:options', function() {
                    this.modal.set(
                        'values',
                        this.model.get('options')
                    );
                }, this);
            }

            this.widthChangerView = new FwBuilderComponents.ItemView.WidthChanger({
                model: this.model,
                view: this
            });

            this.labelInlineEditor = new FwBuilderComponents.ItemView.
↪InlineTextEditor({
                model: this.model,
                editAttribute: 'options/label'
            });
        },
        render: function () {
            this.defaultRender({
                label: fw.opg('label', this.model.get('options')),
                required: fw.opg('required', this.model.get('options')),
                default_value: fw.opg('default_value', this.model.get('options')),
                toggle_required: localized.l10n.toggle_required,
                edit: localized.l10n.edit,
                remove: localized.l10n.delete,
                edit_label: localized.l10n.edit_label,
                yes: localized.l10n.yes,
                no: localized.l10n.no
            });
```

```
            if (this.widthChangerView) {
                this.$('.fw-form-item-width').append(
                    this.widthChangerView.$el
                );
                this.widthChangerView.delegateEvents();
            }

            if (this.labelInlineEditor) {
                this.$('.fw-form-item-preview-label-edit').append(
                    this.labelInlineEditor.$el
                );
                this.labelInlineEditor.delegateEvents();
            }
        },
        openEdit: function() {
            this.modal.open();
        },
        removeItem: function() {
            this.remove();

            this.model.collection.remove(this.model);
        },
        toggleRequired: function() {
            var values = _.clone(
                // clone to not modify by reference, else model.set() will not␣
↪trigger the 'change' event
                this.model.get('options')
            );

            values.required = !values.required;

            this.model.set('options', values);
        },
        openLabelEditor: function() {
            this.$('.fw-form-item-preview-label-wrapper').hide();

            this.labelInlineEditor.show();

            this.listenToOnce(this.labelInlineEditor, 'hide', function() {
                this.$('.fw-form-item-preview-label-wrapper').show();
            });
        },
        updateDefaultValueFromPreviewInput: function() {
            var values = _.clone(
                // clone to not modify by reference, else model.set() will not␣
↪trigger the 'change' event
                this.model.get('options')
            );

            values.default_value = this.$('.fw-form-item-preview-input input').val();

            this.model.set('options', values);
        },
        onWrapperClick: function(e) {
            if (!this.$el.parent().length) {
                // The element doesn't exist in DOM. This listener was executed after␣
↪the item was deleted
                return;
```

```
            }

            if (!fw.elementEventHasListenerInContainer(jQuery(e.srcElement), 'click',␣
↪this.$el)) {
                this.openEdit();
            }
        }
    });

    var Item = builder.classes.Item.extend({
        defaults: function() {
            var defaults = _.clone(localized.defaults);

            defaults.shortcode = fwFormBuilder.uniqueShortcode(defaults.type +'_');

            return defaults;
        },
        initialize: function() {
            this.defaultInitialize();

            this.modalOptions = localized.options;

            this.view = new ItemView({
                id: 'fw-builder-item-'+ this.cid,
                model: this
            });
        }
    });

    builder.registerItemClass(Item);
});
```

Create `framework-customizations/extensions/forms/includes/builder-items/yes-no/static/backend.css`:

```
/* controls */

.fw-option-type-form-builder .fw-form-builder-item-type-yes-no .fw-form-item-controls␣
↪.fw-form-item-control-buttons {
    display: none;
}

.fw-option-type-form-builder .fw-form-builder-item-type-yes-no:hover .fw-form-item-
↪controls .fw-form-item-control-buttons {
    display: inline-block;
}

.fw-option-type-form-builder .fw-form-builder-item-type-yes-no .fw-form-item-controls␣
↪.fw-form-item-control-buttons > a,
.fw-option-type-form-builder .fw-form-builder-item-type-yes-no .fw-form-item-controls␣
↪.fw-form-item-control-buttons > a:hover {
    text-decoration: none;
}

.fw-option-type-form-builder .fw-form-builder-item-type-yes-no .fw-form-item-controls␣
↪.fw-form-item-control-buttons a.fw-form-item-control-required {
    color: #999999;
}
```

```
.fw-option-type-form-builder .fw-form-builder-item-type-yes-no .fw-form-item-controls␣
↪.fw-form-item-control-buttons a.fw-form-item-control-required.required {
    color: #ff0000;
}

/* end: controls */


/* preview */

.fw-option-type-form-builder .fw-form-builder-item-type-yes-no .fw-form-item-preview {
    padding: 5px 0;
}

.fw-option-type-form-builder .fw-form-builder-item-type-yes-no .fw-form-item-preview .
↪fw-form-item-preview-label {
    padding: 5px 0 10px;
}

.fw-option-type-form-builder .fw-form-builder-item-type-yes-no .fw-form-item-preview .
↪fw-form-item-preview-label span {
    display: none;
}

.fw-option-type-form-builder .fw-form-builder-item-type-yes-no .fw-form-item-preview .
↪fw-form-item-preview-label span.required {
    display: inline;
    color: #ff0000;
}

/* end: preview */
```

Include the item type by creating the `framework-customizations/extensions/forms/hooks.php` file with the following contents:

```
<?php if (!defined('FW')) die('Forbidden');

/** @internal */
function _action_theme_fw_ext_forms_include_custom_builder_items() {
    require_once dirname(__FILE__) .'/includes/builder-items/yes-no/class-fw-option-
↪type-form-builder-item-yes-no.php';
}
add_action('fw_option_type_form_builder_init', '_action_theme_fw_ext_forms_include_
↪custom_builder_items');
```

At this point the item is working only in backend. If you save the form, add it in a page (or post) using Page Builder and open that page in frontend, you will see the item attributes array.

To make item working in frontend, follow the instructions below:

- Change the `frontend_render()` method:

```
class FW_Option_Type_Form_Builder_Item_Yes_No extends FW_Option_Type_Form_Builder_Item
{

    ...
```

```
    public function frontend_render(array $item, $input_value)
    {
        if (is_null($input_value)) {
            $input_value = $item['options']['default_value'];
        }

        return fw_render_view(
            $this->locate_path(
                // Search view in 'framework-customizations/extensions/forms/form-
↪builder/items/yes-no/views/view.php'
                '/views/view.php',
                // Use this view by default
                dirname(__FILE__) .'/view.php'
            ),
            array(
                'item' => $item,
                'input_value' => $input_value
            )
        );
    }
}
```

- Create the view `framework-customizations/extensions/forms/includes/builder-items/yes-no/view.php`:

```
<?php if (!defined('FW')) die('Forbidden');
/**
 * @var array $item
 * @var array $input_value
 */

$options = $item['options'];
?>
<div class="<?php echo esc_attr(fw_ext_builder_get_item_width('form-builder', $item[
↪'width'] .'/frontend_class')) ?>">
    <div class="field-radio input-styled">
        <label><?php echo fw_htmlspecialchars($item['options']['label']) ?>
            <?php if ($options['required']): ?><sup>*</sup><?php endif; ?>
        </label>
        <div class="custom-radio">
            <div class="options">
                <?php
                foreach (array('yes' => __('Yes', 'unyson'), 'no' => __('No', 'unyson
↪')) as $value => $label): ?>
                    <?php
                    $choice_attr = array(
                        'value' => $value,
                        'type' => 'radio',
                        'name' => $item['shortcode'],
                        'id' => 'rand-'. fw_unique_increment(),
                    );

                    if ($input_value === $value) {
                        $choice_attr['checked'] = 'checked';
                    }
                    ?>
                    <input <?php echo fw_attr_to_html($choice_attr) ?> />
                    <label for="<?php echo esc_attr($choice_attr['id']) ?>"><?php␣
↪echo $label ?></label>
```

```
                <?php endforeach; ?>
            </div>
        </div>
    </div>
</div>
```

- Change the `frontend_validate()` method:

```
class FW_Option_Type_Form_Builder_Item_Yes_No extends FW_Option_Type_Form_Builder_Item
{

    ...

    public function frontend_validate(array $item, $input_value)
    {
        $options = $item['options'];

        $messages = array(
            'required' => str_replace(
                array('{label}'),
                array($options['label']),
                __('This {label} field is required', 'unyson')
            ),
            'not_existing_choice' => str_replace(
                array('{label}'),
                array($options['label']),
                __('{label}: Submitted data contains not existing choice', 'unyson')
            ),
        );

        if ($options['required'] && empty($input_value)) {
            return $messages['required'];
        }

        // check if has not existing choices
        if (!empty($input_value) && !in_array($input_value, array('yes', 'no'))) {
            return $messages['not_existing_choice'];
        }
    }
}
```

Now the field will be displayed in frontend as a radio box and the validation will work. The submitted value will be used by the form type you chose when created the form, for e.g. the Contact Forms sub-extensions will send the value in email.

You can inspect the built-in form items to learn what possibilities for customization are available (for e.g. what methods from the extended class you can overwrite).

6.4.9 Breadcrumbs

Creates a simplified navigation menu for the pages that can be placed anywhere in the theme. This will make navigating around the website much easier.

- *Helpers*

Helpers

- `fw_ext_get_breadcrumbs($separator = '>')` - use this function to return breadcrumbs HTML.

```
<h3>My page</h3>
<?php echo fw_ext_get_breadcrumbs( '>' ) ?>
<!-- Home >> Books >> PHP For Beginners -->
```

Note: This function should be used only in the front-end area after WordPress `wp` action.

- `fw_ext_breadcrumbs($separator = '>')` - use this function to render breadcrumbs in your template.

```
<h3>My page</h3>
<?php fw_ext_breadcrumbs( '>' ) ?>
<!-- Home >> Books >> PHP For Beginners -->
```

Note: This function should be used only in the front-end area after WordPress `wp` action.

View

- `breadcrumbs.php` is the template where you can define how the breadcrumbs will be shown on the page. You can overwrite the default view with your own, by creating a `breadcrumbs.php` file in the extension's `views` directory in the child theme.

Filters

- `fw_ext_breadcrumbs_build` - in some cases you want to modify the breadcrumbs items that will be rendered, or a specific item. This filter allows you to modify the breadcrumbs items array before it will be rendered.

```
/**
 * @internal
 */
function _filter_my_custom_breadcrumbs_items( $items ) {
    // do some changes ...

    return $items;
}
add_filter( 'fw_ext_breadcrumbs_build', '_filter_my_custom_breadcrumbs_
↪items' );
```

- `fw_ext_breadcrumbs_search_query` - this filter is used in the search archive template and it contains the search query word. In case you want to modify the word or customize it, like capitalizing it, use this filter.

```
/**
 * @internal
 */
function _filter_my_custom_breadcrumbs_search_word( $word ) {
    return strtoupper( $word );
}
add_filter( 'fw_ext_breadcrumbs_search_query', '_filter_my_custom_
↪breadcrumbs_search_word' );
```

Note: This filter doesn't affect the search query

Date format filters

- `fw_ext_breadcrumbs_date_day_format` - date format for day archives (`d F Y`)
- `fw_ext_breadcrumbs_date_month_format` - date format for day archives (`F Y`)
- `fw_ext_breadcrumbs_date_year_format` - date format for day archives (`Y`)

These 3 filters are used to modify the date format in date archives

```
/**
 * @internal
 */
function _filter_my_custom_breadcrumbs_archive_date_format( $date_format ) {
    return 'd, F Y';
}
add_filter( 'fw_ext_breadcrumbs_date_day_format', '_filter_my_custom_breadcrumbs_
↪archive_date_format' );
```

6.4.10 SEO

This extension will enable you to have a fully optimized WordPress website by adding optimized meta titles, keywords and descriptions. It doesn't have any functionality that is reflected visually in the front end. It offers additional functionality for its sub-extensions, like *Tags* module.

Option placeholders

In order to keep all sub-extensions options together, the SEO extension creates special options sections in:

- **Post Options** - a section *(box or tab)* named **SEO**. In case the post options has the General box with id `general`, the seo section will appear as a sub tab for that box, in other cases it creates a new box.
- **Term Options** - a special section in Term Options.

Options Filters

All the filters have the same functionality, the only differences is where they add options.

- `fw_ext_seo_settings_options` - use to add your own tab in Settings Options **SEO** tab.
- `fw_ext_seo_general_settings` - use to add your own box in Settings Options **SEO > General** tab.
- `fw_ext_seo_general_setting_options` - use to add your own options in Settings Options **SEO > General > General Settings** box.
- `fw_ext_seo_post_type_options` - add options in post options **SEO** box.
- `fw_ext_seo_taxonomy_options` - add options in term options **SEO** section.

  All filters have the same parameter `$options` array.

```
/**
 * @internal
 */
function _filter_set_my_framework_titles_metas_tab( $options ) {
    $options['my_id_tab'] = array(
        'title'   => __( 'My Options', '{domain}' ),
        'type'    => 'tab',
        'options' => array(
            'my_id_title' => array(
                'label' => __( 'Title', '{domain}' ),
                'desc'  => __( 'Set title', '{domain}' ),
                'type'  => 'text',
                'value' => ''
            ),
            'my_id_description' => array(
                'label' => __( 'Description', '{domain}' ),
                'desc'  => __( 'Set description', '{domain}' ),
                'type'  => 'textarea',
                'value' => ''
            ),
        )
    );

    return $options;
}
add_filter( 'fw_ext_seo_settings_options', '_filter_set_my_framework_
↪titles_metas_tab' );
```

Tags

The SEO extension has a list of built in SEO tags, but in some cases you'll want to add your own. To add a new SEO tag you have to use the `fw_ext_seo_init_tags` filter. This is the format for a SEO tag:

```
'tag_name' => array(
    'desc'  => __( 'My new tag', '{domain}' ),
```

```
    'value' => '',
)
```

`tag_name` must be unique. This tag will be available as `%%tag_name%%`.

Add new tag

```
/**
 * @internal
 */
function _filter_add_my_seo_tag($tags) {
    $tags['mytag'] = array(
        'desc'  => __( 'My new tag', '{domain}' ),
        'value' => '',
    );

    return $tags;
}
add_filter( 'fw_ext_seo_init_tags', '_filter_add_my_seo_tag' );
```

The seo tags are created when the extension is initialized, in some cases you cannot know the value of the tag in the current state, like `%%title%%` tag. So in `fw_ext_seo_init_tags` filter, you can add the tag without value, and define the value after the current page location is defined, by using the `fw_ext_seo_update_tags` filter.

Update tag value

```
/**
 * @internal
 */
function _filter_update_my_seo_tag( $tags ) {
    if ( isset($tags['mytag']) && is_front_page() ) {
        $tags['mytag']['value'] = __('Home', '{domain}');
    }

    return $tags;
}
add_filter( 'fw_ext_seo_update_tags', '_filter_update_my_seo_tag' );
```

Actions

- `fw_ext_seo_init_location` - is, initialized with WordPress `wp` action and defines the current page location, used to update SEO tags. Sends as first parameter `$location` an array with details about current page location.

Helpers

- `fw_ext_seo_parse_meta_tags($text)` - parses a string and replaces all SEO tags with their values.

> **Note:** Use this function after the `fw_ext_seo_init_location` action.

Titles and Meta

A sub-extension of the SEO extension, used to setup the theme SEO title and meta keywords for search engines.

Configuration

```
/**
 * Posts types that you want to exclude from titles and meta settings
 */
$cfg['excluded_post_types'] = array('attachment');

/**
 * Taxonomies that you want to exclude from titles and meta settings.
 */
$cfg['excluded_taxonomies'] = array('post_tag');
```

Views

- `meta.php` - Template to render the meta keywords and description.

Hooks

- `fw_ext_seo_titles_metas_load_metas` - Filter to modify some meta properties before it will be rendered in front-end.

```
/**
 * @internal
 * @param array $data All meta that needs to be rendered on the current page
 * @param array $location Current page location details
 */
function _filter_modify_seo_meta($data, $location) {
    /**
     * The view to display current meta.
     * If the view key is not set, then will be loaded meta.php.
     */
    $data['view'] = 'my-view';

    return $data;
}
add_filter('fw_ext_seo_titles_metas_load_metas', '_filter_modify_seo_meta');
```

- `fw_ext_seo_titles_metas_load_title` - Filter to make some modifications in page title before it will be rendered.

```
/**
 * @internal
 * @param string $title The current title
 * @param string $separator Separator symbol
 * @param string $sepdirection Separator position
 * @param array $location Current page location details
 */
function _filter_modify_seo_title($title, $separator, $sepdirection, $location) {
    // ...

    return $title;
}
add_filter('fw_ext_seo_titles_metas_load_title', '_filter_modify_seo_title');
```

Sitemap

Generates the `sitemap.xml` file for search engines.

- *Configuration*
- *Views*
- *Hooks*

Configuration

```
/**
 * Search engines where to report about the sitemap existence.
 * By default the extension supports only Google and Bing.
 */
$cfg['search_engines'] = array('google', 'bing');

/**
 * The frequency of the sitemap refresh (measured in days).
 */
$cfg['sitemap_refresh_rate'] = 2;

/**
 * Exclude post types from sitemap indexing.
 */
$cfg['excluded_post_types'] = array('attachment');

/**
 * Exclude taxonomies from sitemap indexing.
 */
$cfg['excluded_taxonomies']  = array('post_tag');

/**
 * Setup the URL frequency and priority for each post_type, taxonomy and the homepage
 */
$cfg['url_settings'] = array(
    'home' => array(
        'priority'  => 1,
```

```
        'frequency' => 'daily',
    ),
    'posts' => array(
        'priority' => 0.6,
        'frequency' => 'daily',
        /**
         * In case you have specific posts type that you want to set different␣
↪settings
         */
        'type' => array(
            'page' => array(
                'priority'  => 0.5,
                'frequency' => 'weekly',
            )
        )
    ),
    'taxonomies' => array(
        'priority'  => 0.4,
        'frequency' => 'weekly',
        /**
         * In case you have specific taxonomy that you want to set different settings
         */
        'type' => array(
            'post_tag'  => array(
                'priority'  => 0.3,
                'frequency' => 'weekly',
            )
        )
    )
);
```

Views

There are 3 views you can customize:

- `sitemap-header.php` - Header content for the `sitemap.xml` file.
- `sitemap.php` - Content for the `sitemap.xml` file. You can edit this file in case you want to exclude some items from sitemap.
- `sitemap-style.php` - Gives sitemap a user friendly view when it's accessed in the browser.

Hooks

- `fw_ext_seo_sitemap_date_format` - Filter to change the date format of the last modified date in sitemap.

```
/** @internal */
function _filter_modify_sitemap_date_format( $format ) {
    return 'Y M, d';
}
add_filter('fw_ext_seo_sitemap_date_format', '_filter_modify_sitemap_date_format');
```

- `fw_ext_seo_sitemap_pre_update` - Action fired when the sitemap prepares to be updated.
- `fw_ext_seo_sitemap_updated` - Action fired after the sitemap was updated.

- `fw_ext_seo_sitemap_pre_delete` - Action fired when the sitemap prepares to be deleted.
- `fw_ext_seo_sitemap_deleted` - Action fired after the sitemap was deleted.

6.4.11 Events

This extension adds a fully fledged Events module to your theme. It comes with built in pages that contain a calendar where events can be added.

Hooks

- `fw_theme_ext_events_after_content` - adding some html after the content

```
/** @internal */
function _action_theme_render_html($post) {
    if (!empty($post) and $post === fw()->extensions->get( 'events' )->
↪get_post_type_name() ) {
        echo '<div>'. __('Hello world', '{domain}') .'</div>';
    }
}
add_action('fw_theme_ext_events_after_content', '_action_theme_render_html
↪');
```

- `fw_ext_events_post_slug` - event custom post slug

```
/** @internal */
function _filter_custom_events_post_slug($slug) {
    return 'event';
}
add_filter('fw_ext_events_post_slug', '_filter_custom_events_post_slug');
```

- `fw_ext_events_taxonomy_slug` - event taxonomy slug

```
/** @internal */
function _filter_custom_events_taxonomy_slug($slug) {
    return 'events';
}
add_filter('fw_ext_events_taxonomy_slug', '_filter_custom_events_taxonomy_
↪slug');
```

- `fw_ext_events_post_type_name` - event custom post labels (plural and singular)

```
/** @internal */
function _filter_event_labels($labels) {
    $labels = array(
        'singular' => __('Custom Event', '{domain}'),
```

```
        'plural'    => __('Custom Events', '{domain}'),
    );

    return $labels;
}
add_filter('fw_ext_events_post_type_name', '_filter_event_labels');
```

- `fw_ext_events_category_name` - event taxonomy labels (plural and singular)

```
/** @internal */
function _filter_event_tax_labels_names($labels) {
    $labels = array(
        'singular' => __( 'Custom Category', '{domain}' ),
        'plural'   => __( 'Custom Categories', '{domain}' ),
    );

    return $labels;
}
add_filter( 'fw_ext_events_category_name', '_filter_event_tax_labels_names
↪' );
```

New options on the Event edit page

A sub-extension which implements `FW_Events_Interface_Tabs` will include options announced in `fw_get_tabs_options()` method.

```
<?php if (!defined('FW')) die('Forbidden');

class FW_Extension_Event_Tickets extends FW_Extension implements FW_Events_Interface_
↪Tabs {

    public function fw_get_tabs_options() {
        return array(
            'events_tab' => array(
                'title'   => __( 'New Demo Tab Options', '{domain}' ),
                'type'    => 'tab',
                'options' => array(
                    'demo_text_id' => array(
                        'type'  => 'text',
                        'desc'  => 'Demo text description',
                        'label' => 'Demo Text Label',
                    )
                )
            )
        );
    }
}
```

Views

Templates are located in the `views/` directory. Here is the list of templates that you can customize:

- `single.php` - Events single post template. By default is used `single.php` from the theme root directory, you can overwrite it by creating `framework-customizations/extensions/events/views/single.php`.

- `taxonomy.php` - Events category template. By default is used `taxonomy.php` from the theme root directory, you can overwrite it by creating `framework-customizations/extensions/events/views/taxonomy.php`.
- `content.php` - Default events single page template content. It is loaded if the `framework-customizations/extensions/events/views/single.php` doesn't exist and is used `single.php` from the theme root directory. The content of this view is rendered using worpdress the_content filter, when the event single page is loaded.

Events Tags

A way to process events search tags.

Frontend render

There are some ways you can display an event in frontend:

The `events-tags` extension automatically connects to the [calendar] and [map] shortcodes, which is available in **Drag & Drop page builder** in the **Content Elements** tab.

Also it can be rendered from code - the shortcode `[map]` has public method `'render_custom'` that you can use to render a map on frontend.

```
$shortcode_map = fw()->extensions->get('shortcodes')->get_shortcode('map');

if (!empty($shortcode_map)) {
    echo $shortcode_map->render_custom(
        array(
            array(
                'title' => __('Some Title', '{domain}'),
                'url' => 'https://example.com',
                'description' => __('Some description', '{domain}'),
                'thumb' => array('attachment_id' => get_post_thumbnail_id( $post->ID␣
↪) ),
                'location' => array(
                    'coordinates' => array(
                        'lat' => '-34',
                        'lng' => '150'
                    )
                )
            )
        )
    );
}
```

6.4.12 Social

This extension groups in one place all the settings that need to work with social networks.

Filters

To be able to insert the settings on this page were created following filters:

- `fw_ext_social_tabs` - Offers the possibility to add tabs.
- `fw_ext_social_boxes_from_general_tab` - Allows adding boxes with options in general tab.
- `fw_ext_social_main_box_from_general_tab` - Allows adding options in general tab.

Twitter

Group the settings to work with the social network Twitter and includes a library to access the API for this social network.

Filters

- `fw_ext_social_twitter_boxes_options` - Provides the ability to add boxes and options in twitter tab.
- `fw_ext_social_twitter_general_box_options` - Allow you to add options in main box of twitter tab.

Helpers

- `fw_ext_social_twitter_get_connection` - Returns an instance of TwitterOAuth (see: https://github.com/abraham/twitteroauth), based on keys inserted.(Requires completing the following fields: consumer key, consumer secret, access token and access token secret).

Facebook

This sub-extension grouping settings for Facebook social network and offers the possibility to work with the Facebook Graph API.

Filters

- `fw_ext_social_facebook_boxes_options` - Provides the ability to add boxes and options in facebook tab.
- `fw_ext_social_facebook_general_box_options` - Allow you to add options in main box of facebook tab.

Helpers

- `fw_ext_social_facebook_graph_api_explorer` - Allows access Facebook Graph API.

6.4.13 Builder

This extension provides the core builder functionality that you can extend to create new builders.

Changing the grid

By default the Builder uses a bootstrap like grid, with the same class names but prefixed with `.fw-{bootstrap-class-name}`. The grid css is enqueued in all frontend pages from `framework/extensions/builder/static.php`. Also this extension defines the grid columns for all builders (for e.g. `page-builder` and `form-builder`) in `framework/extensions/builder/config.php`.

Changing the grid for all builders

1. Overwrite `framework/extensions/builder/config.php` by creating `{theme}/framework-customizations/extensions/builder/config.php`

```php
<?php if (!defined('FW')) die('Forbidden');

$cfg = array();

$cfg['default_item_widths'] = array(
    /**
     * Copy/Paste here default columns https://github.com/ThemeFuse/
↪Unyson-Builder-Extension/blob/master/config.php
     * and add, remove or change them
     */
);
```

2. Prevent default grid css enqueue and enqueue your own css. Create `{theme}/framework-customizations/extensions/builder/static.php`

```
<?php if (!defined('FW')) die('Forbidden');

if (!is_admin()) {
    wp_register_style(
        'fw-theme-frontend-grid',
        get_template_directory_uri() .'/framework-customizations/
↪extensions/builder/static/frontend-grid.css',
        array(),
        fw()->theme->manifest->get_version()
    );
}
```

Changing the grid for one builder

Other extensions use the `fw_ext_builder_get_item_width($builder_type, $width_id)` function to get and output grid css class in frontend

```
<div class="<?php echo esc_attr(fw_ext_builder_get_item_width('page-builder', '1_2/
↪frontend_class')) ?>" >
```

The function loads the grid from config, but allows you to change it via this filter. You can use the filter to change the grid columns for some builder type.

```
add_filter(
    'fw_builder_item_widths:page-builder',
    '_filter_theme_custom_page_builder_columns'
);
function _filter_theme_custom_page_builder_columns($columns) {
    $columns['3_7'] = array(
        'title' => '3/7',
        'backend_class' => 'custom-backend-3-7-column', // you must enqueue in␣
↪backend a css with this class
        'frontend_class' => 'frontend-custom-3-7-column', // you must enqueue in␣
↪frontend a css with this class
    );

    return $columns;
}
```

The Builder

The builder is just an option type. But you can't use it right away, because it's too abstract and doesn't have any concrete purpose. You can only extend it and create new builders based on it.

Data Structure

The javascript side of the builder is based on backbone, so it uses collections and models to store the data:

```
[
    {
        type: 'foo',
        _items: [],
        attr_x: 'Hello',
```

```
        ...
    },
    {
        type: 'bar',
        _items: [ {type: 'baz', ...}, ... ],
        attr_y: 'Hi',
        ...
    },
    ...
]
```

Every model (also called item) has a required attribute `type`. Also it has an attribute `_items` that is generated automatically by the backbone-relational plugin, the purpose of which is to make possible to have nested items easier. There are no rules for other attributes, every item has whatever attributes it wants.

The same data structure is used on the php side, this collection is simply transformed into an array with `json_decode($collection, true)`.

Creating a Builder

This tutorial will explain you how to create a simple demo builder for html `<ul>` and `<ol>` lists. First, create an option type that extends the builder option type:

```
// file: theme/inc/includes/option-types/lists-builder/class-fw-option-type-lists-
↪builder.php

class FW_Option_Type_Lists_Builder extends FW_Option_Type_Builder
{
    public function get_type() {
        return 'lists-builder';
    }
}
FW_Option_Type::register('FW_Option_Type_Lists_Builder');
```

That's it, the new builder was created. Use it in your post options to see what it shows at this point.

Note: This example assumes that you use in your theme this directory structure.

1. Include the option type:

```
// file: theme/inc/includes/lists-builder.php

/** @internal */
function _action_include_demo_lists_builder() {
    if (!fw_ext('builder')) {
        /**
         * Lists Builder requires the FW_Option_Type_Builder class
         * which does not exist if the 'builder' extension is not active.
         *
         * You can install and activate the 'builder' extension by␣
↪installing any extension that uses it,
         * for e.g. Page Builder or Learning (which has the Learning Quiz␣
↪Builder sub-extension)
         */
        return;
```

```
    }

    require_once dirname(__FILE__) .'/option-types/lists-builder/class-fw-
↪option-type-lists-builder.php';
}
add_action('fw_init', '_action_include_demo_lists_builder', 9);
```

2. Add it in post options:

```
// file: theme/framework-customizations/theme/options/posts/post.php

$options = array(
    'lists-builder-box' => array(
        'type' => 'box',
        'title' => __('Lists Builder', '{domain}'),
        'options' => array(
            'lists-builder' => array(
                'type' => 'lists-builder',

                // this will make it full width
                'label' => false,
            ),
        ),
    ),
);
```

3. Go to `your.site/wp-admin/edit.php` page, open any post edit page and look for the "Lists Builder" box.

As you can see, the box is empty. At least you've successfully created the builder, now you can improve it.

Creating Items

To build lists you'll need the following elements: `<ul>`, `<ol>` and `<li>`. In builder these elements can be created as item types. The `<ul>` and `<ol>` (containers for `<li>`) will be created as one item type (with sub types), and `<li>` as another item type. To create item types for a builder type you have to:

1. Find out what item types the builder accepts.

   That information can be found in the `FW_Option_Type_Builder::item_type_is_valid()` method. The builder you created above doesn't have a custom `item_type_is_valid()` method, so it is inherited from the extended class, and that method looks like this:

```
/**
 * Overwrite this method to force your builder type items to extend␣
↪custom class or to have custom requirements
 * @param FW_Option_Type_Builder_Item $item_type_instance
 * @return bool
 */
protected function item_type_is_valid($item_type_instance)
{
    return is_subclass_of($item_type_instance, 'FW_Option_Type_Builder_
↪Item');
}
```

2. Register item types.

Create and register item type that will represent the <ul> and <ol> elements:

```
// file: theme/inc/includes/option-types/lists-builder/item-types/oul/
↪class-fw-lists-builder-item-type-oul.php

class FW_Lists_Builder_Item_Type_OUl extends FW_Option_Type_Builder_Item
{
    /**
     * Specify which builder type this item type belongs to
     * @return string
     */
    public function get_builder_type()
    {
        return 'lists-builder';
    }

    /**
     * The item type
     * @return string
     */
    public function get_type()
    {
        return 'oul';
    }

    /**
     * The boxes that appear on top of the builder and can be dragged␣
↪down or clicked to create items
     * @return array
     */
    public function get_thumbnails()
    {
        return array(
            array(
                'html' =>
                    '<div class="item-type-icon-title" data-sub-type="ul">
↪'.
                    '    <div class="item-type-icon">&lt;ul&gt;</div>'.
                    '    <div class="item-type-title">'. __('Unordered␣
↪List', '{domain}') .'</div>'.
                    '</div>',
            ),
            array(
                'html' =>
                    '<div class="item-type-icon-title" data-sub-type="ol">
↪'.
                    '    <div class="item-type-icon">&lt;ol&gt;</div>'.
                    '    <div class="item-type-title">'. __('Ordered List
↪', '{domain}') .'</div>'.
                    '</div>',
            ),
        );
    }

    /**
     * Enqueue item type scripts and styles
     */
    public function enqueue_static()
    {
```

```
    }
}
FW_Option_Type_Builder::register_item_type('FW_Lists_Builder_Item_Type_OUl
↪');
```

Create and register item type that will represent the <li> element:

```
// file: theme/inc/includes/option-types/lists-builder/item-types/li/
↪class-fw-lists-builder-item-type-li.php

class FW_Lists_Builder_Item_Type_Li extends FW_Option_Type_Builder_Item
{
    public function get_builder_type()
    {
        return 'lists-builder';
    }

    public function get_type()
    {
        return 'li';
    }

    public function get_thumbnails()
    {
        return array(
            array(
                'html' =>
                    '<div class="item-type-icon-title">'.
                    '    <div class="item-type-icon">&lt;li&gt;</div>'.
                    '    <div class="item-type-title">List Item</div>'.
                    '</div>',
            ),
        );
    }

    public function enqueue_static()
    {
    }
}
FW_Option_Type_Builder::register_item_type('FW_Lists_Builder_Item_Type_Li
↪');
```

3. Include the created files.

   At the end of the _action_include_demo_lists_builder() function (created above), add:

```
// file: theme/inc/includes/lists-builder.php

function _action_include_demo_lists_builder() {
    ...

    require_once dirname(__FILE__) .'/option-types/lists-builder/item-
↪types/oul/class-fw-lists-builder-item-type-oul.php';
    require_once dirname(__FILE__) .'/option-types/lists-builder/item-
↪types/li/class-fw-lists-builder-item-type-li.php';
}
```

Refresh the page and you should see three boxes that can be dragged down. Unfortunately you will get an error in console saying that the item type is not registered. This happens because you also have to register the item type in javascript and define how it works and looks in builder.

Registering items in javascript

Registering builder items can be done via the `builderInstance.registerItemClass(ItemTypeClass)` method. Because `builderInstance` is created somewhere in builder scripts and it's not a global variable, the only way to get it, is to listen special event `fw-builder:{builder-type}:register-items`.

1. Create the scripts file that registers the `oul` item type:

```
// file:: theme/inc/includes/option-types/lists-builder/item-types/oul/
↪static/scripts.js

fwEvents.one('fw-builder:'+ 'lists-builder' +':register-items',␣
↪function(builder) {
    var ItemClass = builder.classes.Item.extend({
        defaults: {
            type: 'oul' // the item type is specified here
        }
    });

    builder.registerItemClass(ItemClass);
});
```

2. Enqueue the `oul` item type scripts file:

```
class FW_Lists_Builder_Item_Type_OUl extends FW_Option_Type_Builder_Item
{
    ...

    public function enqueue_static()
    {
        wp_enqueue_script(
            'lists-builder-item-type-oul',
            get_template_directory_uri() .'/inc/includes/option-types/
↪lists-builder/item-types/oul/static/scripts.js',
            array('fw-events')
        );
    }
}
```

3. Create the scripts file that registers the `li` item type:

```
// file:: theme/inc/includes/option-types/lists-builder/item-types/li/
↪static/scripts.js

fwEvents.one('fw-builder:'+ 'lists-builder' +':register-items',␣
↪function(builder) {
    var ItemClass = builder.classes.Item.extend({
        defaults: {
            type: 'li' // the item type is specified here
        }
    });
```

```
    builder.registerItemClass(ItemClass);
});
```

4. Enqueue the `li` item type scripts file:

```
class FW_Lists_Builder_Item_Type_Li extends FW_Option_Type_Builder_Item
{
    ...

    public function enqueue_static()
    {
        wp_enqueue_script(
            'lists-builder-item-type-li',
            get_template_directory_uri() .'/inc/includes/option-types/
↪lists-builder/item-types/li/static/scripts.js',
            array('fw-events')
        );
    }
}
```

Refresh the page and try to click or drag down the boxes. The items should appear in the builder, but they are using the default view and doesn't have any concrete functionality. At this point, you have a working builder. If you add some items and save the post, after page refresh the builder will recover from the saved json value. Customize the views and add some functionality to items to be able to build lists with them:

1. Replace the `oul` item type scripts with:

```
// file: theme/inc/includes/option-types/lists-builder/item-types/oul/
↪static/scripts.js

fwEvents.one('fw-builder:'+ 'lists-builder' +':register-items',␣
↪function(builder) {
    var ItemView = builder.classes.ItemView.extend({
        template: _.template(
            '<div style="border: 1px solid #ccc; padding: 0 10px;">'+
                '<p>&lt;<span><%- type %></span>&gt; <a href="#" onclick=
↪"return false;" class="dashicons fw-x"></a></p>'+

                /**
                 * Special element with 'builder-items' class
                 * displays the items that are in the '_items' attribute␣
↪of the model
                 */
                '<div class="builder-items"><!-- list items --></div>'+
            '</div>'
        ),
        render: function() {
            // It is recommended to do the template render using this␣
↪method
            this.defaultRender({
                type: this.model.get('list_type')
            });
        }
    });

    var ItemClass = builder.classes.Item.extend({
        defaults: {
            type: 'oul', // the item type is specified here
```

```
            list_type: 'ul'
        },
        initialize: function(atts, opts) {
            if (opts && opts.$thumb) {
                /**
                 * When the item box is dragged down or clicked, opts.
↪$thumb contains the box element
                 * so you can extract the data-sub-type attribute set in␣
↪html.
                 *
                 * Note: opts.$thumb doesn't exist when the item is␣
↪created from code
                 * for e.g. recovered from json after page refresh
                 */
                this.set('list_type', opts.$thumb.find('[data-sub-type]').
↪attr('data-sub-type'));
            }

            this.view = new ItemView({
                id: 'lists-builder-item-'+ this.cid,
                model: this
            });

            // it is recommended to call this method
            this.defaultInitialize();
        },
        /**
         * This method controls which item types are allowed to be added␣
↪inside this item in the '_items' attribute
         * @param {String} type
         * @returns {boolean}
         */
        allowIncomingType: function(type) {
            if (type == 'li') {
                return true;
            } else {
                return false;
            }
        }
    });

    builder.registerItemClass(ItemClass);
});
```

2. Replace the `li` item type scripts with:

```
// file: theme/inc/includes/option-types/lists-builder/item-types/li/
↪static/scripts.js

fwEvents.one('fw-builder:'+ 'lists-builder' +':register-items',␣
↪function(builder) {
    var ItemView = builder.classes.ItemView.extend({
        template: _.template(
            '<div style="border: 1px solid #ccc; padding: 0 10px;">'+
            '<p>'+
                '<span><%= text %></span> '+
                '<a href="#" onclick="return false;" class="dashicons␣
↪dashicons-edit"></a>'+
```

```
                '<a href="#" onclick="return false;" class="dashicons fw-x
↪"></a>'+
            '</p>'+
            '</div>'
        ),
        events: {
            'click a.dashicons.fw-x': 'defaultRemove',
            'click .dashicons-edit': 'openTextEdit'
        },
        render: function() {
            this.defaultRender({
                text: this.model.get('text')
            });
        },
        openTextEdit: function() {
            var text = prompt('Edit <li> text', this.model.get('text'));

            if (text === null) {
                return;
            }

            this.model.set('text', text);
        }
    });

    var ItemClass = builder.classes.Item.extend({
        defaults: {
            type: 'li', // the item type is specified here
            text: 'Hello World!' // <li>{text}</li>
        },
        initialize: function(atts, opts) {
            this.view = new ItemView({
                id: 'lists-builder-item-'+ this.cid,
                model: this
            });

            this.defaultInitialize();
        },
        /**
         * This method controls to which item types this item is allowed␣
↪to be added/moved
         * @param {String} type
         * @returns {boolean}
         */
        allowDestinationType: function(type) {
            if (type == 'oul') {
                return true;
            } else {
                return false;
            }
        }
    });

    builder.registerItemClass(ItemClass);
});
```

Now the javascript side of the builder has the minimum functionality to be able to build lists. After you build a list and saved the post, the html of the list needs to be generated so you can display it on the page. To do that, continue to the

next step.

Generate Custom Value

By default the builder saves its value as an array with one key `json` which stores the original value used in javascript. From the original value, you can generate any custom values and store them in custom keys. In the case with Lists Builder, you have to generate the lists html from that original json value to be able to display the list in html. This can achieved by overwriting the builder `_get_value_from_input()` method.

```
class FW_Option_Type_Lists_Builder extends FW_Option_Type_Builder
{
    ...

    /**
     * Generate the html of the list
     * {@inheritdoc}
     */
    protected function _get_value_from_input($option, $input_value)
    {
        $value = parent::_get_value_from_input($option, $input_value);

        $html = '';
        foreach (json_decode($value['json'], true) as $list) {
            $html .= '<'. $list['list_type'] .'>';

            foreach ($list['_items'] as $list_item) {
                $html .= '<li>'. $list_item['text'] .'</li>';
            }

            $html .= '</'. $list['list_type'] .'>';
        }
        $value['html'] = $html;

        return $value;
    }
}
```

Now you can use the generated html in post template. Add to `theme/single.php`:

```
...

while ( have_posts() ) : the_post();

    echo fw_get_db_post_option( null, 'lists-builder/html' );

...
```

Congratulations, now you can create new builders!

There are many things that can be improved in the Lists Builder, but this article will become too big. You can inspect the builder code and other builders like Page Builder, Forms Builder and Learning Quiz Builder to find the answers for the questions that may appear while developing your own builder.

6.4.14 Feedback

The extension adds the possibility for users to leave feedback impressions about a post (product, article, etc). This system can be activated for some post types, and replaces the default comments system.

Helpers

- `fw_ext_feedback()` - displays summary information about the feedback received for a specific post.

Views

- `reviews.php` - the template for displaying reviews.

Hooks

- `fw_ext_feedback` - allows you to add summary information about the feedback received for a specific post.
- `fw_ext_feedback_listing_walker` - provides the ability to send a custom walker class object to use the when rendering the reviews.

```
    /** @internal */
function _filter_fw_ext_feedback_listing_walker() {
            require dirname( __FILE__ ) . '/includes/extends/class-fw-feedback-stars-
↪walker.php';

            return new FW_Feedback_Stars_Walker();
    }
    add_filter( 'fw_ext_feedback_listing_walker', '_filter_fw_ext_feedback_listing_
↪walker' );
```

Stars Feedback

The `feedback-stars` is a child extension that allows visitors to appreciate a post using star rating.

Helpers

- `fw_ext_feedback_stars_get_post_rating()` - returns brief information about the post's votes.
- `fw_ext_feedback_stars_get_post_detailed_rating()` - returns detailed information about the post's votes.

Views

- `rate.php` - display the stars in the form of introduction of review.
- `rate.php` - displays information about the votes allocated to a post.
- `rate.php` - output a single review in the HTML5 format.
- `rate.php` - output a single review.

6.4.15 Learning

This extension adds a Learning module to your theme. Using this extension you can add courses, lessons and tests for your users to take.

Config

From config file you can edit the lesson, course and course category taxonomy slugs.

```
$cfg['slugs'] = array(
    'courses'    => 'course',
    'lessons'    => 'lesson',
    'categories' => 'courses',
);
```

Views

Templates are located in the `views/` directory. Here is the list of templates that you can customize:

- `single-course.php` - Learning course single post template. By default is used `single.php` from the theme root directory, you can overwrite it by creating `framework-customizations/extensions/learning/views/single-course.php`.
- `single-lesson.php` - Learning lesson single post template. By default is used `single.php` from the theme root directory, you can overwrite it by creating `framework-customizations/extensions/learning/views/single-lesson.php`.
- `taxonomy.php` - Learning category template. By default is used `taxonomy.php` from the theme root directory, you can overwrite it by creating `framework-customizations/extensions/learning/views/taxonomy.php`.
- `content-course.php` - Default learning course single page template content. It is loaded if the `framework-customizations/extensions/learning/views/single-course.php` doesn't

exist and is used `single.php` from the theme root directory. The content of this view is rendered using WordPress the_content filter, when the course single page is loaded.

- `content-lesson.php` - Default learning lesson single page template content. It is loaded if the `framework-customizations/extensions/learning/views/single-lesson.php` doesn't exist and is used `single.php` from the theme root directory. The content of this view is rendered using WordPress the_content filter, when the lesson single page is loaded.

Helpers

- `fw_ext_learning_get_course_lessons()` - Returns an array with all course lesson posts.

```
/**
 * @param null|int $post_id The id of the course post
 * @return WP_Post[]
 */
$lessons = fw_ext_learning_get_course_lessons( $post_id );
```

- **`fw_ext_learning_get_previous_lesson()` - Returns the previous lesson post.** This function is similar to previous_post_link() WordPress function, but it returns the entire post object.

Attention: Do not use the `previous_post_link()` function to get previous lesson link, you'll not get the desired result.

```
/**
 * @param null|int $post_id (optional) The id of the course post
 * @return WP_Post[]|null - in case there are no previous posts, or $post_id is not a␣
↪valid lesson post
 */
$prev = fw_ext_learning_get_previous_lesson( $post_id );
```

- **`fw_ext_learning_get_next_lesson()` - Returns the next lesson post.** This function is similar to next_post_link() WordPress function, but it returns the entire post object.

Attention: Do not use the `next_post_link()` function to get next lesson link, you'll not get a the desired result.

```
/**
 * @param null|int $post_id (optional) The id of the course post
 * @return WP_Post[]|null - in case there are no previous posts, or $post_id is not a␣
↪valid lesson post
 */
$prev = fw_ext_learning_get_next_lesson( $post_id );
```

Usage example

If you edit the lesson template and want to make a pagination to next and previous lessons.

```
<?php
global $post;
```

```
$prev = fw_ext_learning_get_previous_lesson( $post->ID );
$next = fw_ext_learning_get_next_lesson( $post->ID );
?>
<nav class="lesson-nav">
    <a class="prev" href="<?php get_permalink($prev->ID)?>"><?php _e( 'Previous lesson
↪', '{domain}' )?></a>
    <a class="next" href="<?php get_permalink($next->ID)?>"><?php _e( 'Next lesson', '
↪{domain}' )?></a>
</nav>
```

Filters

- `fw_ext_learning_lessons_label_name` - Rename lesson custom post default name (singular and plural).

```
/** @internal */
function _filter_fw_ext_learning_rename_lesson_custom_post( $names ) {
    $names['singular'] = __( 'Singular Name', '{domain}' );
    $names['plural'] = __( 'Plural Name', '{domain}' );

    return $names;
}
add_filter( 'fw_ext_learning_lessons_label_name', '_filter_fw_ext_learning_rename_
↪lesson_custom_post' );
```

- `fw_ext_learning_courses_label_name` - Rename course custom post default name (singular and plural).

```
/** @internal */
function _filter_fw_ext_learning_rename_course_custom_post( $names ) {
    $names['singular'] = __( 'Singular Name', '{domain}' );
    $names['plural'] = __( 'Plural Name', '{domain}' );

    return $names;
}
add_filter( 'fw_ext_learning_courses_label_name', '_filter_fw_ext_learning_rename_
↪course_custom_post' );
```

- `fw_ext_courses_category_name` - Rename course custom post category default name (singular and plural).

```
/** @internal */
function _filter_fw_ext_learning_rename_course_custom_post_category( $names ) {
    $names['singular'] = __( 'Singular Name', '{domain}' );
    $names['plural'] = __( 'Plural Name', '{domain}' );

    return $names;
}
add_filter( 'fw_ext_courses_category_name', '_filter_fw_ext_learning_rename_course_
↪custom_post_category' );
```

FW_Extension_Learning class

The `FW_Extension_Learning` is the Learning extension base class and in development process it may offer a lot of great methods to make the development easier. You'll need the current instance of the

`FW_Extension_Learning`. You can get it using the `fw_ext('extension_name')` function:

```
/**
 * @var FW_Extension_Learning $learning
 */
$learning = fw_ext('learning');
```

Do not forget to check the the result is not `null`, this happens when the extension is not active.

Methods

- `get_course_post_type()` - Returns the courses post type name.

```
/**
 * @var string $type The course custom post type
 */
$type = $learning->get_course_post_type();
```

- `get_course_slug()` - Returns the courses post type slug.
- `get_lesson_post_type()` - Returns the lesson post type name.
- `get_lessons_slug()` - Returns the lesson post type slug.
- `get_categories_taxonomy()` - Returns the course post type taxonomy name.
- `get_categories_slug()` - Returns the course post type taxonomy slug.
- `is_course($post_id)` - Check if the post is a course post type.

```
if( $learning->is_course( $post_id ) ) {
    ...
}
```

Learning Quiz

A sub-extension of the Learning extension that offers users the possibility to build tests and quiz for lessons.

Views

Templates are located in the `views/` directory. Here is the list of templates that you can customize:

- `start-quiz.php` - Quiz star button from the lesson page.
- `single.php` - Learning quiz single post template. By default is used `single.php` from the theme root directory, you can overwrite it by creating `framework-customizations/extensions/learning/extensions/learning-quiz/views/single.php`.

- `content.php` - Default learning quiz single page template content. It is loaded if the `framework-customizations/extensions/learning/extensions/learning-quiz/views/single.php` doesn't exist and is used `single.php` from the theme root directory. The content of this view is rendered using WordPress the_content filter, when the lesson single page is loaded.

Helpers

- `fw_ext_learning_quiz_has_quiz( $post_id )` - Check if the post is lesson and if it has a quiz.

```
if ( fw_ext_learning_quiz_has_quiz( $post_id ) ) { ... }
```

- `fw_ext_learning_quiz_get_quiz( $post_id )` - Return the quiz of the lesson.

```
/**
 * @param int $post_id
 * @return WP_Post|null - in case the the id is not valid or is not lesson post type,␣
↪or the lesson doesn't have a quiz.
 */
$quiz = fw_ext_learning_quiz_get_quiz( $post_id );
```

- `fw_ext_learning_quiz_get_quiz_permalink( $post_id )` - Return the permalink of the quiz post.
- `fw_ext_learning_quiz_get_response( $post_id )` - After the quiz form is submitted, it returns a response after processing the quiz.

```
/**
 * @param int $post_id
 * @return array(
 *  questions => FW_Quiz_Question_Process_Response[]
 *  accumulated => (int|float) The amount of points accumulated
 *  minimum-pass-mark - (int|float) The minimum pass-mark
 * )
 */
$response = fw_ext_learning_quiz_get_response( $post_id );
```

Actions

- `fw_ext_learning_quiz_form_process` - Action fired when the quiz form was submitted and processed.

```
/**
 * @internal
 * @param int $post_id
 * @return array(
 *  questions => FW_Quiz_Question_Process_Response[]
 *  accumulated => (int|float) The amount of points accumulated
 *  minimum-pass-mark - (int|float) The minimum pass-mark
 * )
 */
function _action_fw_process_quiz_response( $response ) {
    // ...
}
add_action( 'fw_ext_learning_quiz_form_process', '_action_fw_process_quiz_response' );
```

6.4.16 Translation

This extension lets you translate your website in any language or even add multiple languages for your users to change at their will from the front-end.

Helpers

- `fw_ext_translation_get_frontend_active_language()` - Frontend active language.
- `fw_ext_translation_get_backend_active_language()` - Backend active language.

Filters

- `fw_ext_translation_change_render_language_switcher` - Change the view of the language switcher.

```
add_action( 'fw_ext_translation_change_render_language_switcher',␣
↪function ( $html, $frontend_urls ) {
    $html = '';

    foreach ( $frontend_urls as $lang_code => $url ) {
        $html .= '<a href="' . esc_attr($url) . '">' . $lang_code . '</a>
↪';
    }

    return $html;
}, 10, 2 );
```

6.4.17 WordPress Shortcodes

This extensions gives a way to insert and render correctly Unyson shortcodes inside WordPress editor.

Understanding the purpose of this extension

At first, this extension is not a silver bullet for all of the use cases you may want to try, it is quite limited in what it can achieve. If you want to get more details you should really go and read the whole discussion on GitHub.

Warning: This document is a work in process.

The structure of a Unyson shortcode

At first, you should know that an Unyson shortcode consists of three parts that make him look the way it does:

1. HTML. Usually it is located in `view.php`
2. All of the static. Located in `static.php`
3. Dynamic CSS. Enqueueud with `wp_add_inline_style` on `'fw_ext_shortcodes_enqueue_static:{name}'`

Depending on your use case, it may be easier or harder to get those components rendered correctly. I'll give a short table below that will make all of this clear.

1. Shortcode in Post Editor
2. Shortocde in `wp-editor` option type of any Page Builder Shortcode
3. Shortocde in `wp-editor` that is inserted anywhere else (like Theme Settings or any `OptionsModal`)

use case vs. what you get	HTML	`static.php`	dynamic css
1 Post Editor	yes	yes	yes
2 Page Builder Shortcode	yes	yes	no
3 Any `wp-editor`	yes	no	no

Shortcodes in main post editor

By default, you'll a get a button in the main post editor with all of the shortcodes that are enabled, except the `section` and `column` ones. This is actually the most simple use-case and you have nothing to do in order to get them working. Everything should be out of the box here.

You can in fact, customize which shortcodes are showed up using this snippet of code:

```
<?php if (!defined('FW')) die('Forbidden');

add_filter('fw:ext:wp-shortcodes:default-shortcodes', _set_default_shortcodes);

function _set_default_shortcodes($previous_shortcodes) {
    return array( 'button', 'notification' );
}
```

Shortcodes in another Page Builder Shortcode

6.5 Bumblebee Extensions

While a good deal of features used in theme development is pure Unyson, some functionality needs Bumblebee to work. Here you can find Bumblebee extensions documented.

6.5.1 Bee Widgets

Bee Widgets extension allows for easy widget creation by converting an existing shortcodes to widgets. It's an easier and usually cleaner way than creating a widget from scratch.

Note: Bee Widget extension is automatically enabled when Theme Plugin is active.

Create sample widget

Let's create a `Theme socials` widget based on `ct_socials` shortcode. In order to add a widget to the theme, create the following file in Theme Plugin structure: `wp-content/plugins/theme-plugin/extensions/bee-widgets/includes/class-bee-widget-ct-socials.php`

Important: There should be `ct_socials` shortcode defined in theme plugin already in the example path: `wp-content\plugins\theme-plugin\extensions\shortcodes\shortcodes\ct-socials`

Note: The name of the file should be `class-bee-widget-widgetname.php`. Thus it will be autoincluded and autoregistered.

`class-bee-widget-ct-socials.php` content:

```
<?php

/** Widget extending Bee Widget class */
class Bee_Widget_Ct_Socials extends Bee_Widget
{
    /**
     * Set the name of a shortcode which will be converted to widget
     * Throws an exception if specified shortcode not found
     * @return string
     */
    protected function get_bee_shortcode_name() {
        return 'ct_socials';
    }

    /**
     * Set desired widget name
     * @return string
     */
    protected function get_bee_widget_name() {
        return 'Theme socials';
    }

    /**
     * Set desired widget description
     * @return string
     */
    protected function get_bee_widget_description() {
        return 'Theme socials description';
    }

}
```

```
/** Widget will be autoregistered if put in proper file and location.
  * Otherwise, you can manually register the widget in WordPress with the code below:

    Bee_Widget_Ct_Socials::init_register();

**/
```

Note: Methods `get_bee_widget_name` and `get_bee_widget_description` are optional to overwrite.

That's all! The widget should now be available in wp-admin

Hooks

Besides methods above, there are also filters:

- `bee_widget_options` which allows you to change widget options,
- `fw_ext_bee_widgets_paths` which allows you to look for widgets in custom locations
- `fw_ext_visual_composer_map_skip_shortcodes` which allows you to hide shortcodes/widgets from VC editor

which is useful when designing a shortcode for widget context only. For more information, see Visual Composer Integration

Option dependency

Option dependency can be used in the same way as in shortcodes. Currently supported option types are most simple inputs including: text, select, radio, checkbox.

6.5.2 Visual Composer

Unyson shortcodes are automatically translated to Visual Composer shortcodes by Visual Composer extension.

Important: Visual Composer extension has to be activated in wp-admin Unyson settings menu.

You can read about Visual Composer shortcode related settings at Visual Composer Integration

Hooks

_filter_fw_visual_composer_extend_shortcodes

Should you need to modify an existing VC shortcode, this filter may be of help.

Here's an example code which adds `text` option to existing `ct_button` and `ct_icon` shortcodes (in `theme-plugin/extensions/visual-composer/hooks.php` file) :

```
<?php

/** Visual composer related hooks, ie. add/modify VC shortcode options */

add_filter( '_filter_fw_visual_composer_extend_shortcodes', 'fw_ext_visual_composer_
↪extend_shortcodes' );
function fw_ext_visual_composer_extend_shortcodes( $shortcodes_params ) {

    $shortcodes_params['ct_button'] = array(
            'my_title' => array(
                    'type'  => 'text',
                    'label' => esc_html__( 'Title', 'ct_theme' ),
                    'value' => 'My title',
            )
    );

    $shortcodes_params['ct_icon'] = array(
            'my_title' => array(
                    'type'  => 'text',
                    'label' => esc_html__( 'Title', 'ct_theme' ),
                    'value' => 'My title',
            )
    );

    return $shortcodes_params;
}
```

fw_ext_visual_composer_map_skip_shortcodes

You can skip mapping for selected shortcodes so they won't show up in VC Editor. This is helpful for hiding shortcodes you only use for widgets. Use `fw_ext_visual_composer_map_skip_shortcodes` filter for that. Example:

```
add_filter( 'fw_ext_visual_composer_map_skip_shortcodes', 'ct_bee_fw_filter_visual_
↪composer_map_skip_shortcodes' );
function ct_bee_fw_filter_visual_composer_map_skip_shortcodes( $shortcodes ) {
        array_push( $shortcodes, 'ct_flickr', 'ct_twitter', 'ct_latest_news', 'ct_
↪socials' );
        return $shortcodes;
}
```

fw_ext_visual_composer_map_shortcode_options

You can filter every shortcode options before they are mapped to Visual Composer.

fw_ext_visual_composer_mapper_shortcode_skip_css_editor

If you use `css_editor` type shortcode param in pair with `shortcode` type shortcode option, JS conflict may occur. Therefore, the default behaviour is to skip `css_editor` param inheritance. With this hook you can change that behaviour, though it is not recommended.

Native hooks

You may also use native Visual Composer hooks. Read more about them:

- Change default value of a parameter
- Change a shortcode's HTML output

VC Grid Item Elements

The following grid item elements are available as part of Bumblebee

Post Taxonomy

This elements displays current post's taxonomy terms. To add it to a grid item, go to Visual Composer -> Grid Builder and add a Post Taxonomy element. There, you will be able define taxonomy name and number of terms to display.

6.5.3 Sass Compiler

Sass Compiler extension allows users to use Customizer options to control theme styles (SCSS) variables such as theme motive color, as well as automatic sass recompilation during Build process.

Important: Sass Compiler extension is disabled by default. User needs to activate it in wp-admin Unyson settings menu.

Configuration

In order to configure Sass Compiler in your project, create the following file: `wp-content/themes/projectname/theme/framework-customizations/extensions/sass-compiler/config.php`

With the following content:

```
<?php if ( ! defined( 'FW' ) ) {
    die( 'Forbidden' );
}

$uploads = wp_upload_dir();

$cfg = array(
    'handles'      => array( 'fw-ct-bee-sass-style' => 'style.scss' ),
    'sass_dir'     => get_stylesheet_directory() . '/assets/sass', // where are␣
↪stored SCSS files
    'css_dir'      => get_stylesheet_directory() . '/assets/css',  // where to store␣
↪CSS files
```

```
    'formatter'     => defined( 'WP_DEBUG' ) && WP_DEBUG ?
↪'Leafo\ScssPhp\Formatter\Expanded' : 'Leafo\ScssPhp\Formatter\Crunched',
    'css_save_dir' => $uploads['path'],
    'css_uri'      => $uploads['url'],
);
```

This assumes you have enqueued a style handle `fw-ct-bee-sass-style` with a path of `wp-content/themes/projectname/theme/assets/sass/style.scss` and the path to the theme custom CSS file is `wp-content/themes/projectname/theme/assets/css/style.css`

That's basically it.

Customizer Variables

Let's assume you have `$motive` variable in your SCSS file. In order to control it in Customizer, two things are required:

- Customizer option has `ct-sass` param with the value of your variable name
- For fonts use `ct-sass-fonts` param with the value of your variable name in scss files

Content of `wp-content/themes/projectname/theme/framework-customizations/theme/options/customizer.php`:

```
$options = array(
    'theme' => array(
        'title'   => esc_html__( 'General', 'ct_theme' ),
        'options' => array(
            'motive' => array(
                'type'    => 'color-picker',
                'label'   => esc_html__( 'Pick the motive color', 'ct_theme' ),
                'desc'    => esc_html__( 'In order to use this feature, please have␣
↪Sass Compiler enabled in Unyson extensions', 'ct_theme' ),
                'value'   => '#00bcdf',
                'ct-sass' => 'motive'
            ),
        )
    ),
);
```

- SCSS variable has a `!default` directive

Content of `wp-content/themes/projectname/theme/assets/sass/partials/_motive.scss`:

```
$motive : #00bcdf !default;
```

Tip: As in the example, you can define SCSS variables in convenient partials which can be included in style.scss.

Note: Don't forget to enable Sass Compiler extension in wp-admin Unyson menu before testing.

Hooks

Hooks should be added to `wp-content/themes/project/theme/framework-customizations/extensions/sass-compiler/hooks.php`

fw_ext_sass_compiler_variables

Should you ever need to use assets path variable in SCSS file, you may find `fw_ext_sass_compiler_variables` filter useful.

```
<?php

/**
 * Set path to css assets folder to fix relative paths in css compiled to /uploads
 */
add_filter( 'fw_ext_sass_compiler_variables', 'fw_ct_bee_filter_theme_fw_ext_sass_
↪compiler_variables', 10, 2 );
function fw_ct_bee_filter_theme_fw_ext_sass_compiler_variables( $vars, $compiler ) {
    $vars['$path'] = get_stylesheet_directory_uri() . '/assets/css/';
    $vars['$path'] = str_replace( 'http://', '//', $vars['$path'] );
    $vars['$path'] = str_replace( 'https://', '//', $vars['$path'] );

    return $vars;
}
```

6.5.4 Post Types Extensions

When creating a custom post type in a theme, it is neat to put it in an extension. There are many already developed extensions containing custom post types, ie.

- Portfolio extension (post type `ct-portfolio`)
- Team Members extension (`ct-team`)
- FAQ extension (`ct-faq`)
- Testimonials extension (`ct-testimonials`)
- Shop Locator extension (`ct-shop-locator`)

Important: All of these extensions need to be activated in wp-admin Unyson settings menu in order to work.

Structure

All above extensions are similar in structure which follows guidelines: Extensions, for example here's a structure of Testimonials extension:

```
ct-testimonials/
├─class-fw-extension-ct-testimonials.php
├─manifest.php
├─options.php
├─posts.php
├─shortcodes/
  ├─ct-testimonials-slider/
```

```
    ├─class-fw-shortcode-ct-testimonials-slider.php
    ├─config.php
    ├─options.php
    ├─static.php
    ├─view/
    │  ├─view.php
    ├─static/
      ├─js/
        ├─init.js
```

Features

Activating any of those extensions adds custom post type features in wp-admin menu to add new posts and taxonomies. Most of them also add shortcodes (in the example `ct-testimonials-slider`) which render their own view.

Views

If needed, you can create single or archive custom post type view as documented in Custom Post Types. In the extension above, there are no views except for the shortcode view.

Static

As you can see in the structure, there is `init.js` file inside `shortcodes` directory. It could be placed inside `ct-testimonials/static/` directly, then it would be enqueued always when the extension is active. The way it is however, it is only enqueued when the shortcode is rendered as it is not needed on other pages. This is done automatically by the shortcode class.

Extension class

The extension class is basically a controller for post metaboxes, post data being sent to the view, as well as basic post type settings, as name, slug, taxonomy etc.

```php
<?php if ( ! defined( 'FW' ) ) {
    die( 'Forbidden' );
}

class FW_Extension_CT_Testimonials extends FW_Extension {
    private $post_type = 'ct-testimonials';

    /**
     * @var string
     */
    private $testimonials_slug = 'testimonials';

    /**
     * @var string
     */
    private $testimonials_thumbnail = 'thumbnail';

    /**
     * @internal
     */
    public function _init() {
```

```
        if ( is_admin() ) {
            $this->admin_filters();
        }
    }

    /**
     * Return the testimonials custom post type
     * @return string
     */
    public function get_post_type() {
        return $this->post_type;
    }

    /**
     * Return the testimonials custom post slug
     * @return string
     */
    public function get_testimonials_slug() {

        return $this->testimonials_slug;
    }

    public function get_testimonial_thumbnail() {
        return $this->testimonials_thumbnail;
    }

    /**
     * Return the testimonials data
     *
     * @param array $args
     *
     * @return array
     */
    public function get_posts_data( $args = array() ) {
        $collector = array();

        $queryArgs = array(
            'post_type'   => $this->get_post_type(),
            'numberposts' => - 1,
        );
        $queryArgs = array_merge( $queryArgs, $args );
        $posts     = get_posts( $queryArgs );

        foreach ( $posts as $post ) {
            setup_postdata( $post );

            array_push( $collector, (object) array(
                'desc'          => get_the_content( $post->ID ),
                'thumbnail'     => get_the_post_thumbnail( $post->ID ),
                'thumbnail_id'  => get_post_thumbnail_id( $post->ID ),
                'thumbnail_url' => get_the_post_thumbnail_url( $post->ID ),
                'meta_options'  => (object) fw_get_db_post_option( $post->ID )
            ) );

        }
        wp_reset_postdata();

        return $collector;
```

```
    }


    /**
     * @internal
     *
     * @param array $options
     * @param string $post_type
     *
     * @return array
     */
    public function _filter_admin_add_testimonial_options( $options, $post_type ) {
        if ( $post_type !== $this->get_post_type() ) {
            return $options;
        }

        $fields = array(
            'author' => array(
                'label' => __( 'Name', 'ct_theme' ),
                'type'  => 'text',
                'value' => 'John Doe',
            ),

            'position' => array(
                'label' => __( 'Company', 'ct_theme' ),
                'type'  => 'text',
                'value' => 'Company inc.',
            ),
        );

        $tabs = array(
            'demo' => array(
                'title'   => __( 'Author', 'ct_theme' ),
                'type'    => 'tab',
                'options' => array(
                    $fields
                ),
            ),
        );

        if ( empty( $options ) ) {
            $options = $tabs;
        } else {
            $currentOptions = current( $options );
            $currentKey     = key( $currentOptions['options'] );
            array_unshift( $options[ $currentKey ]['options'], $tabs );
        }

        return $options;
    }

    private function admin_filters() {
        add_filter( 'fw_post_options', array( $this, '_filter_admin_add_testimonial_
↪options' ), 10, 2 );
    }

}
```

Post type register

In `posts.php` the post type and taxonomies registration takes place. Keep this file as generic as you can to easily copy from one project to another.

```
<?php if ( ! defined( 'FW' ) ) {
    die( 'Forbidden' );
}

register_post_type( fw()->extensions->get( 'ct-testimonials' )->get_post_type(), array(
    'labels'              => array(
        'name'               => __( 'Testimonials', 'ct_theme' ),
        'singular_name'      => __( 'Testimonial', 'ct_theme' ),
        'menu_name'          => __( 'Testimonials', 'ct_theme' ),
        'add_new'            => __( 'Add New', 'ct_theme' ),
        'add_new_item'       => __( 'Add New Testimonial', 'ct_theme' ),
        'new_item'           => __( 'New Testimonial', 'ct_theme' ),
        'edit_item'          => __( 'Edit Testimonial', 'ct_theme' ),
        'view_item'          => __( 'View Testimonial', 'ct_theme' ),
        'all_items'          => __( 'Testimonials', 'ct_theme' ),
        'search_items'       => __( 'Search Testimonials', 'ct_theme' ),
        'parent_item_colon'  => '',
        'not_found'          => __( 'No Testimonials found', 'ct_theme' ),
        'not_found_in_trash' => __( 'No Testimonials found in Trash', 'ct_theme' ),
    ),
    'singular_label'      => __( 'testimonial', 'ct_theme' ),
    'public'              => true,
    'publicly_queryable'  => true,
    'exclude_from_search' => false,
    'show_ui'             => true,
    'show_in_menu'        => true,
    'menu_position'       => 4,
    'menu_icon'           => 'dashicons-format-quote',
    'capability_type'     => 'post',
    'hierarchical'        => false,
    'supports'            => array( 'title', 'editor', 'thumbnail' ),
    'has_archive'         => false,
    'rewrite'             => array( 'slug' => 'testimonials' ),
    'query_var'           => false,
    'can_export'          => true,
    'show_in_nav_menus'   => true,
    'taxonomies'          => array( 'post_tag', 'category' )
) );

//Register taxonomy

register_taxonomy( 'testimonial_category', array('testimonials'), array('labels' => array(
    'name' => _x('Testimonial Categories', 'taxonomy general name', 'ct_theme'),
    'singular_name' => _x('Testimonial Category', 'taxonomy singular name', 'ct_theme'),
    'search_items' => __('Search Categories', 'ct_theme'),
    'popular_items' => __('Popular Categories', 'ct_theme'),
    'all_items' => __('All Categories', 'ct_theme'),
    'parent_item' => null,
    'parent_item_colon' => null,
    'edit_item' => __('Edit Testimonial Category', 'ct_theme'),
```

```
    'update_item' => __('Update Testimonial Category', 'ct_theme'),
    'add_new_item' => __('Add New Testimonial Category', 'ct_theme'),
    'new_item_name' => __('New Testimonial Category Name', 'ct_theme'),
    'separate_items_with_commas' => __('Separate Testimonial category with commas',
↪'ct_theme'),
    'add_or_remove_items' => __('Add or remove Testimonial category', 'ct_theme'),
    'choose_from_most_used' => __('Choose from the most used Testimonial category',
↪'ct_theme'),
    'menu_name' => __('Categories', 'ct_theme')
),
    'hierarchical' => true,
    'public' => false,
    'show_in_nav_menus' => false,
    'show_ui' => true,
    'show_tagcloud' => false,
    'query_var' => 'testimonial_category',
    'rewrite' => false) );
```

6.5.5 Footer Extension

CT Footer extension allows for quick and easy sidebars configuration, registration and styling. When activated it replaces the default theme footer and sidebars.

Important: CT Footer extension needs to be activated in wp-admin Unyson settings menu in order to work.

Typical setup

In theme boilerplate inside footer.php, there is a function call `fw_ct_bee_render_footer()` which renders either:

1. Default footer view from theme page template *default-footer.php* when CT Footer is not active or
2. Footer rendered by CT Footer extension according to configuration located in theme plugin at: *extensions/ct-footer/config.php*

Important: Both of those footers needs to be properly styled, as theme reviewers will check the theme with and without theme plugin activated.

Configuration

You can configure footer structure and classes inside *extensions/ct-footer/config.php* file. Below there is an example configuration with instructions in comments

```
<?php

/** Modify below settings to generate sidebars */

/** Set classes to each sidebar wrapper depending on number of columns */
$sidebar_wrappers = array(

    /** only one sidebar */
```

```
    // footer row 1, column 1
    'sidebars_1_footer_1_1'      => 'col-sm-12 col-md-12 col-lg-9 ct-footer-column ct-
↪footer-column--oneColumn ct-u-padding-both-25',

    // footer row 2, column 1
    'sidebars_1_footer_2_1'      => 'col-sm-12 col-md-12 col-lg-9 ct-footer-column ct-
↪footer-column--oneColumn ct-u-padding-both-25',

    // postfooter row 1, column 1
    'sidebars_1_post-footer_1_1' => 'col-sm-12 col-md-12 col-lg-12',

    // postfooter row 2, column 1
    'sidebars_1_post-footer_2_1' => 'col-sm-12 col-md-12 col-lg-12',


    /** two sidebars */

    // footer row 1, column 1 and 2
    'sidebars_2_footer_1_1' => 'col-sm-6 col-md-6 col-lg-offset-1 col-lg-4 ct-footer-
↪column ct-footer-column--twoColumns ct-u-padding-both-25',
    'sidebars_2_footer_1_2' => 'col-sm-6 col-md-6 col-lg-4 ct-footer-column ct-footer-
↪column--twoColumns ct-u-padding-both-25',

    // footer row 2, column 1 and 2
    'sidebars_2_footer_2_1' => 'col-sm-6 col-md-6 col-lg-offset-1 col-lg-4 ct-footer-
↪column ct-footer-column--twoColumns ct-u-padding-both-25',
    'sidebars_2_footer_2_2' => 'col-sm-6 col-md-6 col-lg-4 ct-footer-column ct-footer-
↪column--twoColumns ct-u-padding-both-25',

    // post footer row 1, column 1 and 2
    'sidebars_2_post-footer_1_1' => 'col-sm-6 col-md-6 col-lg-6 ct-postfooter-column␣
↪ct-postfooter-column--left',
    'sidebars_2_post-footer_1_2' => 'col-sm-6 col-md-6 col-lg-6 ct-postfooter-column␣
↪ct-postfooter-column--right',

    // post footer row 2, column 1 and 2
    'sidebars_2_post-footer_2_1' => 'col-sm-6 col-md-6 col-lg-6 ct-postfooter-column␣
↪ct-postfooter-column--left',
    'sidebars_2_post-footer_2_2' => 'col-sm-6 col-md-6 col-lg-6 ct-postfooter-column␣
↪ct-postfooter-column--right',

    /** three sidebars */

    'sidebars_3_footer_1_1' => 'col-sm-6 col-md-6 col-lg-3 ct-footer-column ct-u-
↪padding-both-25',
    'sidebars_3_footer_1_2' => 'col-sm-6 col-md-6 col-lg-3 ct-footer-column ct-u-
↪padding-both-25',
    'sidebars_3_footer_1_3' => 'col-sm-12 col-md-12 text-center-md col-lg-3 ct-footer-
↪column ct-u-padding-both-25',

    'sidebars_3_footer_2_1' => 'col-sm-6 col-md-6 col-lg-3 ct-footer-column ct-u-
↪padding-both-25',
    'sidebars_3_footer_2_2' => 'col-sm-6 col-md-6 col-lg-3 ct-footer-column ct-u-
↪padding-both-25',
    'sidebars_3_footer_2_3' => 'col-sm-12 col-md-12 text-center-md col-lg-3 ct-footer-
↪column ct-u-padding-both-25',
```

```
    /** post footer has only 2 columns, so leave below empty or delete it */
    'sidebars_3_post-footer_1_1' => '',
    'sidebars_3_post-footer_1_2' => '',
    'sidebars_3_post-footer_1_3' => '',

    'sidebars_3_post-footer_2_1' => '',
    'sidebars_3_post-footer_2_2' => '',
    'sidebars_3_post-footer_2_3' => '',

    /** four sidebars */

    'sidebars_4_footer_1_1' => 'col-sm-6 col-md-6 col-lg-offset-1 col-lg-4 ct-footer-
↪column ct-footer-column--twoColumns ct-u-padding-both-25',
    'sidebars_4_footer_1_2' => 'col-sm-6 col-md-6 col-lg-4 ct-footer-column ct-footer-
↪column--twoColumns ct-u-padding-both-25',
    'sidebars_4_footer_1_3' => 'col-sm-6 col-md-6 col-lg-4 ct-footer-column ct-footer-
↪column--twoColumns ct-u-padding-both-25',
    'sidebars_4_footer_1_4' => 'col-sm-6 col-md-6 col-lg-4 ct-footer-column ct-footer-
↪column--twoColumns ct-u-padding-both-25',

    'sidebars_4_footer_2_1' => 'col-sm-6 col-md-6 col-lg-offset-1 col-lg-4 ct-footer-
↪column ct-footer-column--twoColumns ct-u-padding-both-25',
    'sidebars_4_footer_2_2' => 'col-sm-6 col-md-6 col-lg-4 ct-footer-column ct-footer-
↪column--twoColumns ct-u-padding-both-25',
    'sidebars_4_footer_2_3' => 'col-sm-6 col-md-6 col-lg-4 ct-footer-column ct-footer-
↪column--twoColumns ct-u-padding-both-25',
    'sidebars_4_footer_2_4' => 'col-sm-6 col-md-6 col-lg-4 ct-footer-column ct-footer-
↪column--twoColumns ct-u-padding-both-25',

    /** post footer has only 2 columns, so leave below empty or delete it */
    'sidebars_4_post-footer_1_1' => '',
    'sidebars_4_post-footer_1_2' => '',
    'sidebars_4_post-footer_1_3' => '',
    'sidebars_4_post-footer_1_4' => '',

    'sidebars_4_post-footer_2_1' => '',
    'sidebars_4_post-footer_2_2' => '',
    'sidebars_4_post-footer_2_3' => '',
    'sidebars_4_post-footer_2_4' => '',


);

$cfg = array(

    /** How many rows with sidebars should the footer consist of? */
    'footer_elements'  => array(

        /** sidebar_id => properties */
        'footer_1'      => array(

            // name of sidebars in this row
            'name'     => 'Upper footer',

            // number of sidebars in this row
            'sidebars' => 4,

            // HTML before sidebar
```

```
            'before'   => '<div class="ct-footer-line"></div><div class="ct-footer ct-
↪u-padding-both-35"><div class="container"><div class="row">',

            // HTML after sidebar
            'after'    => '</div></div></div></div>',

        ),

        'footer_2'      => array(
            'name'     => 'Lower footer',
            'sidebars' => 4,
            'before'   => '<div class="ct-footer-line"></div><div class="ct-footer ct-
↪u-padding-both-35"><div class="container"><div class="row">',
            'after'    => '</div></div></div></div>',
        ),

        'post-footer_1' => array(
            'name'     => 'Upper post footer',
            'sidebars' => 2,
            'before'   => '<div class="ct-postfooter ct-u-padding-both-25"><div class=
↪"container"><div class="row">',
            'after'    => '</div></div></div>',
        ),

        'post-footer_2' => array(
            'name'     => 'Lower post footer',
            'sidebars' => 2,
            'before'   => '<div class="ct-postfooter ct-u-padding-both-25"><div class=
↪"container"><div class="row">',
            'after'    => '</div></div></div>',
        ),

    ),

    /** Assign sidebar classes from above, do not change */
    'sidebar_wrappers' => $sidebar_wrappers,

);
```

Hooks

There are hooks which allow you to overwrite default HTML rendered by the extension, skip selected sidebars or change their properties. These are:

Footer output (everything) hooks:

```
apply_filters( 'fw_ext_ct_footer_before_footer', '<footer class="ct-footer">' );
apply_filters( 'fw_ext_ct_footer_after_footer', '</footer>' );
apply_filters( 'fw_ext_ct_footer', $output, $cfg );
```

Footer elements (rows) hooks:

```
apply_filters( 'fw_ext_ct_footer_before_footer_element', $wrapper, $footer_element );
apply_filters( 'fw_ext_ct_footer_after_footer_element', $wrapper, $footer_element );
```

Sidebar hooks:

```
apply_filters( 'fw_ext_ct_footer_register_sidebar_skip', false, $element, $i,
↪$properties );
apply_filters( 'fw_ext_ct_footer_before_sidebar', $wrapper, $sidebar_id, $footer_
↪element, $column );
apply_filter( 'fw_ext_ct_footer_render_sidebar_output', $output, $footer_element,
↪$column, $index, $return );
apply_filter( 'fw_ext_ct_footer_render_sidebar_return', $return, $footer_element,
↪$column, $index, $output );
apply_filters( 'fw_ext_ct_footer_after_sidebar', $wrapper, $sidebar_id, $footer_
↪element, $column );
```

Sidebar properties hooks:

```
apply_filters( 'fw_ext_ct_footer_register_sidebar_name', $name, $element, $properties␣
↪);
apply_filters( 'fw_ext_ct_footer_register_sidebar_id', $id, $element, $properties );
apply_filters( 'fw_ext_ct_footer_register_sidebar_description', $description,
↪$element, $properties );
apply_filters( 'fw_ext_ct_footer_register_sidebar_before_widget', $before_widget,
↪$element, $properties );
apply_filters( 'fw_ext_ct_footer_register_sidebar_after_widget', $after_widget,
↪$element, $properties );
apply_filters( 'fw_ext_ct_footer_register_sidebar_before_title', $before_title,
↪$element, $properties );
apply_filters( 'fw_ext_ct_footer_register_sidebar_after_title', $after_title,
↪$element, $properties );
```

6.5.6 Theme Demo Plugin

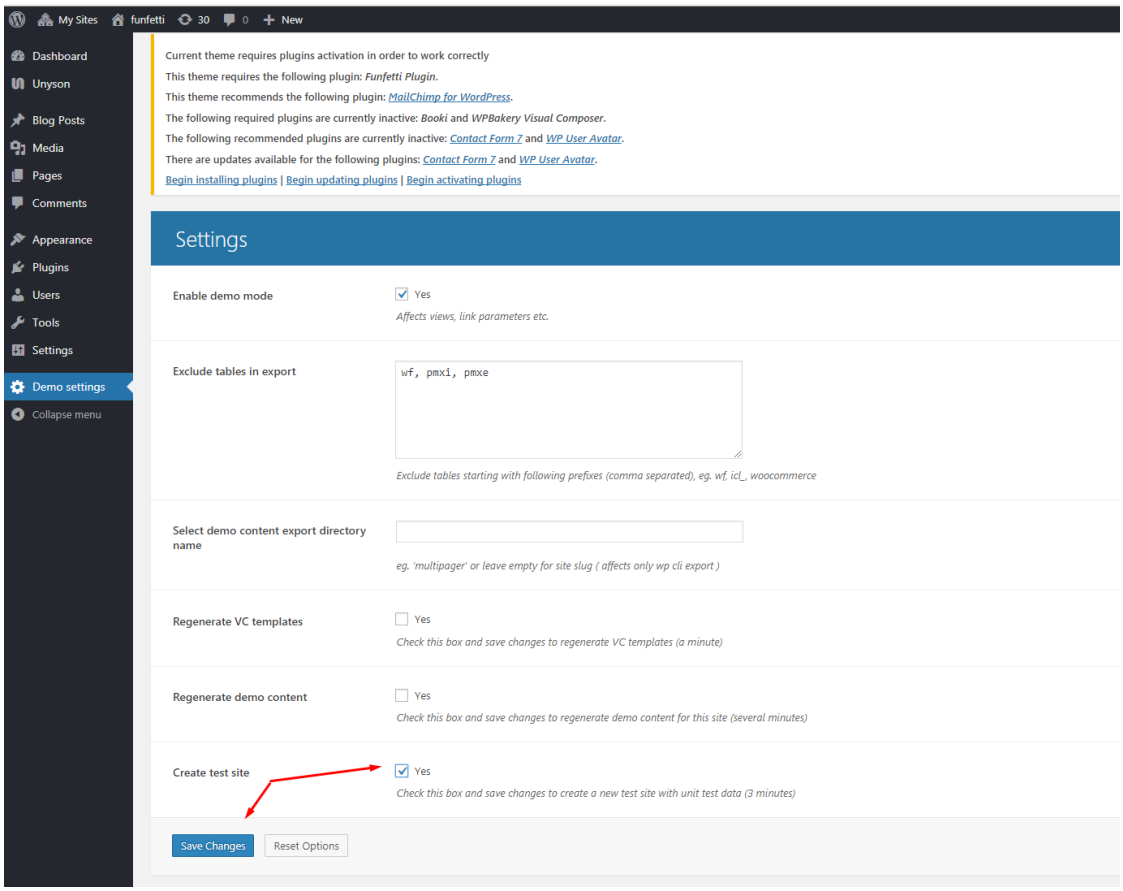
Theme Demo Plugin extensions allow you to:

- Test shortcodes and VC templates
- Use theme hooks to alter demo website looks, ie. layout urls
- Use Unyson Backups hooks to auto generate theme demo contents
- Create clean test site on demo page for unit testing
- Provide build functions to recompile SCSS. Read more: Build

Note: While Theme Demo Plugin is essential for demo page, you can also activate it on your local installation. Grab it from Theme Demo Plugin boilerplate repository.

Creating test site on the demo page

For QA purpuses and if you already have a demo page for your theme, you can easily create a test site on your demo page. It will have theme-check, unit test data and monster widget already installed.



1. Go to *wp-admin -> Demo Settings* (see screenshot above)
2. Check "Create test site"
3. Press save
4. After it finishes, you can click on "My Sites" and open new "bee automated tests" site
5. The test site will be updated every time your demo page is updated.

Currently, you can test the site using preinstalled tools:

- theme-check,
- unit test data,
- monster widget

6.6 Sample Extensions

You can find plenty of extensions and shortcodes written already in previous projects. Just grab the samples plugin from Bumblebee Samples, test and copy the extensions you need to your project.

Following, there are descriptions of individual shortcodes and extensions.

6.6.1 Sample Shortcodes

Button

Icon Box

Styled google map

Icon

Image Carousel

Image Gallery

6.6.2 Testimonials Extension

6.6.3 FAQ Extension

6.6.4 Portfolio Extension

6.6.5 Team Extension

6.6.6 Twitter Extensions

6.6.7 Flickr Extension

6.7 Extensions

6.7.1 Introduction

Extensions are functionalities that add something new to framework or to another extension. They can be installed via Extensions page, bundled with a theme or loaded from a plugin (or any directory).

Directory Structure

Every extension has everything it needs in its own folder: settings, options, scripts, styles, etc. In this way, extensions can be easy added or removed without affecting other files.

The extension directory has the following structure:

```
{extension-name}/
├─manifest.php  # Data about extension: version, name, dependencies, etc.
├─class-fw-extension-{extension-name}.php # class FW_Extension_{Extension_Name}␣
↪extends FW_Extension { ... }
├─config.php    # Extension specific configurations
├─static.php    # wp_enqueue_style() and wp_enqueue_script()
├─posts.php     # register_post_type() and register_taxonomy()
├─hooks.php     # add_filter() and add_action()
├─helpers.php   # Helper functions and classes
├─readme.md.php # Install instructions
├─shortcodes/
   ├─example-shortcodes
     ├─config.php
     ├─options.php
     ├─static.php
     ├─view/
       ├─view.php
├─options/
  ├─posts/      # Post types options
    ├─post.php
    ├─{post-type}.php
    └─...
  ├─taxonomies/ # Taxonomies terms options
    ├─category.php
    ├─post_tag.php
    ├─{taxonomy}.php
    └─...
  └─...
├─settings-options.php # Extension Settings page options
├─views/
  └─...
├─static/
  ├─js/
  ├─css/
  └─...
├─includes/ # All .php files are auto included (no need to require_once)
  ├─other.php
  └─...
└──[extensions/] # Directory for sub extensions
```

Let's take a closer look at each directory and file, and understand how it works.

- `manifest.php` - The only required file, all other files are optional. It contains the base information about extension. More details about the extension manifest.
- `class-fw-extension-{extension-name}.php` - If the extension has some advanced functionality, it can define a class that will be the instance of the extension returned by `fw()->extensions->get('{extension-name}')`. By default an instance of default class will be created, which is an empty class that just extends the `FW_Extension` class. This file can't be overwritten.
- `config.php` - Configuration array, which is accessible through the `$ext->get_config('key')` method. Users can customize it by creating the same file in `{theme-name}/framework-customizations/extension/{extension-name}/config.php` and overwrite only some keys (internally is made `array_merge($extension_config, $theme_customized_config)`).
- `static.php` - Enqueue extension scripts and styles. It is included automatically on the `wp_enqueue_scripts` and `admin_enqueue_scripts` actions, so you can enqueue both admin and frontend scripts and styles from it, but you will have to use the `is_admin()` function. This file can

be overwritten from theme by creating `{theme-name}/framework-customizations/extension/{extension-name}/static.php`.

- `posts.php` - Register theme post types and taxonomies in this file. It is included automatically on the `init` action.
- `hooks.php` - File containing filters and actions. This file is automatically included as early as possible, in this way your extension will not miss any action or filter execution.
- `helpers.php` - All extension's helper functions and classes must be in this file.
- `readme.md.php` - Install instructions for users, to make the extension start working.
- `options/` - A directory containing option files: post types, taxonomies or custom options. The framework will **not** automatically pick them (like theme options), only the extension decides how to use these options. You can access them through the `$ext->get_[...]_options()` methods. Users can overwrite in the theme any file from the `options/` directory, by creating `{theme-name}/framework-customizations/extension/{extension-name}/options/{file-name}.php`.
- `settings-options.php` - Options used for the extension settings page. The framework picks them automatically and saves the values in then database. Use the `fw_get_db_ext_settings_option()` function to get options values from the database. This file can't be overwritten from the theme, that's why it wasn't placed in the `options/` directory.
- `views/` - Contains extension templates. Inside extension class you can render a view like this `$this->render_view('template-name');`. The views can be overwritten in the theme by creating `{theme-name}/framework-customizations/extension/{extension-name}/views/{template-name}.php`.
- `static/` - Contains styles, scripts, images and other static files. Some files can be overwritten in the theme, some not, it depends how they are enqueued in the extension, using `$this->locate_URI()` or `$this->get_declared_URI()`.
- `includes/` - All `.php` files within this directory will be included automatically.

Load Locations

Extensions are loaded from the following directories and in the following order:

1. `framework/extensions/`
2. Custom directories specified via the `fw_extensions_locations` filter
3. `{parent-theme}/framework-customizations/extensions/`
4. `{child-theme}/framework-customizations/extensions/`

Custom Load Locations

You can load extensions from any directory via the `fw_extensions_locations` filter. For e.g. to load extensions from your own plugin:

```
/**
 * @internal
 */
function _filter_plugin_awesome_extensions($locations) {
    $locations[ dirname(__FILE__) . '/extensions' ]
    =
    plugin_dir_url( __FILE__ ) . 'extensions';
```

```
    return $locations;
}
add_filter('fw_extensions_locations', '_filter_plugin_awesome_extensions');
```

Child Extensions

Child Extensions are used to split a big extension into sub-extensions to separate the functionalities or when some extensions are tightly connected to the parent extension and can't exist without it, so they will be loaded only if the parent extension exists, is loaded and activated.

A child extension can be located in any *load location* but must be on the same relative path. Here are some examples where an extension can exists and where its child extensions can be placed:

1. If the `hello` extension is located in framework, the child extensions can be placed in: framework, parent theme and child theme.

```
framework/
└─extensions/
  └─hello/
    └─extensions/
      ├─hello-child/
      └─...

├─parent-theme/
│ └─framework-customizations/
│   └─extensions/
│     └─hello/
│       └─extensions/
│         ├─hello-child/
│         └─...
└─child-theme/
  └─framework-customizations/
    └─extensions/
      └─hello/
        └─extensions/
          ├─hello-child/
          └─...
```

2. If the `hello` extension is located in parent theme, the child extensions can be placed in: parent theme and child theme.

```
├─parent-theme/
│ └─framework-customizations/
│   └─extensions/
│     └─hello/
│       └─extensions/
│         ├─hello-child/
│         └─...
└─child-theme/
  └─framework-customizations/
    └─extensions/
      └─hello/
        └─extensions/
          ├─hello-child/
          └─...
```

3. If the `hello` extension is located in child theme, the child extensions can be placed only in the child theme.

```
└child-theme/
  └framework-customizations/
    └extensions/
      └hello/
        └extensions/
          ├hello-child/
          └...
```

6.7.2 Create Extension

To create an extension:

1. Create a directory with the name of the extension in any `extensions/` directory with a manifest.php file in it.

   Internally that will create an instance of `FW_Extension_Default` class. Optionally, you can place a file `class-fw-extension-{extension-name}.php` with the following contents, in the newly created directory and start create some advanced functionality:

   ```php
   <?php if (!defined('FW')) die('Forbidden');

   class FW_Extension_{Extension_Name} extends FW_Extension
   {
       // ...
   }
   ```

2. To make the extension visible on the Extensions list page *(by default it is hidden)* set the manifest `display` parameter to `true`.
3. Make sure you understand what the manifest `standalone` parameter means.

6.7.3 Cookbook

The Cookbook is a collection of specific recipes that explain how to correctly solve the most recurrent problems that developers face in their day to day work.

Disable Child Extensions

Child extensions will not be activated if parent extension will `return false;` from `_init()`.

```php
<?php if (!defined('FW')) die('Forbidden');

class FW_Extension_Example extends FW_Extension
{
    /**
     * @internal
     */
    protected function _init()
    {
```

```
        // ...

        if ($this->something_is_wrong()) {
            return false; // prevent child extensions activation
        }
    }
}
```

Validate Child Extensions

The parent extension has the possibility to check each child extension if it's valid or not. If the child extension is not valid, it will not be activated. To do that, the parent extension must overwrite the `_child_extension_is_valid()` method.

The method should return `true` if child extension is valid, and `false` if not.

```
<?php if (!defined('FW')) die('Forbidden');

class FW_Extension_Example extends FW_Extension
{
    /**
     * {@inheritdoc}
     */
    public function _child_extension_is_valid($child_extension_instance)
    {
        // force child extensions to extend some custom class, instead of FW_Extension
        return is_subclass_of($child_extension_instance, 'FW_Ext_Demo_Custom_Class');
    }

    // ...
}
```

6.8 Components

6.8.1 Introduction

The Unyson framework core has three components:

- *Theme*
- *Backend*
- *Extensions*

Accessing one of the core's component is done in this way:

```
fw()->{$component}->{$method}()
```

`fw()` returns the framework object, this being the only way to access the framework core.

6.8.2 Theme

The Theme component makes the connection between the theme and the framework. The working directory is `framework-customizations/theme/` within child and parent themes.

- `get_options($name)` - return options array from specified option file `framework-customizations/theme/options/{$name}.php`.

```
$custom_options = fw()->theme->get_options('custom');
```

- `get_settings_options()` - return options array from `framework-customizations/theme/options/settings.php`.

```
$settings_options = fw()->theme->get_settings_options();
```

- `get_customizer_options()` - return options array from `framework-customizations/theme/options/customizer.php`.

```
$customizer_options = fw()->theme->get_customizer_options();
```

- `get_post_options($post_type)` - return options array from `framework-customizations/theme/options/posts/{$post_type}.php`.

```
$custom_post_options = fw()->theme->get_post_options('custom_post');
```

- `get_taxonomy_options($taxonomy)` - return options array from `framework-customizations/theme/options/taxonomies/{$post_type}.php`.

```
$category_options = fw()->theme->get_taxonomy_options('category');
```

- `get_config($key = null)` - return entire config array from `framework-customizations/theme/config.php` or only specified key.

```
$backlisted_extensions = fw()->theme->get_config('extensions_blacklist');
```

- `locate_path($rel_path)` - search full path of the file by a given relative path. Will search in the **child theme** then in the **parent theme**.

```
echo fw()->theme->locate_path('/custom.php');

// prints '/.../wp-content/themes/scratch-theme/framework-customizations/
↪theme/custom.php'
```

6.8.3 Backend

Admin side functionality:

- `option_type($type)` - get instance of a registered option type.

```
$option_type_text = fw()->backend->option_type('text');

echo $option_type_text->render('demo', array( 'value' => 'Demo Value' ));
```

- `render_option($id, $option, $data = array(), $design = 'default')` - render option html together with `label`, `desc` and `help`.

> **Attention:** Does not accept container options.

```
// simple usage
echo fw()->backend->render_option('demo', array( 'type' => 'text' ));

// advanced usage
echo fw()->backend->render_option(
    'demo',
    array(
        'type'  => 'text',
        'label' => __('Demo Label', '{domain}'),
        'desc'  => __('Demo Description', '{domain}'),
        'html'  => __('Demo Help Tip', '{domain}'),
        'value' => 'default value',
    ),
    array(
        'id_prefix'   => 'custom-id-prefix-',
        'name_prefix' => 'custom_name_prefix',
        'value'       => 'overwrite default value'
    ),
    'taxonomy'
);
```

- `render_options(&$options, &$values = array(), $options_data = array(), $design = 'default')` - generate html from any array of options.

```
$options = array(
    'option-1' => array( 'type' => 'text',     'value' => 'default value 1
↪' ),
    'option-2' => array( 'type' => 'textarea', 'value' => 'default value 2
↪' ),
);

// simple usage
echo fw()->backend->render_options($options);

$values = array(
    'option-1' => 'Some value', // this overwrites default value
    // 'option-2' value is not specified, so it will have default value
);

// advanced usage
echo fw()->backend->render_options(
    $options,
    $values,
    array(
        'id_prefix'   => 'custom-id-prefix-',
        'name_prefix' => 'custom_name_prefix',
        'value'       => 'overwrite default value'
    ),
    'taxonomy'
);
```

- `render_box($id, $title, $content, $other = array())` - render WordPres metabox.

```
// simple usage
echo fw()->backend->render_box('some-html-id', 'Title', 'Some <strong>
↪Content</strong>');

// advanced usage
```

```
echo fw()->backend->render_box(
    'some-html-id',
    'Title',
    'Some <strong>Content</strong>',
    array(
        'html_before_title' => '&lt;',
        'html_after_title'  => '&gt;',
        'attr' => array(
            'class' => 'custom-class'
        ),
    )
);
```

- `enqueue_options_static($options)` - enqueue options scripts and styles

```
$options = array(
    'option-1' => array( 'type' => 'text',     'value' => 'default value 1
↪' ),
    'option-2' => array( 'type' => 'textarea', 'value' => 'default value 2
↪' ),
);

fw()->backend->enqueue_options_static($options);
```

6.8.4 Extensions

The core of Extensions.

- `get($extension_name)` - get instance of an existing active extension.

```
echo fw()->extensions->get('extension_name')->get_name();
```

Also it can be used to check if an extension exists (is active).

```
if (fw()->extensions->get('extension_name')) {
    fw()->extensions->get('extension_name')->some_method();
}

// or there is shorter alias for this method

if (fw_ext('extension_name')) {
    fw_ext('extension_name')->some_method();
}
```

6.9 Helpers

6.9.1 Introduction

Helpers are classes and functions with useful functionality. Here are built-in helpers that you can use:

- *PHP Helpers*
- *JavaScript Helpers*
- *CSS Helpers*

6.9.2 PHP Helpers

General

General PHP helpers:

- `fw_print($value, $die = false)` - styled version of `print_r()`.
- `fw_html_tag($tag, $attr = null, $end = null)` - generate html tag.

```
echo fw_html_tag('script', array('src' => '/demo.js'), true);

// <script src="/demo.js"></script>
```

- `fw_attr_to_html(array $attr_array)` - generate html attributes from array.

```
echo '<div '. fw_attr_to_html(array('id' => 'foo', 'class' => 'bar')) .'>
↪</div>';

// <div id="foo" class="bar" ></div>
```

- `fw_akg($keys, &$array_or_object, $default_value = null, $keys_delimiter = '/')` - get array multikey value.

Note: **MultiKey** is a string composed from multiple array keys, separated by a delimiter character, that represents an array structure. For example

```
'a/b/c'
```

represents

```
array(
    'a' => array(
        'b' => array(
            'c' => null
        )
    )
)
```

```php
$demo = array(
    'a' => array(
        'b' => 'hello'
    )
 );

echo fw_akg('a/b', $demo);

// 'hello'
```

- `fw_aks($keys, $value, &$array_or_object, $keys_delimiter = '/')` - set a *multikey* value in array.

```php
$demo = array(
    'a' => array()
);

fw_aks('a/b', 'hello', $demo);

print_r($demo);

/*
array(
    'a' => array(
        'b' => 'hello'
    )
)
*/
```

- `fw_aku($keys, &$array_or_object, $keys_delimiter = '/')` - unset a *multikey* from array.

```php
$demo = array(
    'a' => array(
        'b' => array()
    )
);

fw_aku('a/b', $demo);

print_r($demo);

/*
array(
    'a' => array()
)
*/
```

- `fw_rand_md5()` - generate a random md5.
- `fw_unique_increment()` - random number incremented every time you call the function.

```php
echo fw_unique_increment(), PHP_EOL;
echo fw_unique_increment(), PHP_EOL;
echo fw_unique_increment(), PHP_EOL;

/*
9370
```

```
9371
9372
*/
```

- `fw_stripslashes_deep_keys($value)` - strip slashes (recursive) from values and keys (if value is array) if `magic_quotes_gpc = On`.
- `fw_addslashes_deep_keys($value)` - add slashes (recursive) to values and keys (if value is array) if `magic_quotes_gpc = On`.
- `fw_current_screen_match($rules)` - check if current `global $current_screen;` (available in admin side) matches the given rules. Used to detect on which admin page you currently are. Thus you can for example enqueue a script only on a target page, not on all admin pages.

```
/**
 * @internal
 */
function _action_enqueue_demo_admin_scripts() {
    // To find out what is the current screen of the current page,␣
↪uncomment next line
    //global $current_screen; fw_print($current_screen);

    $only = array(
        'only' => array(
            array( 'id' => 'dashboard' )
        )
    );

    if (fw_current_screen_match($only)) {
        // enqueue this script only on dashboard page
        wp_enqueue_script(
            'demo-dashboard',
            get_template_directory_uri() .'/js/demo-only.js'
        );
    }

    $exclude = array(
        'exclude' => array(
            array( 'id' => 'dashboard' ),
            array( 'post_type' => 'post' )
        )
    );

    if (fw_current_screen_match($exclude)) {
        // enqueue this script on all admin pages
        // except dashboard page and all pages from posts menu (add, edit,
↪ categories, tags)
        wp_enqueue_script(
            'demo-dashboard',
            get_template_directory_uri() .'/js/demo-excluded.js'
        );
    }
}
add_action('admin_enqueue_scripts', '_action_enqueue_demo_admin_scripts');
```

Note: You can combine `only` and `exclude` in the same rules array.

- `fw_locate_theme_path_uri($rel_path)` - search by relative path, in child then in parent theme directory, and return URI.

```
echo fw_locate_theme_path_uri('/styles.css');

// http://your-site.com/wp-content/themes/child-theme/style.css
```

- `fw_locate_theme_path($rel_path)` - search by relative path, in child then in parent theme directory, and return full path.

```
echo fw_locate_theme_path('/styles.css');

// /var/www/wordpress/public_html/wp-content/themes/child-theme/style.css
```

- `fw_render_view($file_path, $view_variables = array())` - safe render view and return html. In view will be accessible only passed variables, not current context variables.

```
$private = 'Top Secret';

echo fw_render_view(
    get_stylesheet_directory() .'/demo-view.php',
    array('message' => 'Hello')
);

/* demo-view.php
<?php if (!defined('FW')) die('Forbidden');

echo $message;
echo $private;
*/

// Hello
// Notice: Undefined variable: private
```

- `fw_get_variables_from_file($file_path, array $variables)` - extract specified variables from file.

```
$variables = fw_get_variables_from_file(
    get_stylesheet_directory() .'/demo-variables.php',
    array(
        'message' => 'Hi',
        'foo' => 'bar'
    )
);

/* demo-variables.php
<?php if (!defined('FW')) die('Forbidden');

$message = 'Hello';
*/

print_r($variables);

/*
array(
    'message' => 'Hello',
    'foo' => 'bar'
)
```

```
*/
```

- `fw_include_file_isolated($file_path)` - include files isolated and don't give access to current context variables.

```
$private = 'Top Secret';

fw_include_file_isolated(get_stylesheet_directory() .'/demo-isolated.php
↪');

/* demo-isolated.php
<?php if (!defined('FW')) die('Forbidden');

echo $private;
*/

// Notice: Undefined variable: private
```

- `fw_html_attr_name_to_array_multi_key($attr_name)` - convert html `name` attribute to *multi-key*.

```
echo fw_html_attr_name_to_array_multi_key('a[b][c]');

// 'a/b/c'
```

- `fw_is_real_post_save()` - used in 'save_post' action to detect if it's a real post save, not a revision, auto save or something else.
- `fw_current_url()` - generate current page url from `$_SERVER` data.
- `fw_is_valid_domain_name($domain_name)` - check if a domain name is valid.
- `fw_htmlspecialchars($string)` - UTF-8 version of php's `htmlspecialchars()`. Just a shorthand not to write two more parameters for default `htmlspecialchars()` every time.

> **Note:** In php 5.2 `htmlspecialchars()` default encoding is not UTF-8.

- `fw_human_time($seconds)` - convert seconds to human readable time.

```
echo fw_human_time(12345);

// '3 hours'
```

- `fw_strlen($string)` - UTF-8 version of php's `strlen()`.

```
echo strlen('!'), PHP_EOL;
echo fw_strlen('!'), PHP_EOL;

// 13
// 7
```

- `fw_is_post_edit()` - check if you are currently on a post edit page. It also detects if form submit was made from the post edit page.
- `fw_dirname_to_classname($dirname)` - convert directory name to string to be used as/in class name.

```
echo 'FW_'. fw_dirname_to_classname('hello-world');

// FW_Hello_World
```

- `fw_fix_path($path)` - make sure a path is in unix style, with / directory separators.
- `fw_get_stylesheet_customizations_directory()` - Full path to the child-theme/framework-customizations directory.
- `fw_get_stylesheet_customizations_directory_uri()` - URI to the child-theme/framework-customizations directory.
- `fw_get_template_customizations_directory()` - Full path to the parent-theme/framework-customizations directory.
- `fw_get_template_customizations_directory_uri()` - URI to the parent-theme/framework-customizations directory.
- `fw_get_framework_directory()` - Full path to the parent-theme/framework directory.
- `fw_get_framework_directory_uri()` - URI to the parent-theme/framework directory

Cache

Use cache to store frequently accessed data. Cache is just a big array and has one useful feature: it will automatically begin to unset array keys if the php memory is close to full. So it is safe to store in it as much data as you want (of course the maximum allowed by php, by default is ~100Mb).

```
function get_foo_bar() {
    $cache_key = 'foo/bar';

    try {
        /**
         * This will throw an exception if the key was not found
         */
        return FW_Cache::get($cache_key);
    } catch (FW_Cache_Not_Found_Exception $e) {
        $data = _generate_foo_bar_data();

        FW_Cache::set($cache_key, $data);

        return $data;
    }
}
```

Attention: Don't do this:

```
    ...
} catch (FW_Cache_Not_Found_Exception $e) {
    FW_Cache::set($cache_key, _generate_foo_bar_data());

    return FW_Cache::get($cache_key);
}
```

because `FW_Cache::set(...)` can fail or the data that was set can be removed after automatically memory free.

Options

Functions for working with options:

- `fw_extract_only_options(array $options)` - extract only regular options from any array of options.

```
$options = array(
    array(
        'type' => 'box',
        'options' => array(
            'demo-1' => array(
                'type' => 'text'
            )
        )
    ),
    array(
        'type' => 'box',
        'options' => array(
            array(
                'type' => 'tab',
                'options' => array(
                    'demo-2' => array(
                        'type' => 'textarea'
                    )
                )
            )
        )
    )
);

print_r( fw_extract_only_options($options) );

/*
array(
    'demo-1' => array(
        'type' => 'text'
    ),
    'demo-2' => array(
        'type' => 'textarea'
    )
)
*/
```

- `fw_get_options_values_from_input(array $options, $input_array = null)` - extract options values from input array. If no input array is provided, values from `$_POST` will be used.

```
$options = array(
    'demo-1' => array( 'type' => 'text', 'value' => 'default value 1' ),
    'demo-2' => array( 'type' => 'text', 'value' => 'default value 2' ),
);

$input_values = array(
    'demo-1' => 'input value 1',
    'demo-3' => 'input value 3',
);

$values = fw_get_options_values_from_input($options, $input_values);
```

```
print_r($values);

/*
array(
    'demo-1' => 'input value 1',
    'demo-2' => 'default value 2',
)
*/
```

- `fw_prepare_option_value($value)` - by default WordPress offers filters for other plugins to alter database options and post meta. For ex translation plugins use these filters to translate things. If you save your options values in a custom place (like framework does by default, by saving options in a serialized array in database options and post meta) the WordPress filter doesn't know how to work with them.

  Tip: Use this function to pass an option value through filters and translation features that simulates WordPress default behavior. This function is already used in core so you don't have to bother about passing options values through it each time. Use it if you will do something custom and strings will not be translated.

Database

- `fw_get_db_settings_option($option_id, $default_value = null)` - get value from the database of an option from the theme settings page. Settings options are located in `framework-customizations/theme/options/settings.php`.
- `fw_set_db_settings_option($option_id, $value)` - set a value in the database for an option from the theme settings page.

- `fw_get_db_customizer_option($option_id, $default_value = null)` - get value from the database of an option from the customizer page. Customizer options are located in `framework-customizations/theme/options/customizer.php`.
- `fw_set_db_customizer_option($option_id, $value)` - set a value in the database for an option from the customizer page.

- `fw_get_db_post_option($post_id, $option_id, $default_value = null)` - get a post option value from the database. Post options are located in `framework-customizations/theme/options/posts/{post-type}.php`.
- `fw_set_db_post_option($post_id, $option_id, $value)` - set a post option value in the database.

- `fw_get_db_term_option($term_id, $taxonomy, $option_id, $default_value = null)` - get a term option value from the database. Term options are located in `framework-customizations/theme/options/taxonomies/{taxonomy}.php`.
- `fw_set_db_term_option($term_id, $taxonomy, $option_id, $value)` - set a term option value in the database.

- `fw_get_db_ext_settings_option($extension_name, $option_id, $default_value = null)` - get extension settings option value from the database.
- `fw_set_db_ext_settings_option($extension_name, $option_id, $value)` - update extension settings option value in the database.
- `fw_get_db_extension_data($extension_name, $key, $default_value = null)` - get a value from the database of some private data stored by an extension.
- `fw_set_db_extension_data($extension_name, $key, $value)` - extensions uses this function to store private values in the database.

FW_Form

A convenient way to create forms. You can create a form class instance and give it three callbacks that control the render, validate and save process.

```
$my_form = new FW_Form('<unique-id>', array(
    'render'   => '_my_form_render',
    'validate' => '_my_form_validate',
    'save'     => '_my_form_save',
));

function _my_form_render() {
    $input_value = FW_Request::POST('demo');

    echo '<input type="text" name="demo" maxlength="10" value="'. esc_attr($input_
↪value) .'">';
}

function _my_form_validate($errors) {
    $input_value = FW_Request::POST('demo');

    if (fw_strlen($input_value) > 10) {
        $errors['demo'] = __('Value cannot be more that 10 characters long', '{domain}
↪');
    }

    return $errors;
}

function _my_form_save() {
    $input_value = FW_Request('demo');

    // do something with value
}

echo $my_form->render();
// this will output:
// <form ... ><input type="text" name="demo" maxlength="10" value=""></form>
```

Customize errors

By default the errors are displayed right before the `<form>` tag. You can display the errors in your own way and cancel the default display. Before the errors are displayed, an action is fired so you can use it:

```
/**
 * @param FW_Form $form
 * @internal
 */
function _action_theme_fw_form_errors_display($form) {
    /**
     * Once the errors was accessed/requested
     * the form will cancel/abort the default errors display
     */
    $errors = $form->get_errors();

    echo '<ul class="your-custom-errors-class">';
    foreach ($errors as $input_name => $error_message) {
        echo fw_html_tag(
            'li',
            array('data-input-name' => $input_name),
            $error_message
        );
    }
    echo '</ul>';
}
add_action('fw_form_display_errors_frontend', '_action_theme_fw_form_errors_display');
```

Ajax submit

You can use this script to make `FW_Form` ajax submittable.

Enqueue the script in frontend:

```
// file: {theme}/inc/static.php
// https://github.com/ThemeFuse/Theme-Includes

if (!is_admin()) {
    wp_enqueue_script(
        'fw-form-helpers',
        fw_get_framework_directory_uri('/static/js/fw-form-helpers.js')
    );
    wp_localize_script('fw-form-helpers', 'fwAjaxUrl', admin_url( 'admin-ajax.php',
↪'relative' ));
}
```

Run the initialization script:

```
jQuery(function(){
    fwForm.initAjaxSubmit({
        //selector: 'form[some-custom-attribute].or-some-class'

        // Open the script code and check the `opts` variable
        // to see all options that you can overwrite/customize.
    });
});
```

Create Settings Page

If you want to create a settings page similar to Theme Settings, this will help you getting started.

FW_Flash_Messages

You can display small messages that will be stored on the user's session for exactly one additional request. This is useful when processing a form: you want to redirect and have a special message shown on the next page. These types of messages are called "flash" messages.

Adding a flash message

```
FW_Flash_Messages::add(
    'unique-id',
    __('Test message', '{domain}'),
    'success' // available types: info, warning, error, success
);
```

Displaying flash messages

In admin the messages are displayed as admin notices.

In frontend the messages are printed in footer, then a javascript tries to find on the page the content container and move the messages in it. This position guessing sometimes fails when the page has some special html structure and the messages may not be visible or will be displayed in an inappropriate place. You can choose a place in your template and display the messages manually:

```
<?php if (defined('FW')) { FW_Flash_Messages::_print_frontend(); } ?>
```

6.9.3 JavaScript Helpers

Useful javascript functions and classes. The main helper is `fw`, an object containing constants, methods and classes. To use these helpers, add `fw` to your script dependencies:

```
wp_register_script(..., ..., array('fw'));
```


General

General javaScript helpers:

- `fw.FW_URI` - URI to the framework directory.
- `fw.SITE_URI` - URI to the site root directory.
- `fw.intval(value)` - alternative to php intval(). Returns `0` on failure, instead of `NaN` like parseInt() does.
- `fw.md5(string)` - calculate md5 hash of the string.
- `fw.loading` - show loading on the page.

Tip: Useful when doing AJAX requests and want to inform your users about that.

```
fw.loading.show();

setTimeout(function(){
    fw.loading.hide();
}, 3000);
```

The `show()` and `hide()` methods can be called multiple times. If `show()` is called 10 times, then `hide()` should be called 10 times for loading to disappear. This is done for cases when this helper is used by multiple asynchronous scripts, the loading should not disappear until all scripts complete the work.

- `fw.capitalizeFirstLetter(text)` - capitalizes the first letter of a string.
- `fw.ops(properties, value, obj, delimiter)` - same as `fw_aks(...)` from *PHP Helpers*, but for javascript objects.

```
var obj = {foo: {}};

fw.ops('foo/bar', 'demo', obj);

console.log(obj); // {foo: {bar: 'demo'}}
```

- `fw.opg(properties, obj, defaultValue, delimiter)` - same as `fw_akg(...)` from *PHP Helpers*, but for javascript objects.

```
var obj = {foo: {bar: 'hello'}};

console.log( fw.opg('foo/bar', obj) ); // 'hello'
```

- `fw.randomMD5()` - generate random md5.

Options Modal

Modal with options. Display html generated from a given options array. After the user completes the form and presses "Save", values are available as a javascript object.

```
var modal = new fw.OptionsModal({
    title: 'Custom Title',
    options: [
        {'test_1': {
            'type': 'text',
            'label': 'Test1'
        }},
        {'test_2': {
            'type': 'textarea',
            'label': 'Test2'
        }}
    ],
    values: {
        'test_1': 'Default 1',
        'test_2': 'Default 2'
    },
    size: 'small' // 'medium', 'large'
```

```
});

// listen for values change
modal.on('change:values', function(modal, values) {
    console.log(values);
});

// replace values
modal.set('values', {
    'test_1': 'Custom 1',
    'test_2': 'Custom 2'
});

modal.open();
```

Note: Make sure to enqueue scripts and styles for the options you use in modal. Usually it is done before page is displayed.

```
fw()->backend->enqueue_options_static($modal_options);
```

Events

`fwEvents` is a global object on which you can trigger or listen custom events. This way different scripts can communicate with each other.

```
// script-1.js

fwEvents.on('script-2:message', function(data){
    console.log('script-1 received a message from script-2: '+ data.message);
});

// script-2.js

fwEvents.trigger('script-2:message', {message: 'Hello World!'});
```

6.9.4 CSS Helpers

Useful css classes for admin side. To use these helpers, add `fw` to your style dependencies:

```
wp_register_style(..., ..., array('fw'));
```

General

Alignment classes

Easily realign text to components with text alignment classes.

```
<p class="fw-text-left">Left aligned text.</p>
<p class="fw-text-center">Center aligned text.</p>
<p class="fw-text-right">Right aligned text.</p>
<p class="fw-text-justify">Justified text.</p>
<p class="fw-text-nowrap">No wrap text.</p>
```

Transformation classes

Transform text in components with text capitalization classes.

```
<p class="fw-text-lowercase">Lowercased text.</p>
<p class="fw-text-uppercase">Uppercased text.</p>
<p class="fw-text-capitalize">Capitalized text.</p>
```

Responsive images

Images can be made responsive-friendly via the addition of the `.fw-img-responsive` class. This applies `max-width: 100%;` and `height: auto;` to the image so that it scales nicely to the parent element.

```
<img src="..." class="fw-img-responsive" alt="image Responsive" >
```

Delete icon

Use the generic delete icon for links that delete something.

```
<a href="#" class="dashicons fw-x"></a>
```

Quick floats

Float an element to the left or right with a class. `!important` is included to avoid specificity issues. Classes can also be used as mixins.

```
<div class="fw-pull-left">...</div>
<div class="fw-pull-right">...</div>
```

Center content blocks

Set an element to `display: block` and center via margin. Available as a mixin and class.

```
<div class="fw-center-block">...</div>
```

Clearfix

Easily clear floats by adding `.fw-clearfix` to the parent element. Utilizes the micro clearfix as popularized by Nicolas Gallagher. Can also be used as a mixin.

```
<div class="fw-clearfix">...</div>
```

Showing and hiding content

Force an element to be shown or hidden. These classes use `!important` to avoid specificity conflicts, just like the quick floats. They are only available for block level toggling. They can also be used as mixins. Furthermore, `.fw-invisible` can be used to toggle only the visibility of an element, meaning its display is not modified and the element can still affect the flow of the document.

```
<div class="fw-show">...</div>
<div class="fw-hidden">...</div>
```

Image replacement

Utilize the `.fw-text-hide` class or mixin to help replace an element's text content with a background image.

```
<h1 class="fw-text-hide">Custom heading</h1>
```

Grid system

Css helpers includes a responsive, fluid grid system that appropriately scales up to 12 columns as the device or viewport size increases. Grid systems are used for creating layouts through a series of rows and columns that house your content. Here's how the grid system works:

- Use rows to create horizontal groups of columns.
- Content should be placed within columns, and only columns may be immediate children of rows.
- Predefined grid classes like `.fw-row` and `.fw-col-xs-4` are available for quickly making grid layouts.

- Grid columns are created by specifying the number of twelve available columns you wish to span. For example, three equal columns would use three `.fw-col-xs-4`.
- If more than 12 columns are placed within a single row, each group of extra columns will, as one unit, wrap onto a new line.
- Grid classes apply to devices with screen widths greater than or equal to the breakpoint sizes, and override grid classes targeted at smaller devices. Therefore, applying any `.fw-col-md-` class to an element will not only affect its styling on medium devices but also on large devices if a `.fw-col-lg-` class is not present.

This grid system was inspired from bootstrap with some modifications:

- Added `.fw-` prefix to classes
- Changed media queries screen sizes
- Rows are used without containers (no `.container` and `.container-fluid`)
- Rows have no padding

Media queries

We use the following media queries to create the key breakpoints to a narrower set of devices.

```
/* Extra small devices (phones) */
@media (max-width: 782px) { ... }

/* Small devices (tablets) */
@media (min-width: 783px) and (max-width: 900px) { ... }

/* Medium devices (desktops) */
@media (min-width: 901px) and (max-width: 1199px) { ... }

/* Large devices (large desktops) */
@media (min-width: 1200px) { ... }
```

Columns

Using a set of `.fw-col-*` classes, you can create grid systems that look good on any device:

- `.fw-col-xs-*` - extra small devices (phones).
- `.fw-col-sm-*` - small devices (tablets)
- `.fw-col-md-*` - medium devices (desktops)
- `.fw-col-lg-*` - large devices (large desktops)

Tip: More details about grid system and examples can be found here.

Responsive utilities

For faster mobile-friendly development, use these utility classes for showing and hiding content by device via media query.

Important: Try to use these on a limited basis and avoid creating entirely different versions of the same site. Instead, use them to complement each device's presentation.

Available classes

Use a single or combination of the available classes for toggling content across viewport breakpoints.

	Extra small devices (<783px)	Small devices (783px)	Medium devices (901px)	Large devices (1200px)
`.visible-xs-*`	**Visible**	Hidden	Hidden	Hidden
`.visible-sm-*`	Hidden	**Visible**	Hidden	Hidden
`.visible-md-*`	Hidden	Hidden	**Visible**	Hidden
`.visible-lg-*`	Hidden	Hidden	Hidden	**Visible**
`.hidden-xs`	**Hidden**	Visible	Visible	Visible
`.hidden-sm`	Visible	**Hidden**	Visible	Visible
`.hidden-md`	Visible	Visible	**Hidden**	Visible
`.hidden-lg`	Visible	Visible	Visible	**Hidden**

The `.visible-*-*` classes for each breakpoint come in three variations, one for each CSS display property value listed below.

Group of classes	CSS `display`
`.visible-*-block`	`display: block;`
`.visible-*-inline`	`display: inline;`
`.visible-*-inline-block`	`display: inline-block;`

So, for extra small (`xs`) screens for example, the available `.visible-*-*` classes are: `.visible-xs-block`, `.visible-xs-inline`, and `.visible-xs-inline-block`.

6.10 Filters & Actions

6.10.1 Actions

General

- `fw_init` - The framework is fully loaded and you can safely access any of its components. Useful when you need to init some theme components only when the framework is installed.

```
add_action('fw_init', '_action_theme_fw_init');
function _action_theme_fw_init() {
    $value = fw_get_db_customizer_option('hello');
    // fw()->...
}
```

- `fw_backend_add_custom_settings_menu` - Change **Theme Settings** menu. You can register the menu yourself as a top or sub menu, then the script will detect if it was registered (by slug) and will skip the default internal register.

```
add_action('fw_backend_add_custom_settings_menu', '_action_theme_custom_
↪fw_settings_menu');
function _action_theme_custom_fw_settings_menu($data) {
    add_menu_page(
        __( 'Awesome Settings', '{domain}' ),
        __( 'Awesome Settings', '{domain}' ),
        $data['capability'],
        $data['slug'],
        $data['content_callback']
    );
}
```

- `fw_backend_add_custom_extensions_menu` - Change **Unyson** menu. Works the same as *previous action*.

Assets Enqueue

- `fw_admin_enqueue_scripts:settings` - Enqueue assets only in **Theme Settings** page.

```
add_action('fw_admin_enqueue_scripts:settings', '_action_theme_enqueue_
↪scripts_theme_settings');
function _action_theme_enqueue_scripts_theme_settings() {
    wp_enqueue_script(
        'theme-settings-scripts',
        get_template_directory_uri() .'/js/admin-theme-settings.js',
        array('fw'),
        fw()->theme->manifest->get_version(),
        true
    );
}
```

- `fw_admin_enqueue_scripts:customizer` - Enqueue assets only in **Customizer** page.
- `fw_admin_enqueue_scripts:post` - Enqueue assets only in **Post Edit** page.

```
add_action('fw_admin_enqueue_scripts:post', '_action_theme_enqueue_
↪scripts_post_edit');
function _action_theme_enqueue_scripts_post_edit(WP_Post $post) {
    if ($post->post_type == 'page') {
        wp_enqueue_script(
```

```
            'page-edit-scripts',
            get_template_directory_uri() .'/js/admin-page-edit.js',
            array('fw'),
            fw()->theme->manifest->get_version(),
            true
        );
    }
}
```

- `fw_admin_enqueue_scripts:term` - Enqueue assets only in **Taxonomy Term Edit** page.

```
add_action('fw_admin_enqueue_scripts:term', '_action_theme_enqueue_
↪scripts_term_edit');
function _action_theme_enqueue_scripts_term_edit($taxonomy) {
    if ($taxonomy == 'category') {
        wp_enqueue_script(
            'category-edit-scripts',
            get_template_directory_uri() .'/js/admin-category-edit.js',
            array('fw'),
            fw()->theme->manifest->get_version(),
            true
        );
    }
}
```

Database

- `fw_post_options_update` - After database post option or all options were updated. The description of parameters can be found here.

```
add_action('fw_post_options_update', '_action_theme_fw_post_options_update
↪', 10, 4);
function _action_theme_fw_post_options_update($post_id, $option_id, $sub_
↪keys, $old_value) {
    if ($option_id === 'hello' && empty($sub_keys)) {
        // do something ...
    }
}
```

6.10.2 Filters

General

- `fw_framework_customizations_dir_rel_path` - Relative path of the customizations directory located in theme. By default it is `/framework-customizations`.

```
add_filter(
    'fw_framework_customizations_dir_rel_path',
    '_filter_theme_fw_customizations_dir_rel_path'
);
function _filter_theme_fw_customizations_dir_rel_path($rel_path) {
    /**
     * Make the directory name shorter. Instead of
     * {theme}/framework-customizations/theme/options/post.php
```

```
     * will be
     * {theme}/fw/theme/options/post.php
     */
    return '/fw';
}
```

Options

- `fw_settings_options` - Theme **Settings Options**, which are loaded from `{theme}/framework-customizations/theme/options/settings.php`

```
add_filter('fw_settings_options', '_filter_theme_fw_settings_options');
function _filter_theme_fw_settings_options($options) {
    $options['extra-tab'] = array(
        'type' => 'tab',
        'title' => __('Extra Tab', 'domain'),
        'options' => array(
            'test' => array('type' => 'text'),
        ),
    );

    return $options;
}
```

- `fw_customizer_options` - Theme **Customizer Options**, which are loaded from `{theme}/framework-customizations/theme/options/customizer.php`

```
add_filter('fw_customizer_options', '_filter_theme_fw_customizer_options
↪');
function _filter_theme_fw_customizer_options($options) {
    $options['extra-option'] = array('type' => 'text');

    return $options;
}
```

- `fw_post_options` - **Post Options**, which are loaded from `{theme}/framework-customizations/theme/options/posts/{post-type}.php`

```
add_filter('fw_post_options', '_filter_theme_fw_post_options', 10, 2);
function _filter_theme_fw_post_options($options, $post_type) {
    if ($post_type == 'page') {
        $options['extra-option'] = array('type' => 'text');
    }

    return $options;
}
```

- `fw_taxonomy_options` - **Taxonomy Term Options**, which are loaded from `{theme}/framework-customizations/theme/options/taxonomies/{taxonomy}.php`

```
add_filter('fw_taxonomy_options', '_filter_theme_fw_taxonomy_options', 10,
↪ 2);
function _filter_theme_fw_taxonomy_options($options, $taxonomy) {
    if ($taxonomy == 'category') {
        $options['extra-option'] = array('type' => 'text');
    }
```

```
    return $options;
}
```

6.11 Manifest

6.11.1 Introduction

The Framework, Theme and every Extension has a manifest. The manifest provides important information like: title, version, dependencies, etc.

The Framework generates a manifest for it self, the theme and every extension with default values automatically. You can overwrite the default values by creating a `manifest.php` file in the root folder of the theme or extension and define the `$manifest` array.

6.11.2 Framework

The framework's manifest is located in `framework/manifest.php` and can be accessed like this:

```
fw()->manifest->get('version');
```

It supports the following parameters:

```
<?php if (!defined('FW')) die('Forbidden');

$manifest = array();

$manifest['name']          = __('Framework', 'fw');
$manifest['uri']           = 'http://themefuse.com/framework';
$manifest['description']   = __('WordPress Framework', 'fw');
$manifest['version']       = '1.0';
$manifest['author']        = 'ThemeFuse';
$manifest['author_uri']    = 'http://themefuse.com/';
$manifest['requirements'] = array(
    'wordpress' => array(
        'min_version' => '4.0',
        /*'max_version' => '4.99.9'*/
    ),
);
```

6.11.3 Theme

The theme's manifest is located in `framework-customizations/theme/manifest.php` and can be accessed like this:

```
fw()->theme->manifest->get('version');
```

It supports the following parameters:

```
<?php if (!defined('FW')) die('Forbidden');

$manifest = array();
```

```
/**
 * An unique id to identify your theme
 * For e.g. this is used to store Theme Settings in wp_option 'fw_theme_settings_
↪options:{theme_id}'
 */
$manifest['id'] = get_option( 'stylesheet' );

/**
 * Specify extensions that you customized, that will look good and work well with␣
↪your theme.
 * After plugin activation, the user will be redirected to a page to install these␣
↪extensions.
 */
$manifest['supported_extensions'] = array(
    // 'extension_name' => array(),

    'page-builder' => array(),
    'breadcrumbs' => array(),
    'slider' => array(),
    // ...

    /**
     * These extensions are visible on Unyson Extensions page only if are specified␣
↪here.
     * Because they has no sense to be available for a theme that is not configured␣
↪to support them.
     */
    'styling' => array(),
    'megamenu' => array(),
);

$manifest['requirements'] = array(
    'wordpress' => array(
        'min_version' => '4.0',
        /*'max_version' => '4.99.9'*/
    ),
    'framework' => array(
        /*'min_version' => '1.0.0',
        'max_version' => '1.99.9'*/
    ),
    'extensions' => array(
        /*'extension_name' => array(),*/
        /*'extension_name' => array(
            'min_version' => '1.0.0',
            'max_version' => '2.99.9'
        ),*/
    )
);

// These keys are automatically fetched from theme styles.css
//$manifest['name'] = __('Theme Title', '{domain}');
//$manifest['description'] = __('Another awesome wordpress theme', '{domain}');
//$manifest['uri'] = 'http://themefuse.com/wp-themes-shop/theme-name';
//$manifest['version'] = '1.0';
//$manifest['author'] = 'ThemeFuse';
//$manifest['author_uri'] = 'http://themefuse.com/';
```

6.11.4 Extension

The extension's manifest is located in `{extension-name}/manifest.php` and can be accessed like this:

```
fw()->extensions->get('extension-name')->manifest->get('version');
```

It supports the following parameters:

```
<?php if (!defined('FW')) die('Forbidden');

$manifest = array();

$manifest['name']          = __('Extension Title', '{domain}');
$manifest['uri']           = 'http://extension-homepage.com/';
$manifest['description']   = __('Another awesome framework extension', '{domain}');
$manifest['version']       = '1.0';
$manifest['author']        = 'ThemeFuse';
$manifest['author_uri']    = 'http://themefuse.com/';
$manifest['requirements'] = array(
    'wordpress' => array(
        'min_version' => '4.0',
        /*'max_version' => '4.99.9'*/
    ),
    'framework' => array(
        /*'min_version' => '1.0.0',
        'max_version' => '1.99.9'*/
    ),
    'extensions' => array(
        /*'extension_name' => array(),*/
        /*'extension_name' => array(
            'min_version' => '1.0.0',
            'max_version' => '2.99.9'
        ),*/
    )
);
/**
 * @type bool Display on the Extensions page or it's a hidden extension
 */
$manifest['display'] = false;
/**
 * @type bool If extension can exist alone
 * false - There is no sense for it to exist alone, it exists only when is required␣
↪by some other extension.
 * true  - Can exist alone without bothering about other extensions.
 */
$manifest['standalone'] = false;
/**
 * @type string Thumbnail used on the Extensions page
 * All framework extensions has thumbnails set in the available extensions list
 * but if your extension is not in that list and id located in the theme, you can set␣
↪the thumbnail via this parameter
 */
$manifest['thumbnail'] = null;
```