Unyson Framework

Jun 29, 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.

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

CHAPTER 1

Minimum Requirements

- WordPress 4.4 or greater
- PHP version 5.2.4 or greater
- MySQL version 5.0 or greater

CHAPTER 2

Installation

The easiest (recommended) way to install the framework, is through the 'Plugins' menu: Search for plugin named `Unyson` and click the Install button.

Or you can install the plugin manually:

1. Upload the `unyson` folder to the `/wp-content/plugins/` directory
2. Activate the Unyson plugin through the 'Plugins' menu
3. Configure the plugin by going to the Unyson menu

CHAPTER 3

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.

3.1 Convention

3.1.1 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`

3.1.2 Prefixes

In the framework everything is prefixed to prevent naming conflicts and to give a meaning to functions, classes and methods names.

- *Core*

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()`

```php
/**
 * @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`.

```php
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_'`.

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

do_action('fw_whatever');
```

Theme

- Public functions and classes should be prefixed with:
  - `fw_theme_` for functions
  - `FW_Theme_` for classes

```php
function fw_theme_head() {
    // ...
}

class FW_Theme_Pagination
{
    // ...
}
```

- Private functions and classes should be prefixed with:
  - `_fw_theme_` for functions
  - `_FW_Theme_` for classes

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

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

- Functions used for hooks should be prefixed with:
  - `_action_theme_` for `add_action()`
  - `_filter_theme_` for `add_filter()`

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

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

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

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

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

do_action('fw_theme_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 with:
  - `_fw_ext_<extension-name>_` for functions
  - `_FW_Ext_<extension-name>_` for classes

- Functions used for hooks should be prefixed with:
  - `_action_fw_ext_<extension-name>_` for `add_action()`
  - `_filter_fw_ext_<extension-name>_` for `add_filter()`

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

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

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

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

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

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

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

do_action('fw_ext_demo_whatever');
```

3.1.3 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
```

(continues on next page)

(continued from previous page)

```
│           ├─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.

```php
$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`

```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.

3.2 Options

3.2.1 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' ),
```

(continues on next page)

(continued from previous page)

```
    ),
    '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.

3.2.2 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
```

(continues on next page)

(continued from previous page)

```
    $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.

3.2.3 Customizer

- *Introduction*
- *Examples*
- *Additional Arguments*
- *Live Preview*

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

```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}'),
            ),

        ),
    ),
);
```

- Create a Panel with Sections

```php
$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}'),
                    ),
```

(continues on next page)

(continued from previous page)

```
                ),
            ),

            '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.

```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}'),

                '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

```php
// 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(),
```

(continues on next page)

(continued from previous page)

```
            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).

3.2.4 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;
}
```

(continues on next page)

(continued from previous page)

```
/* 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.

3.2.5 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' ),
```

(continues on next page)

(continued from previous page)

```
    '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}'),
```

(continues on next page)

(continued from previous page)

```
    '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}'),
```

(continues on next page)

(continued from previous page)

```
    '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,
    /**
```

(continues on next page)

(continued from previous page)

```
     * 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,
)
```

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' ),
```

(continues on next page)

(continued from previous page)

```
    // 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}'),
```

(continues on next page)

(continued from previous page)

```
    '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',
```

(continues on next page)

(continued from previous page)

```
            '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.
```

(continues on next page)

(continued from previous page)

```
        '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.
        'step'        => 15        // Minutes step
        ),
    '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.
        'step'        => 15        // Minutes step
        )
    ),
    'value' => array(
        'from' => '',
        'to' => ''
    )
)
```

Icon v2

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

    /**
     * small | medium | large | sauron
     * Yes, sauron. Definitely try it. Great one.
     */
```

(continues on next page)

(continued from previous page)

```
    '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' ),
```

(continues on next page)

(continued from previous page)

```
    /**
     * 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.
```

(continues on next page)

(continued from previous page)

```
     * 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.
```

(continues on next page)

(continued from previous page)

```
        */
    ),
    '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.

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

    /**
     * By default, you don't have any shortcodes into the editor.
     *
     * You have two possible values:
     *   - false:   You will not have a shortcodes button at all
     *   - true:    the default values you provide in wp-shortcodes
```

(continues on next page)

(continued from previous page)

```
     *              extension filter will be used
     *
     *   - An array of shortcodes
     */
    'shortcodes' => false // true, array('button', map')

    /**
     * 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}'),
        )
```

(continues on next page)

(continued from previous page)

```
    ),
    /*
    '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(
```

(continues on next page)

(continued from previous page)

```
                '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 ...
```

(continues on next page)

(continued from previous page)

```
    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',
```

(continues on next page)

(continued from previous page)

```
                    '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.

Dynamic Multi Picker

While basic set of pre-defined pickers is enough in most of the cases, you may want to move it somewhere up or down from the main multi picker block. You may even want to move the picker in another tab so that your options looks more clean. In this case, the possibility of **detaching** the picker for the multi picker will help you a lot.

The first step is to define your picker somewhere in the same **form** (we name that a **context**) Please note that the `select` here is not nested under a `multi` or other option. Also, it is important to note that the ID for the `select` here is `gadget`.

```
$options = array(
    'gadget' => array(
        'type'    => 'select',
        'choices' => array(
            'phone'  => __('Phone', '{domain}'),
            'laptop' => __('Laptop', '{domain}')
        ),
    ),
);

// In view.php
$current_value = fw_akg('gadget', $atts);
```

Next, you would add the `multi-picker` body and **connect** it to that particular `select` with `gadget` ID.

```
$options = array(
    'gadget' => array(
        'type'    => 'select',
        'choices' => array(
            'phone'  => __('Phone', '{domain}'),
            'laptop' => __('Laptop', '{domain}')
        ),
    ),

    'multi-picker' => array(
        'type' => 'multi-picker',

        // Here is the ID of our select
```

(continues on next page)

(continued from previous page)

```
        'picker' => 'gadget',

        'choices' => array(
            'phone' => array(
                'text' => array(
                    'type' => 'text'
                )
            ),

            'laptop' => array(
                'text' => array(
                    'type' => 'icon-v2'
                )
            )
        )
    ),


);


// In view.php
$current_value = fw_akg('gadget', $atts);

$current_picker_value = fw_akg(
    'multi-picker/' . $current_value . '/text',
    $atts
);

fw_print($current_picker_value);
```

You would notice that the multi picker updates when that select changes.

You can even connect two (or three) multi pickers to the same picker.

```
$options = array(
    'gadget' => array(
        'type'    => 'select',
        'choices' => array(
            'phone'  => __('Phone', '{domain}'),
            'laptop' => __('Laptop', '{domain}')
        ),
    ),

    'first-multi-picker' => array(
        'type' => 'multi-picker',

        // Here is the ID of our select
        'picker' => 'gadget',

        'choices' => array(
            'phone' => array(
                // options for the first choice
            ),

            'laptop' => array(
                // options for the second choice
```

(continues on next page)

(continued from previous page)

```
            )
        )
    ),

    'second-multi-picker' => array(
        'type' => 'multi-picker',

        // Here is the ID of our select
        'picker' => 'gadget',

        'choices' => array(
            'phone' => array(
                // options for the first choice
            ),

            'laptop' => array(
                // options for the second choice
            )
        )
    ),
);
```

Enjoy!

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' ),
```

(continues on next page)

(continued from previous page)

```
    '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',
```

(continues on next page)

(continued from previous page)

```
    '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}'),
)
```

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);
    });
```

3.2.6 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';
```

(continues on next page)

(continued from previous page)

```
}
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)
```

(continues on next page)

(continued from previous page)

```
         *
         * 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' => ''
        );
```

(continues on next page)

(continued from previous page)

```
    }
}

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()`.

3.2.7 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;
        }
    }

    /**
```

(continues on next page)

(continued from previous page)

```
     * 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()
    ),
)
```

3.3 Extensions

3.3.1 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
├─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/
          └─...
```

3.3.2 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.

3.3.3 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*

- *Validate Child Extensions*
- *Download/Udpate Extension via GitHub*
- *Download/Udpate Extension via custom server.*

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
<?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');
    }

    // ...
}
```

Download/Udpate Extension via GitHub

Let's say you want to add some extensions available for your theme. To install extension from your GitHub repository you can add a file `available-extensions.php` in your `yourTheme/framework-customizations/theme/available-extensions.php` with the following code:

```
<?php defined( 'FW' ) or die();

// is executed when the extension is first installed
$extensions = array(
    'extension_name' => array(
        'display'     => true,
        'parent'      => null,
        'name'        => __( 'Extension name', 'fw' ),
        'description' => __( 'Extension description.', 'fw' ),
        'thumbnail'   => 'pathToThumbnail/thumbnail.jpg',
        'download'    => array(
            'source' => 'github',
            'opts'   => array(
                'user_repo' => 'yourUsername/yourRepositoryExtension'
            ),
        ),
    ),
);
```

To allow your extension to download updates from GitHub add in its manifest.php key with your URL to GitHub repository:

```
$manifest['github_repo'] = 'https://github.com/yourUsername/
↪yourRepositoryExtension';
```

On GitHub create first tag version release otherwise GitHub API can't return any version.

Download/Udpate Extension via custom server.

Let's say you want to add some extensions available for your theme. To install extension from your custom server you can add a file `available-extensions.php` in your `yourTheme/framework-customizations/theme/available-extensions.php` with the following code:

```
<?php defined( 'FW' ) or die();

// is executed when the extension is first installed
$extensions = array(
    'extension_name' => array(
        'display'     => true,
        'parent'      => null,
        'name'        => __( 'Extension name', 'fw' ),
        'description' => __( 'Extension description.', 'fw' ),
        'thumbnail'   => 'pathToThumbnail/thumbnail.jpg',
        'download'    => array(
            'source' => 'custom',
            'opts'   => array(
                // All these keys you can access on your server
                'remote'       => 'https://yourServerName/versions/', //␣
↪Required
                'purchase_key' => get_option( 'user_purchase_key' )
```

(continues on next page)

(continued from previous page)

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

To allow your extension to download updates from your server add in its manifest.php key with your URL to your server:

```
// All keys added in manifest you can access on your custom server so you␣
↪can check for purchase key, username, etc.
$manifest['remote'] = 'https://yourServerName/versions/';
```

On your server create a file index.php in `https://yourServerName/versions/index.php`:

```
<?php

// Here you have entire manifest sttings on update or on install extenions␣
↪array 'opts' from available-extensions.php
$set = json_decode( file_get_contents( 'php://input' ), true );

 /*
 * pull - what client's server wants to get. Version or theme/extension zip␣
↪archive
 * type - item type can be theme or extension
 */

$vars = array(
    'pull' => array( 'version', 'zip' ),
    'type' => array( 'theme', 'extension' )
);

foreach( $vars as $key => $values ) {
    if ( ! isset( $set[ $key ] ) || ! in_array( $set[ $key ], $values ) ) {
        die( json_encode( array( 'error' => 'Something went wrong. Please␣
↪contact support team.' ) ) );
    }
}

 /*
 *  If you have many extensions or themes, check if you have it.
 *  item - is name of extension or id of theme from manifest.php
 */
$my_products = array(
    'extension' => array( 'extension_name', 'extension_name1' ),
    'theme'     => array( 'scratch', 'scratch1' ),
);

if ( ! isset( $set['item'] ) || ! in_array( $set['item'], $my_products[ $set[
↪'type'] ] ) ) {
    die( json_encode( array( 'error' => 'This item doesn\'t exists.'  ) ) );
}

/**
 * Extract latest version for theme or extension.
 */
$version = '1.0.1';
```

(continues on next page)

(continued from previous page)

```
// Client's server asked for version just return it.
if ( $set['pull'] === 'version' ) {
    die( $version );
}

/*
    if ( empty( $set['purchase_key'] ) ) {
        die( json_encode( array( 'error' => 'Please insert purchase key.' ) ) );
    }

    // Check here if purchase_key is valid.
    if ( ! is_valid( $set['purchase_key'] ) ) {
        die( json_encode( array( 'error' => 'Purchase key is not valid.' ) ) );
    }
*/

/**
 * Build path to archive.
 */
$path = dirname( __FILE__ ) . "/{$set['item']}.{$version}.zip";

if ( ! file_exists( $path ) ) {
    die( json_encode( array( 'error' => 'File zip doesn\'t exists.' ) ) );
}

header( 'Content-Type: application/zip' );
header( 'Content-Disposition: attachment; filename=' .  basename( $path ) );
header( 'Content-Length: ' . filesize( $path ) );
readfile( $path );

die();
```

3.4 Components

3.4.1 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.

3.4.2 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'
```

3.4.3 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);
```

3.4.4 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();
}
```

3.5 Helpers

3.5.1 Introduction

Helpers are classes and functions with useful functionality. Here are built-in helpers that you can use:

- *PHP Helpers*
- *JavaScript Helpers*
- *CSS Helpers*

3.5.2 PHP Helpers

- *General*
- *Options*
- *Database*

General

General PHP helpers:

- `fw_print($value)` - styled version of `print_r()`.
- `fw_html_tag($tag, array $attr, $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;

/*
```

(continues on next page)

(continued from previous page)

```
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);

/*
```

(continues on next page)

(continued from previous page)

```
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_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_human_bytes($bytes)` - convert bytes to human readable size.

```
echo fw_human_bytes(12345);

// '2.06 KB'
```

- `fw_strlen($string)` - UTF-8 version of php's `strlen()`.

```
echo strlen('!'), PHP_EOL;
echo fw_strlen('!'), PHP_EOL;
```

(continues on next page)

(continued from previous page)

```
// 13
// 7
```

- `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

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(
```

(continues on next page)

(continued from previous page)

```
        '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_Cache

Use cache to store frequently accessed data. Cache is just a big array and has one useful feature: it will automatically 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.

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::POST('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:

```php
/**
 * @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:

```php
// 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:

```javascript
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.
    });
});
```

FW_Settings_Form

The easiest way to create admin settings forms.

1. Extending the `FW_Settings_Form` class. Create `{theme}/class-fw-settings-form-test.php`:

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

class FW_Settings_Form_Test extends FW_Settings_Form {
    public function get_values() {
        return get_option('test_fw_settings_form', array());
    }

    public function set_values($values) {
        update_option('test_fw_settings_form', $values, false);
    }

    public function get_options() {
        return array(
            'tab1' => array(
                'type' => 'tab',
                'title' => 'Tab #1',
                'options' => array(
                    'opt1' => array(
                        'type' => 'text',
                        'label' => 'Option #1'
                    ),
                    'opt2' => array(
                        'type' => 'textarea',
                        'label' => 'Option #2'
                    ),
                ),
            ),
            'tab2' => array(
                'type' => 'tab',
                'title' => 'Tab #2',
                'options' => array(
                    'tab2_1' => array(
                        'type' => 'tab',
                        'title' => 'Sub Tab #1',
                        'options' => array(
                            'opt2_1' => array(
                                'type' => 'text',
                                'label' => 'Sub Option #1'
                            ),
                        ),
                    ),
                    'tab2_2' => array(
                        'type' => 'tab',
                        'title' => 'Sub Tab #2',
                        'options' => array(
                            'opt2_2' => array(
                                'type' => 'textarea',
                                'label' => 'Sub Option #2'
                            ),
                        ),
                    ),
                ),
            ),
        );
    }
```

(continues on next page)

(continued from previous page)

```
    protected function _init() {
        add_action('admin_menu', array($this, '_action_admin_menu'));
        add_action('admin_enqueue_scripts', array($this, '_action_admin_
↪enqueue_scripts'));

        $this->set_is_side_tabs(true);
        $this->set_is_ajax_submit(true);
    }

    /**
     * @internal
     */
    public function _action_admin_menu() {
        add_menu_page(
            'Test FW Settings Form',
            'Test FW Settings Form',
            'manage_options',
            /** used in @see _action_admin_enqueue_scripts() */
            'test-fw-settings-form',
            array($this, 'render')
        );
    }

    /**
     * @internal
     */
    public function _action_admin_enqueue_scripts() {
        $current_screen = get_current_screen(); // fw_print($current_
↪screen);

        // enqueue only on settings page
        if ('toplevel_page_'. 'test-fw-settings-form' === $current_screen-
↪>base) {
            $this->enqueue_static();
        }
    }
}
```

2. Include and initialize your class. Add in `{theme}/functions.php`:

```
add_action('fw_init', '_action_theme_test_fw_settings_form');
function _action_theme_test_fw_settings_form() {
    if (class_exists('FW_Settings_Form')) {
        require_once dirname(__FILE__) . '/class-fw-settings-form-test.php
↪';
        new FW_Settings_Form_Test('test');
    }
}
```

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(); } ?>
```

3.5.3 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'
```

(continues on next page)

(continued from previous page)

```
});

// 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);
```

Confirmation

General purpose confirmation mechanism that operates with `jQuery.Deferred`.

```
var confirm = fw.soleConfirm.create({
  severity: 'info', // warning | info
  message: 'Some message to display', // or null, if you don't want any
  backdrop: null // null | false | true
});

confirm.result; // Instance of jQuery.Deferred factory

confirm.result.then(function (confirm_instance) {
  // confirm_instance is same as confirm

  // Handle success branch
});

confirm.result.fail(function (confirm_instance) {
  // Handle fail branch
});

confirm.show();
```

Queueing confirms

Confirm is actually using `fw.soleModal` under the hood, which is queued one after the other.

```
var confirm1 = fw.soleConfirm.create();
var confirm2 = fw.soleConfirm.create();
```

(continues on next page)

(continued from previous page)

```
confirm1.show();
confirm2.show();

confirm1.hide(); // That's when the confirm2 will actually pop in, results are␣
↪buffered
```

Same confirm multiple times

Because of the way `jQuery.Deferred` works, one single confirm instance will resolve it's promise exactly one time. If you really need to use the same confirm once again - just reset it.

```
var confirm = fw.soleConfirm.create();

confirm.result.then(function () {
  // handle success
  // will be triggered just once
});

confirm.show();

// ...
// after the user takes his choice
// ...

confirm.show(); // will throw an error!
confirm.reset();

// you'll have to attach your listeners once again, the old one
// will already not be around
confirm.result.then(function () {
  // one more handler
});

confirm.show(); // will work just fine
```

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!'});
```

Reactive Options and Fetch Html helper

This section is a draft for an upcoming documentation for reactive option types.

For now, it documents only fw.options.fetchHtml() helper.

```
fw.options.fetchHtml(
    // An array of options
    {
        a: {
            type: 'text',
            label: 'My Option'
        },

        b: {
            type: 'password'
        }
    },


    // The array of default values
    {
        a: 'Hello'
    }
).then(function (html) {
    console.log(html);
});
```

- `fw.options.fetchHtml(options, values)` - fetch the HTML representation for every option, returns a Promise
- `fw.options.fetchHtml.getCacheEntryFor(options, values)` - get current cached HTML string or false
- `fw.options.fetchHtml.emptyCache()` - empty cache and force each option type to be re-downloaded

3.5.4 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`.

3.6 Manifest

3.6.1 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.

3.6.2 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');
```

(continues on next page)

(continued from previous page)

```
$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'*/
    ),
);
```

3.6.3 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(),
);
```

(continues on next page)

(continued from previous page)

```
$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/';
```

3.6.4 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(
```

(continues on next page)

(continued from previous page)

```
        /*'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;
```

3.7 Built-in Extensions

3.7.1 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*

3.7.2 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
```

(continues on next page)

(continued from previous page)

```
├──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]`.

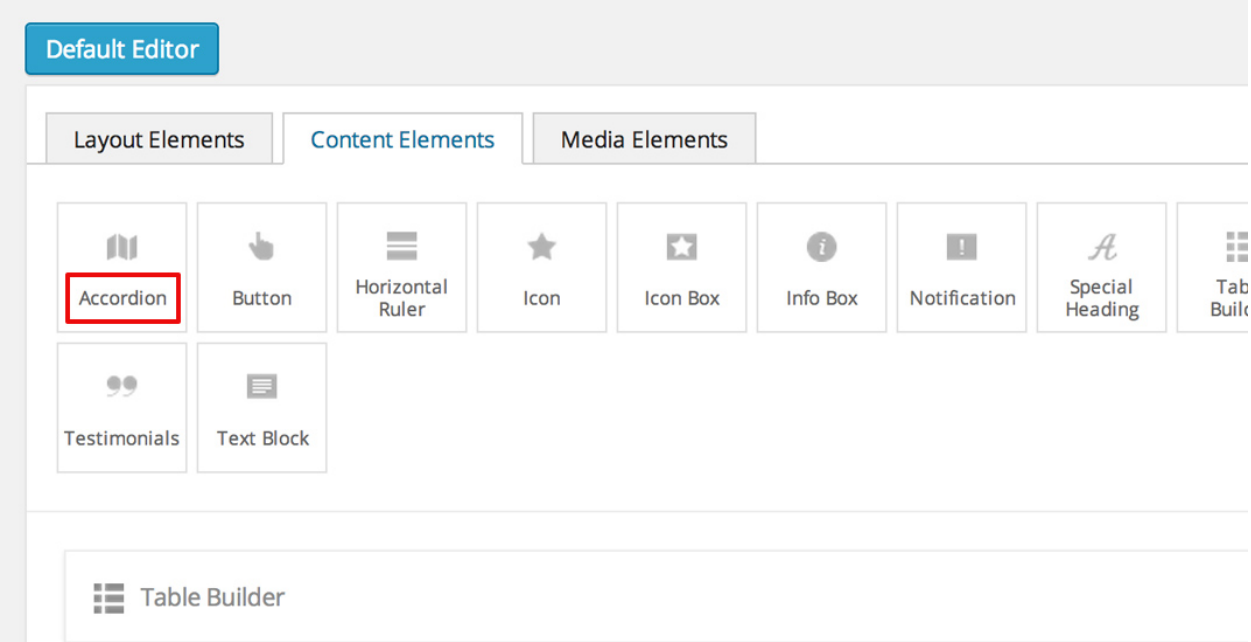
Config File

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.

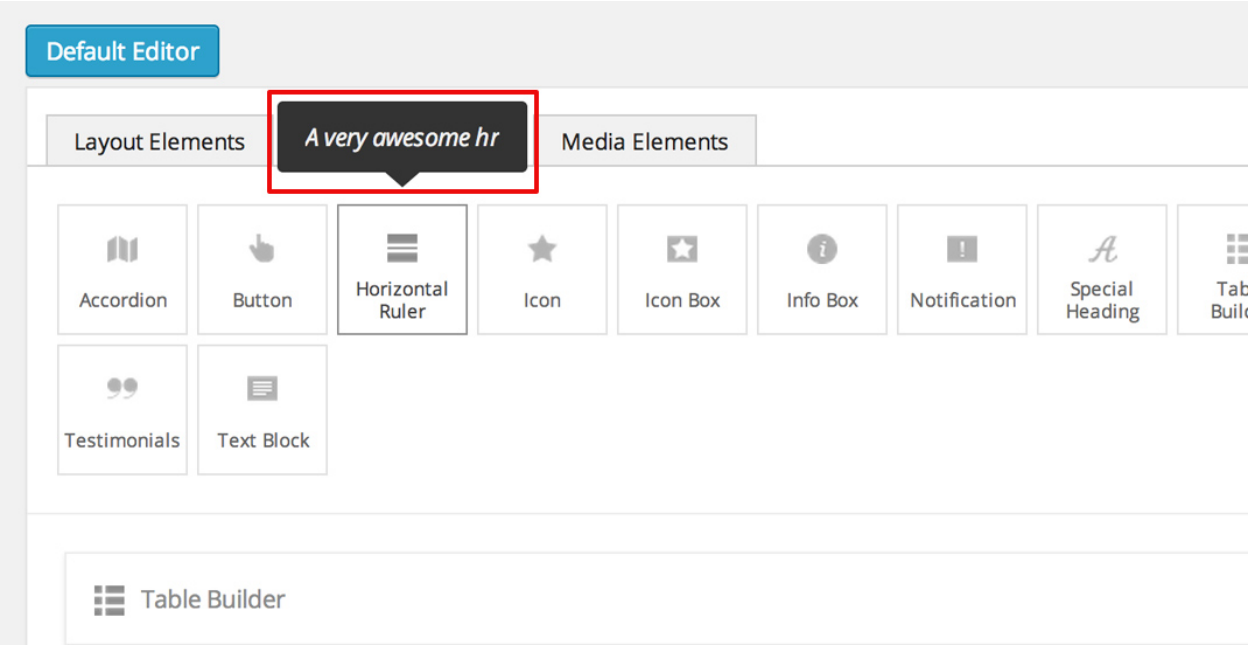
```php
$cfg = array(
    'page_builder' => array(
        'title'         => __('Demo Shortcode', '{domain}'),
        'description'   => __('Demo shortcode description', '{domain}'),
        'tab'           => __('Demo Elements', '{domain}'),
        'popup_size'    => 'small', // can be large, medium 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:

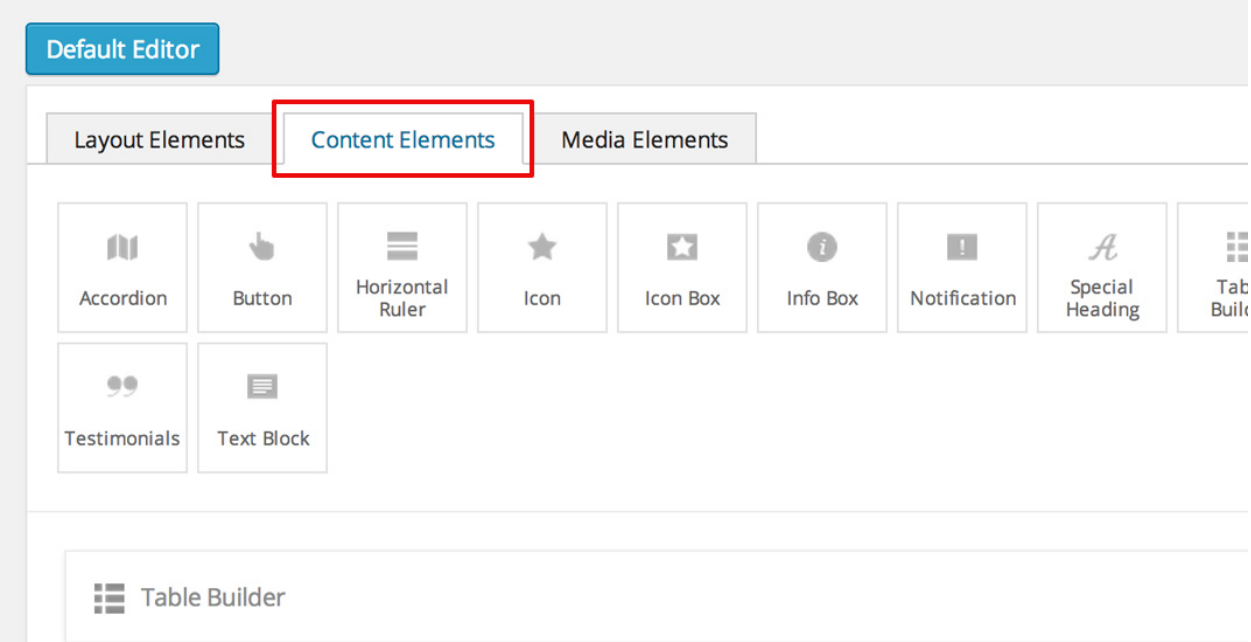
- `title` - the title that will appear in the shortcode box.



- `description` - the text that will be shown in a tooltip when hovering the shortcode box.

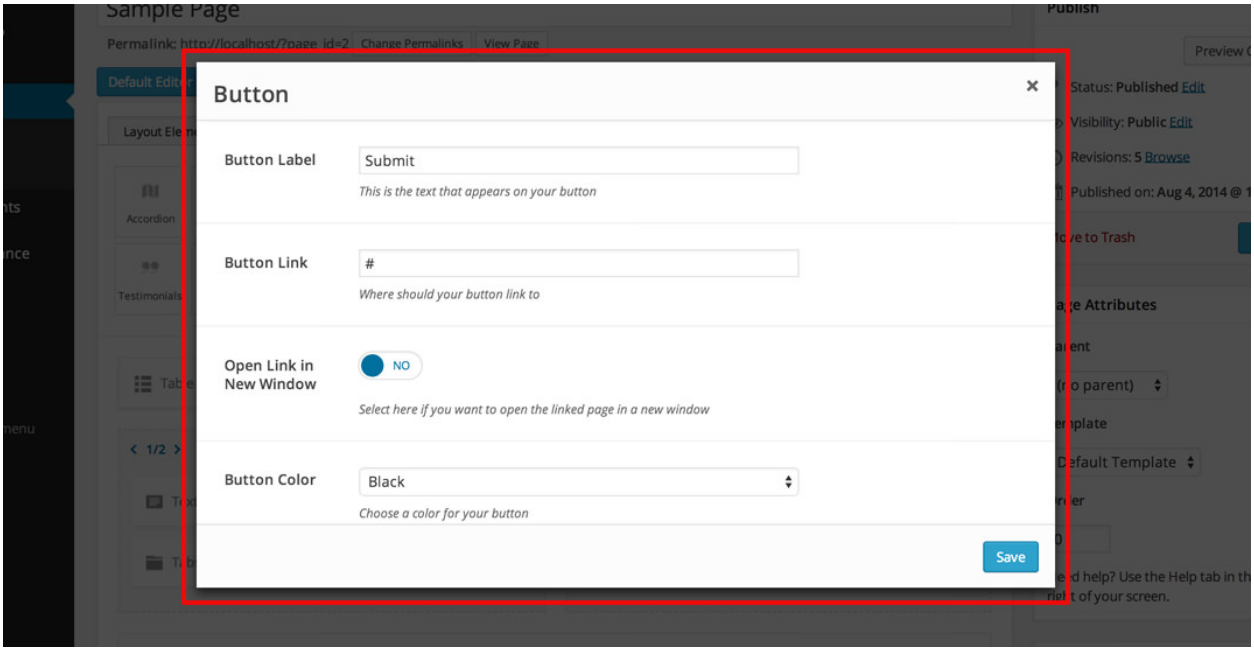


- `tab` - the builder tab in which the shortcode box will appear.



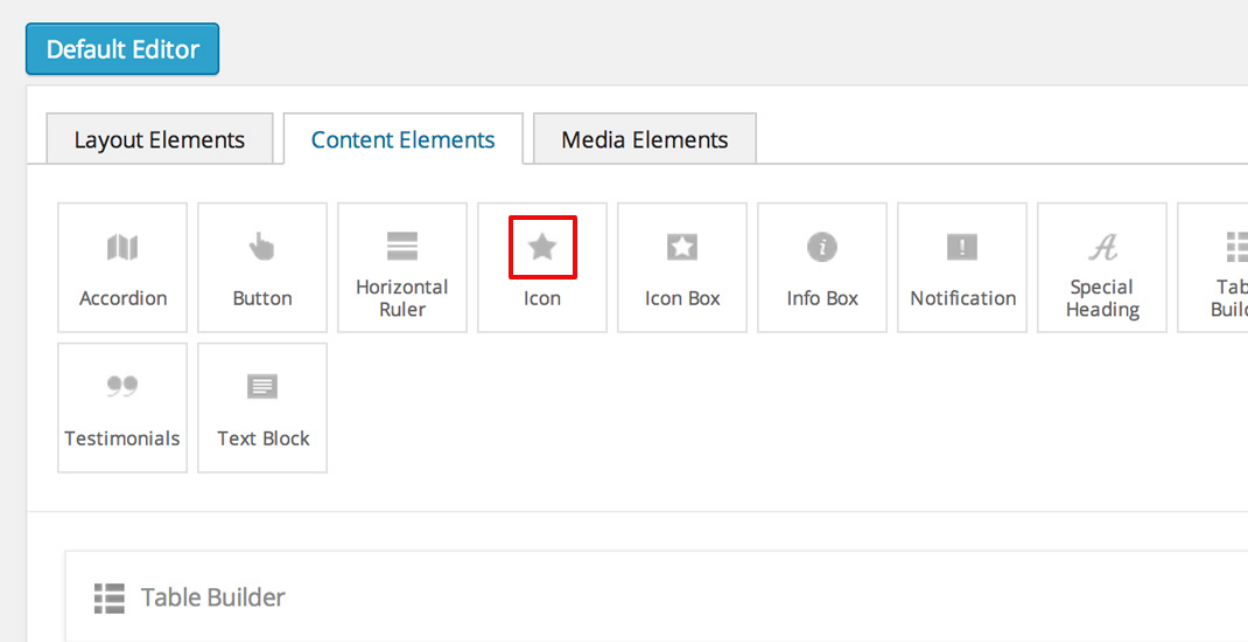
- `popup_size` - the size of the popup in which the *shortcode options* will be displayed.

  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.

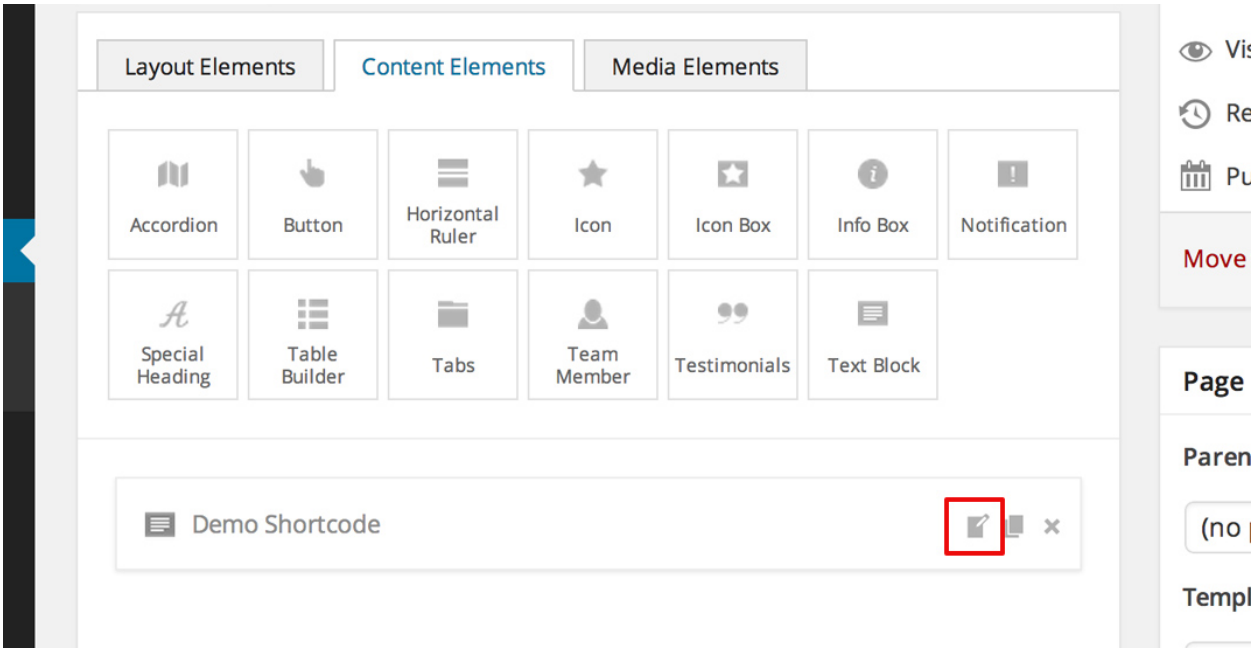


Options file

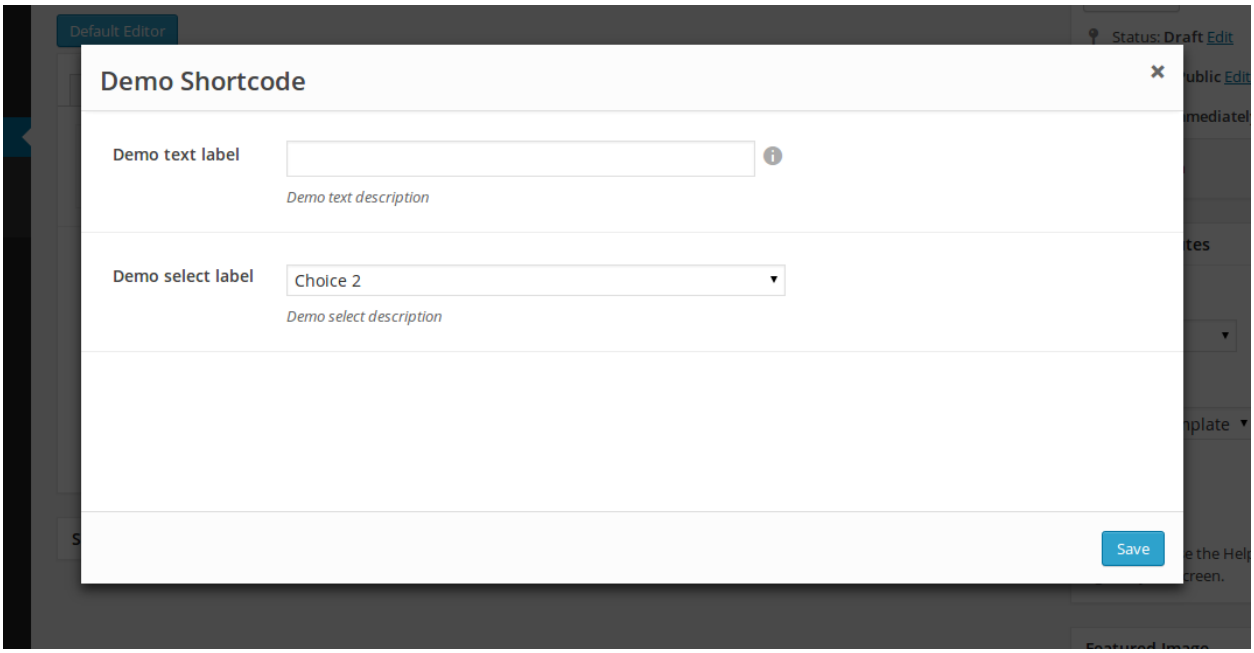
The shortcode directory can contain a file named `options.php` with correctly formed *options*:

```
$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

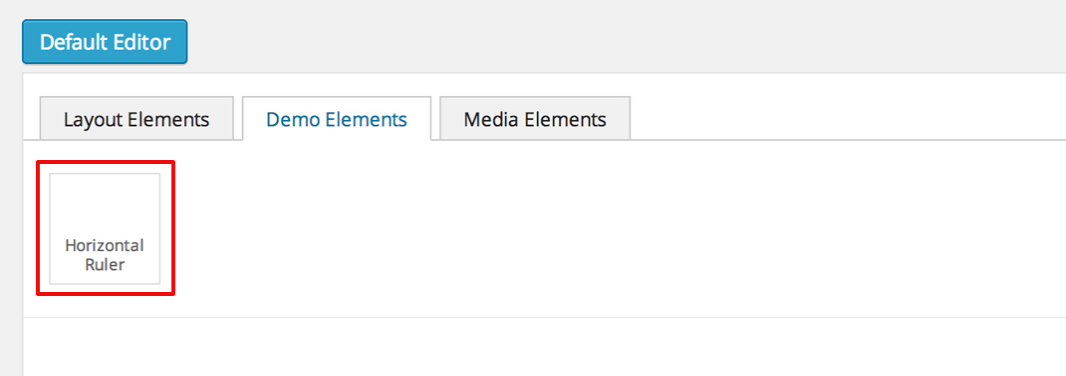
Creating a simple shortcode

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/
└─extensions/
  └─shortcodes/
    └─shortcodes/
      └─hr/
        ├─config.php
        └─views/
          └─view.php
```

Creating a shortcode with options

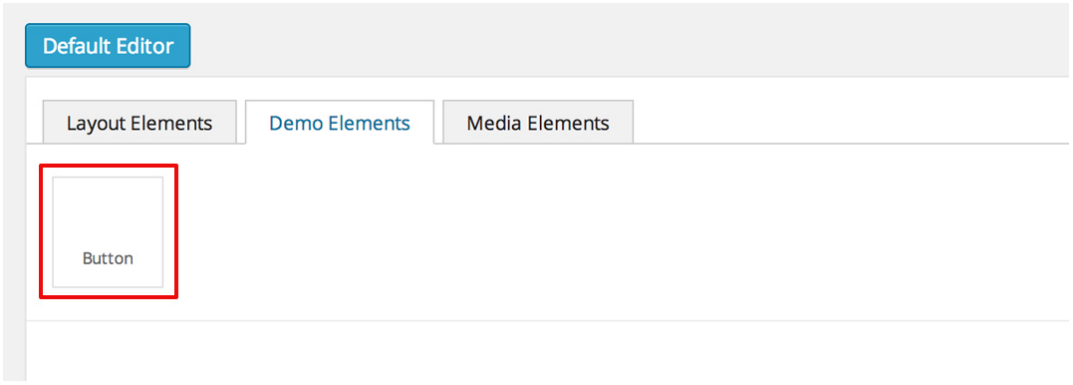
This example will go through creating the `[button]` shortcode.

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:

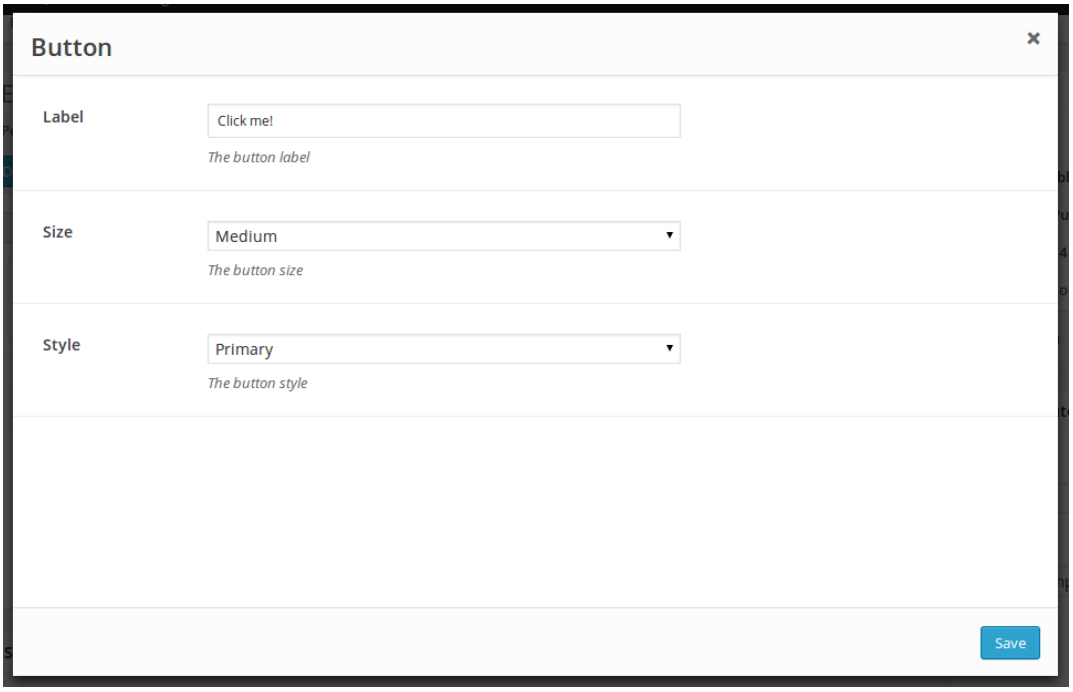


Tip: To add an icon to the shortcode see the *icon section*.

3. Create an *options file* with the options for **label**, **size** and **style**:

```php
<?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
```

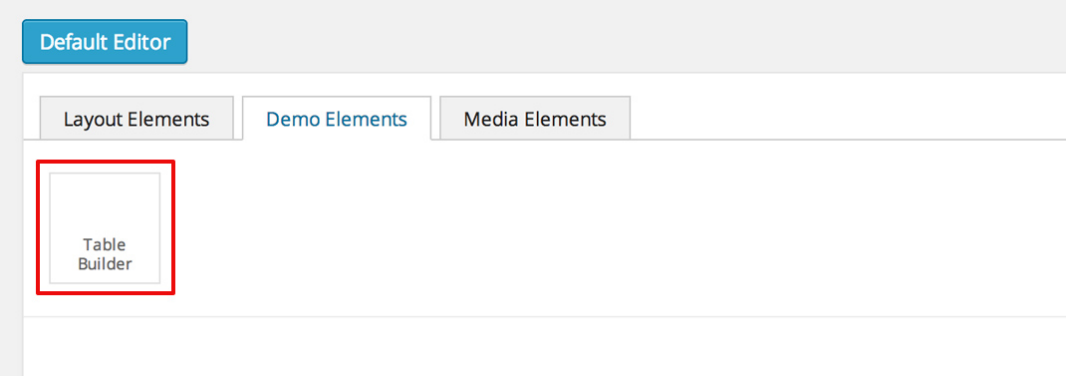
Creating an advanced shortcode with a custom class

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 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');
```

(continues on next page)

(continued from previous page)

```
    }

    // ...

}
```

(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
```

(continues on next page)

(continued from previous page)

```
        ├─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
```

(continues on next page)

(continued from previous page)

```
 * @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;
```

3.7.3 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
```

(continues on next page)

(continued from previous page)

```
    ├─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 `demo-slider`, so the child extension directory will be `framework-customizations/extensions/media/extensions/slider/extensions/demo-slider`.

Important: Make sure you have `framework-customizations/extensions/media/extensions/slider/extensions/demo-slider/manifest.php` with the following contents:

```php
<?php $manifest['standalone'] = true;
```

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
```

(continues on next page)

(continued from previous page)

```
 */
$cfg['multimedia_types'] = array('image');
```

Static files

Scripts, styles and images are stored in the `static/` directory.

- `static/images/` - directory for images. This directory has 2 special images that you should create:
  - `thumb.jpg` - small image with frontend preview of this slider type. Is displayed on the admin side in Slider Type choices.
  - `preview.jpg` - a bigger image with frontend preview of this slider type. It is displayed when the user hovers the `thumb.jpg` 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`.

For `demo-sloder` to work:

1. Download this script in `framework-customizations/extensions/media/extensions/slider/extensions/demo-slider/static/js/unslider-min.js`.
2. Download this style in `framework-customizations/extensions/media/extensions/slider/extensions/demo-slider/static/css/unslider.css`.

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.

Template

View the file that contains the slider template for frontend, is located in `views/{slider-type}.php`. Here is an example for our `demo-slider`:

```
<?php if (!defined('FW')) die('Forbidden');
/**
 * @var array $data
 */

$unique_id = 'demo-slider-'. fw_unique_increment();
?>
<?php if (isset($data['slides'])): ?>
    <script type="text/javascript">
```

(continues on next page)

(continued from previous page)

```
        jQuery(function($){ $('#<?php echo $unique_id ?>').unslider(); });
    </script>
    <div id="<?php echo $unique_id ?>">
        <ul>
            <?php foreach ($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>
    </div>
<?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',
            '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',
            ...
        )
    )
);
```

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 this case the slider type is `demo-slider`, so the class file will be located in `framework-customizations/extensions/media/extensions/slider/extensions/bx-slider/class-fw-extension-demo-slider.php` and will contain:

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

class FW_Extension_Demo_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($slider_post_id, array(
    'width'  => 300,
    'height' => 200
));
```

3.7.4 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">
```

(continues on next page)

(continued from previous page)

```
                    <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>
```

Change 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;
```

Overwrite the Walker

1. Create the walker class

```
// file:: {theme}/framework-customizations/extensions/megamenu/includes/class-fw-ext-
↪mega-menu-custom-walker.php

class FW_Ext_Mega_Menu_Custom_Walker extends FW_Ext_Mega_Menu_Walker
 {
     function start_lvl( &$output, $depth = 0, $args = array(), $class = 'sub-menu' )
↪{
         fw_print('Hello');

         return parent::start_lvl($output, $depth, $args, $class);
```

(continues on next page)

(continued from previous page)

```
    }

    // other customizations ...
}
```

2. Overwrite the default walker via filter

```
// file: {theme}/framework-customizations/extensions/megamenu/hooks.php

// replace default walker
{
    remove_filter('wp_nav_menu_args', '_filter_fw_ext_mega_menu_wp_nav_menu_args');

    /** @internal */
    function _filter_theme_ext_mega_menu_wp_nav_menu_args($args) {
        $args['walker'] = new FW_Ext_Mega_Menu_Custom_Walker();

        return $args;
    }
    add_filter('wp_nav_menu_args', '_filter_theme_ext_mega_menu_wp_nav_menu_args');
}
```

Item Custom Options

1. *Overwrite* these options in your theme.
2. Get the saved db value (it has the *same structure* as `multi-picker` option-type value)

```
if ($item_type = fw_ext_mega_menu_get_db_item_option($item_id, 'type')) {
    $values    = fw_ext_mega_menu_get_db_item_option($item_id, $item_
↪type);
}
```

3. Adapt options popup sizes by overwriting these config keys.

3.7.5 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

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

// file: framework-customizations/extensions/sidebars/config.php
```

(continues on next page)

(continued from previous page)

```
$cfg = array(
    'sidebar_positions' => array(
        'position-id' => array(
            /**
             * Image from: framework-customizations/extensions/sidebars/images/
             * (required)
             */
            'icon_url' => 'picture.png',
            /**
             * Number of sidebars on page.
             * The maximum number is 4.
             * (optional)
             * (default 0)
             */
            'sidebars_number' => 0
        ),
        // other positions ...
    ),
    /**
     * Array that will be passed to register_sidebar($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_get_current_position();

if ($current_position !== 'position-id') {
```

(continues on next page)

(continued from previous page)

```
    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_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_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;
```

(continues on next page)

(continued from previous page)

```
}
add_filter('fw_ext_sidebars_conditional_tags', '_filter_fw_ext_sidebars_
↪add_conditional_tag' );
```

3.7.6 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=""/>
```

(continues on next page)

(continued from previous page)

```
                    </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
 */
```

(continues on next page)

(continued from previous page)

```
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.

3.7.7 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.

Migration

Migration is a term representing moving a WordPress website from one location (e.g. http://localhost/site) to another (e.g. http://site.com).

This is achieved by:

1. Making a full backup copy of the old site(http://localhost/site).
2. Moving it in the `wp-content/uploads/fw-backup` directory on the new site(http://site.com).

After opening the Backup page a new archive will be displayed in the Backup Archive list.

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.

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/.../scrensh ot.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;
        }
```

(continues on next page)

(continued from previous page)

```
    }

    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;
```

(continues on next page)

(continued from previous page)

```
}
add_filter('fw_ext_backups_demo_dirs', '_filter_theme_fw_ext_backups_demo_
↪dirs');
```

Troubleshooting

Requirements

If you have trouble with install demo content or backup content please make sure you have these recommended php values:

```
upload_max_filesize = 128M
max_input_time = -1
post_max_size = 128M
max_input_vars = 8000
max_execution_time = 200
```

Make sure you have enough disk space. The full backup backups the entire wp-content folder and sometimes users have there a lot of "trash" like backups from other plugins, cache ... Some times other plugins backups are really huge 2-4GB. You have to make sure that you do not have the same problem.

LiteSpeed webserver

The admin notice "Unyson: Your website is hosted using the LiteSpeed web server. Please consult this article if you have problems backing up." means that your web hosting company uses the LiteSpeed webserver. LiteSpeed appears to have problems with all WordPress scheduled tasks that last more than a very short time – including all backup plugins. Adding this in an early position in the .htaccess file in your WordPress root folder may fix the problem:

```
RewriteRule (wp-cron|class-fw-extension-backups|class--fw-ext-backups-module-
↪tasks)\.php - [E=noabort:1]
```

Adding the above line does not mean that the problem is definitely fixed, you will only know that via testing. If the above does not help, then you can try to add a line to your wp-config.php(WordPress's alternative scheduling system):

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

If that does not help you, then you'll need the help of your web hosting company to see why WordPress's scheduler isn't working on their setup, or is terminating it prematurely. Or failing that, you'll need a different web hosting company. This problem is a generic one affecting all backup plugins on WordPress that run via the scheduler (which is all of them, as far as we know).

3.7.8 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:

```php
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
```

(continues on next page)

(continued from previous page)

```
        );

        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(
```

(continues on next page)

(continued from previous page)

```
                    '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 +'">'+
```

(continues on next page)

(continued from previous page)

```
                '<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) {
```

(continues on next page)

(continued from previous page)

```
                    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);
        },
```

(continues on next page)

(continued from previous page)

```
        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;
```

(continues on next page)

(continued from previous page)

```
            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;
}
```

(continues on next page)

(continued from previous page)

```
.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
            )
```

(continues on next page)

(continued from previous page)

```
        );
    }
}
```

- 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)
    {
```

(continues on next page)

(continued from previous page)

```
        $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).

3.7.9 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

- `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' );
```

3.7.10 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.

```php
/**
 * @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:

```php
'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

```php
/**
 * @internal
 */
function _filter_add_my_seo_tag($tags) {
    $tags['mytag'] = array(
```

(continues on next page)

(continued from previous page)

```
        '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

```
/**
 * 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',
            )
        )
```

(continues on next page)

(continued from previous page)

```
    ),
    '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.

3.7.11 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_fw_ext_events_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_fw_ext_
↪events_render_html');
```

- `fw_ext_events_post_slug` - event custom post slug

```
/** @internal */
function _filter_theme_fw_ext_events_custom_events_post_slug($slug) {
    return 'event';
}
add_filter('fw_ext_events_post_slug', '_filter_theme_fw_ext_events_custom_
↪events_post_slug');
```

- `fw_ext_events_taxonomy_slug` - event taxonomy slug

```
/** @internal */
function _filter_theme_fw_ext_events_custom_events_taxonomy_slug($slug) {
    return 'events';
}
add_filter('fw_ext_events_taxonomy_slug', '_filter_theme_fw_ext_events_
↪custom_events_taxonomy_slug');
```

- `fw_ext_events_post_type_name` - event custom post labels (plural and singular)

```
/** @internal */
function _filter_theme_fw_ext_events_event_labels($labels) {
    $labels = array(
        'singular' => __('Custom Event', '{domain}'),
        'plural'   => __('Custom Events', '{domain}'),
    );

    return $labels;
}
add_filter('fw_ext_events_post_type_name', '_filter_theme_fw_ext_events_
↪event_labels');
```

- `fw_ext_events_category_name` - event taxonomy labels (plural and singular)

```
/** @internal */
function _filter_theme_fw_ext_events_event_tax_labels_names($labels) {
```

(continues on next page)

(continued from previous page)

```
    $labels = array(
        'singular' => __( 'Custom Category', '{domain}' ),
        'plural'   => __( 'Custom Categories', '{domain}' ),
    );

    return $labels;
}
add_filter( 'fw_ext_events_category_name', '_filter_theme_fw_ext_events_
↪event_tax_labels_names' );
```

- `fw_ext_events_post_options` - custom options for event

```
/** @internal */
function _filter_theme_fw_ext_events_custom_options($options) {
    return array_merge($options, array(
        'events_tab_1' => array(
            'title'   => __('Test title', '{domain}'),
            'type'    => 'tab',
            'options' => array(
                'demo_text_id' => array(
                    'type'  => 'text',
                    'label' => __('Demo Text label', '{domain}'),
                    'desc'  => __('Demo text description', '{domain}'),
                )
            )
        )
    ));
}
add_filter('fw_ext_events_post_options', '_filter_theme_fw_ext_events_
↪custom_options');
```

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'
                    )
                )
            )
        )
    );
}
```

3.7.12 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.

3.7.13 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
<?php if (!defined('FW')) die('Forbidden');

if (!is_admin()) {
    wp_register_style(
        'fw-ext-builder-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_option_types_init', '_action_include_demo_lists_builder');
```

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',
```

(continues on next page)

(continued from previous page)

```
                // 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
```

(continues on next page)

(continued from previous page)

```
     * @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';
    }
```

(continues on next page)

(continued from previous page)

```
    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',
```

(continues on next page)

(continued from previous page)

```
            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
```

(continues on next page)

(continued from previous page)

```
                 * 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({
```

(continues on next page)

(continued from previous page)

```
                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.

3.7.14 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*
- *Views*
- *Hooks*
- *Stars Feedback*

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.

3.7.15 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' );
```

3.7.16 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*

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 );
```

3.7.17 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

3.8 Filters & Actions

3.8.1 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') {
```

(continues on next page)

(continued from previous 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 ...
    }
}
```

3.8.2 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) {
    /**
```

(continues on next page)

(continued from previous page)

```
     * 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) {
```

(continues on next page)

(continued from previous page)

```
    if ($taxonomy == 'category') {
        $options['extra-option'] = array('type' => 'text');
    }

    return $options;
}
```

- `fw_shortcode_get_options` - **Page builder shortcodes options**, can be loaded from anywhere functions.php, your php file but before wordpress hook `add_meta_boxes`

```
add_action( 'fw_shortcode_get_options', '_filter_theme_fw_shortcode_get_
↪options', 10, 2 );
function _filter_theme_fw_shortcode_get_options( $options, $shortcode ) {

    $options = array(
        'default_options' => array(
            'type'    => 'tab',
            'options' => $options, // Add default options to the first␣
↪tab.
            'title'   => __( 'Tab with default shortcode options', '
↪{domain}' ),
            'attr'    => array( 'class' => 'custom-class', 'data-foo' =>
↪'bar' ),
        ),
        'new_tab_options'           => array(
            'type'    => 'tab',
            'options' => array(
                'option_id' => array( 'type' => 'text' ),
            ),
            'title'   => __( 'Tab with our custom options', '{domain}' ),
            'attr'    => array( 'class' => 'custom-class', 'data-foo' =>
↪'bar' ),
        )
    );

    return $options;
}
```