Unyson Framework

Release

Jun 08, 2017

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

Installation

Install the framework with the default theme

If you want to start creating a theme based on the Unyson default theme:

1. Download the archive from the Unyson GitHub repository.
2. Extract the zip in the `wp-content/themes` directory.

Install only the framework

If you already started to create a theme and want to include the Unyson framework in it:

1. Download the archive from the framework's GitHub repository.
2. Extract it to your parent theme directory. After this you must have `framework/` directory in parent theme. It's mandatory to have this exact same folder structure otherwise it will not work.
3. Include the Unyson framework by adding this line in your theme's `functions.php`:

```
require dirname(__FILE__) .'/framework/bootstrap.php';
```

CHAPTER 2

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.

Convention

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`

- First line in all php files should be:

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

Prefixes

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

Core

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

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

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

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

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

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

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

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

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

    – _filter_ for add_filter()

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

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

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

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

        return $content;
    }
}
```

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

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

do_action('fw_whatever');
```

Theme

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

```
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`:

```
/**
 * @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`:

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

do_action('fw_ext_demo_whatever');
```

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 Unyson:

```
themes/
-parent-theme/
| -framework/
| -framework-customizations/
|   -extensions/
|   | -extension-name/
|   | -...
|   -theme/
|     -config.php     # Theme specific configuration
|     -static.php     # wp_enqueue_style() and wp_enqueue_script()
|     -posts.php      # register_post_type() and register_taxonomy()
|     -menus.php      # register_nav_menus()
|     -hooks.php      # add_filter() and add_action()
|     -helpers.php    # Helper functions and classes
|     -manifest.php   # Theme details: title, description, version, dependencies, etc.
|     -options/       # Files containing options
|     | -settings.php # Contains framework settings options
|     | -posts/       # Files containing post types options
|     | | -post.php
|     | | -testimonial.php
|     | | -{post-type}.php
|     | | -...
|     | -taxonomies/  # Files containing taxonomy terms options
|     |   -category.php
|     |   -post_tag.php
|     |   -{taxonomy}.php
|     |   -...
|     -widgets/       # Theme widgets
|     | -{widget-name}/
|     | | -class-fw-widget-{widget-name}.php # class FW_Widget_{Widget_Name} extends␣
↪WP_Widget { ... }
|     | | -views/
|     | |   -some-view.php
|     | |   -...
|     | -...
|     -includes/      # All .php files are auto included (no need to require_once)
|       -some-file.php
|       -...
-child-theme/
  -framework-customizations/
    -... # same as in parent theme, but here you can overwrite specific files from␣
↪parent theme
```

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

- `framework/` - In this directory you'll find the framework. Do not change anything in it, because all the files from that directory will be replaces on a framework update and you'll lose all the changes.
- `framework-customizations/` - Contains everything related to your theme. You are free to change everything you want in this directory.
- `framework-customizations/extension/` - Contains customizations for the framework extensions. You can overwrite options, views and configuration files of the extensions located in the framework directory. You can also store there theme extensions and create sub-extensions for extensions located in the framework.
- `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/config.php` - Theme configuration array, accessible through `fw()->theme->get_config('key');`.
- `framework-customizations/theme/static.php` - Enqueue all theme scripts and styles in this file. It is included automatically on `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.
- `framework-customizations/theme/posts.php` - Register theme post types and taxonomies in this file. It is included automatically on `init` action.
- `framework-customizations/theme/menus.php` - Register menus in this file. It is included automatically on `init` action.
- `framework-customizations/theme/hooks.php` - Add all theme's filters and actions to this file. This file is automatically included as early as possible, in this way your theme will not miss any action or filter execution.
- `framework-customizations/theme/helpers.php` - Add all helper functions and classes used all over your theme to this file.

- `framework-customizations/theme/manifest.php` - Contains an array with information about theme.
- `framework-customizations/theme/options/` - A directory containing option files only: post types options, taxonomy options, settings page options and you can also add custom options files in it. Check out the available *methods to retrieve the theme options*.
- `framework-customizations/theme/widgets/` - Contains theme widgets organized into directories. Widget class files are included automatically on `widgets_init` action.
- `framework-customizations/theme/includes/` - Contains any `.php` file that you want to be included automatically.

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 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.

Options

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. Here are the main places where options are used:

- **Settings Page**: Uses settings options located in `framework-customizations/theme/options/settings.php`
- **Post Add/Edit Page**: Uses post options located in `framework-customizations/theme/options/posts/{$post_type}.php`
- **Taxonomy Term Edit Page**: Uses taxonomy options located in `framework-customizations/theme/options/taxonomies/{$taxonomy}.php`

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(
    '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('id' => 'value')`. 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 parameters, but they are all optional except type. A better customized option will look like this:

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

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.

There are only three types of containers:

- box - WordPress metabox.
- tab - one tab (Tabs from same array level will be collected and generated as multiple tabs).
- group - group options into a wrapper div.

These types are built into the framework and new types of container options can't be defined. The simplest container option array looks something like this and will generate an empty metabox without title:

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

Accepted parameters:

- title *(string)* In box and tab this is used as title. In group it's not used
- attr *(array)* HTML attributes

Attention: These are all the parameters that the container options supports.

A better customized container option will look like this:

```
$options = array(
    array(
        'type'    => 'box',
        'title'   => __('Container Title', 'fw'),
        'attr'    => array('class' => 'custom-class', 'data-foo' => 'bar'),
        'options' => array(
            'id'  => array( 'type' => 'text' )
        )
    )
);
```

This will generate a box with a title and one option in it.

Important: Used in **Post Options** on the first array level, the `box` container accepts additional parameters:

- `'context' => 'normal|advanced|side'`
- `'priority' => 'default|high|core|low'`

These parameters are sent to add_meta_box() function.

Restrictions

Here are some restrictions to keep in mind:

- **Post Options** array on first level can have only `box` containers.
- `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.
- **Taxonomy Options** array on first level cannot have containers.

Note: There are no restrictions 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.

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%).

Note: Every option has its own width type.

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:

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

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

Tip: This is done to prevent css conflicts.

Javascript

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

```javascript
$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.

Built-in Option Types

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

- *Text*
- *Password*
- *Textarea*
- *Hidden*
- *HTML*
- *Checkbox*
- *Checkboxes*
- *Radio*
- *Select*

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);
    });
```

Text

Regular text input.

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

Password

Regular password input.

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

Textarea

Regular textarea.

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

Hidden

Simple hidden input.

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

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

HTML

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

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

```
array(
    'type'  => 'html',
    'value' => 'default hidden value',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    '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*.

Checkbox

Single checkbox.

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

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', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    'choices' => array(
        'choice-1' => __('Choice 1', 'fw'),
        'choice-2' => __('Choice 2', 'fw'),
        'choice-3' => __('Choice 3', 'fw'),
    ),
)
```

Radio

A list of radio buttons.

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

Select

Regular select.

```
array(
    'type'  => 'select',
    'value' => 'choice-3',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    'choices' => array(
        '' => '---',
        'choice-1' => __('Choice 1', 'fw'),
        'choice-2' => array(
            'text' => __('Choice 2', 'fw'),
            'attr' => array('data-foo' => 'bar'),
        ),
        array( // optgroup
            'attr'    => array('label' => __('Group 1', 'fw')),
            'choices' => array(
                'choice-3' => __('Choice 3', 'fw'),
                // ...
            ),
        ),
    ),
    /**
     * Allow not existing choices
     * Used when select is used in another option types and populated dynamically
     */
    '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', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    'choices' => array(
        '' => '---',
        'choice-1' => __('Choice 1', 'fw'),
        'choice-2' => array(
            'text' => __('Choice 2', 'fw'),
            'attr' => array('data-foo' => 'bar'),
        ),
        array( // optgroup
            'attr'    => array('label' => __('Group 1', 'fw')),
            'choices' => array(
                'choice-3' => __('Choice 3', 'fw'),
                // ...
            ),
        ),
    ),
)
```

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', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    /**
     * 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' => '',
    /**
     * An array with the available choices
     * Used only when 'population' => 'array'
     */
    'choices' => array(
        'choice-1' => __('Choice 1', 'fw'),
        'choice-2' => __('Choice 2', 'fw'),
        'choice-3' => __('Choice 3', 'fw'),
    ),
    /**
     * Set maximum items number that can be selected
     */
    'limit' => 100,
)
```

Multi

Option with another options in it.

```
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', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    '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.

Switch

Switch between two choices.

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

Custom Events

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

Color Picker

Pick a color.

```
array(
    'type'  => 'color-picker',
    'value' => '#FF0000',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
)
```

Gradient

Pick gradient colors.

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

Background Image

Choose background image.

```
array(
    'type'  => 'background-image',
    'value' => 'bg-1',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    'choices' => array(
        'none' => array(
            'icon' => get_template_directory_uri() . '/images/bg/bg-0.jpg',
            'css'  => array(
                'background-image' => 'none'
            ),
        ),
        'bg-1' => array(
            'icon'  => get_template_directory_uri() . '/images/bg/bg-1.jpg',
            'css'  => array(
                'background-image'  => 'url("' . get_template_directory_uri() . '/
↪images/bg-1.png' . '")',
                'background-repeat' => 'repeat',
            ),
        ),
        'bg-2' => array(
            'icon' => get_template_directory_uri() . '/images/bg/bg-2.jpg',
            'css'  => array(
                'background-image'  => 'url("' . get_template_directory_uri() . '/
↪images/bg-2.png' . '")',
                'background-repeat' => 'repeat-y'
            ),
        )
    )
)
```

Date Picker

Pick a date in calendar.

```
array(
    'type'  => 'date-picker',
    'value' => '',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    '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
)
```

Image Picker

Pick an image.

```
array(
    'type'  => 'image-picker',
    'value' => 'image-2',
    'attr'  => array(
        'class'    => 'custom-class',
        'data-foo' => 'bar',
    ),
    'label' => __('Label', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    '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.

Icon

Choose a FontAwesome icon.

```
array(
    'type'  => 'icon',
    'value' => 'fa-smile-o',
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
)
```

Upload

Single file upload.

```
array(
    'type'  => 'upload',
    'value' => array(
         'attachment_id' => '9',
         'url' => '//site.com/wp-content/uploads/2014/02/whatever.jpg'
    ),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    /**
     * 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,
)
```

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'
        ),
        array(
             'attachment_id' => '10',
             'url' => '//site.com/wp-content/uploads/2014/02/random.jpg'
        ),
    ),
    'attr'  => array( 'class' => 'custom-class', 'data-foo' => 'bar' ),
    'label' => __('Label', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    /**
     * 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,
)
```

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`.

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', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    'option' => array( 'type' => 'text' ),
)
```

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', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    '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
)
```

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.

Addable Popup

Addable popup with options.

```
array(
    'type' => 'addable-popup',
    'label' => __('Addable Popup', 'fw'),
    'desc'  => __('Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do␣
↪eiusmod tempor incididunt ut labore et dolore magna aliqua.', 'fw'),
    'template' => '{{- demo_text }}',
    'popup-options' => array(
        'demo_text' => array(
            'label' => __('Text', 'fw'),
            '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.', 'fw'),
            '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.', 'fw'),
                __('Sed ut perspiciatis, unde omnis iste natus error sit voluptatem␣
↪accusantium doloremque laudantium', 'fw')
            ),
        ),
        'demo_textarea' => array(
            'label' => __('Textarea', 'fw'),
            '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.', 'fw'),
            '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.', 'fw'),
                __('Sed ut perspiciatis, unde omnis iste natus error sit voluptatem␣
↪accusantium doloremque laudantium', 'fw')
            ),
        ),
    ),
)
```

Typography

Choose font family, size, style and color.

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

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', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
    /**
     * Load TinyMCE, can be used to pass settings directly to TinyMCE using an array
     * bool|array
     */
    'tinymce' => true,
    /**
     * Whether to display media insert/upload buttons
     * boolean
     */
    'media_buttons' => true,
    /**
     * Whether to output the minimal editor configuration used in PressThis
     * boolean
     */
    'teeny' => false,
    /**
     * Whether to use wpautop for adding in paragraphs
     * boolean
     */
    'wpautop' => true,
    /**
     * Additional CSS styling applied for both visual and HTML editors buttons, needs␣
↪to include <style> tags, can use "scoped"
     * string
     */
    'editor_css' => '',
    /**
     * If smething goes wrong try set to true
     * boolean
     */
    'reinit' => false,
)
```

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', 'fw'),
            'type'    => 'select',
            'choices' => array(
                'phone'  => __('Phone', 'fw'),
                'laptop' => __('Laptop', 'fw')
            ),
            'desc'    => __('Description', 'fw'),
            'help'    => __('Help tip', 'fw'),
        )
    ),
    /*
    'picker' => array(
        // '<custom-key>' => option
        'gadget' => array(
            'label'   => __('Choose device', 'fw'),
            'type'    => 'radio',
            'choices' => array(
                'phone'  => __('Phone', 'fw'),
                'laptop' => __('Laptop', 'fw')
            ),
            'desc'    => __('Description', 'fw'),
            'help'    => __('Help tip', 'fw'),
        )
    ),
    */
    /*
    'picker' => array(
        // '<custom-key>' => option
        'gadget' => array(
            'label'   => __('Choose device', 'fw'),
            'type'    => 'image-picker',
            'choices' => array(
                'phone'  => 'http://placekitten.com/70/70',
                'laptop' => 'http://placekitten.com/71/70'
            ),
            'desc'    => __('Description', 'fw'),
            'help'    => __('Help tip', 'fw'),
        )
    ),
    */
    /*
    picker => array(
        // '<custom-key>' => option
        'gadget' => array(
            'label' => __('Choose device', 'fw'),
            'type'  => 'switch',
            'right-choice' => array(
                'value' => 'laptop',
                'label' => __('Laptop', 'fw')
            ),
            'left-choice' => array(
                'value' => 'phone',
                'label' => __('Phone', 'fw')
            ),
            'desc' => __('Description', 'fw'),
            'help' => __('Help tip', 'fw'),
        )
    ),
    */
    'choices' => array(
        'phone' => array(
            'price' => array(
                'type'  => 'text',
                'label' => __('Price', 'fw'),
            ),
            'memory' => array(
                'type'  => 'select',
                'label' => __('Memory', 'fw'),
                'choices' => array(
                    '16' => __('16Gb', 'fw'),
                    '32' => __('32Gb', 'fw'),
                    '64' => __('64Gb', 'fw'),
                )
            )
        ),
        'laptop' => array(
            'price' => array(
                'type'  => 'text',
                'label' => __('Price', 'fw'),
            ),
            'webcam' => array(
                'type'  => 'switch',
                'label' => __('Webcam', 'fw'),
            )
        ),
    ),
    /**
     * (optional) if is true, the borders between choice options will be shown
     */
    'show_borders' => false,
)
```

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', 'fw'),
    'desc'  => __('Description', 'fw'),
    'help'  => __('Help tip', 'fw'),
)
```

Create Option Type

To define a new option type, create a class that extends the base option type class and register it.

```
class FW_Option_Type_New extends FW_Option_Type
{
    protected function get_type()
    {
        return 'new';
    }

    /**
     * @internal
     */
    protected function _enqueue_static($id, $option, $data)
    {
        wp_enqueue_style(
            'fw-option-'. $this->get_type(),
            fw_get_stylesheet_customizations_directory_uri(
                '/includes/options-types/'. $this->get_type() .'/static/css/styles.css
↪'
            )
        );

        wp_enqueue_script(
            'fw-option-'. $this->get_type(),
            fw_get_stylesheet_customizations_directory_uri(
                '/includes/options-types/'. $this->get_type() .'/static/js/scripts.js'
            ),
            array('fw-events', 'jquery')
        );
    }

    /**
     * @internal
     */
    protected function _render($id, $option, $data)
    {
        /**
         * $data['value'] contains value that should be used.
         * We decide if it's correct and how to use it in html
         */
        $option['attr']['value'] = (string)$data['value'];

        /**
         * $option['attr'] contains all attributes.
         *
         * Main (wrapper) option html element should have "id" and "class" attribute.
         *
         * All option types should have in main element the class "fw-option-type-{
↪$type}".
         * Every javascript and css in that option should use that class.
         *
         * Remaining attributes you can:
         *  1. use them all in main element (if option itself has no input elements)
         *  2. use them in input element (if option has input element that contains␣
↪option value)
         *
         * In this case we 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 class="button">'. __('Clear text', 'fw') .'<button/>';
        $html .= '</div>';

        return $html;
    }

    /**
     * @internal
     */
    protected function _get_value_from_input($option, $input_value)
    {
        /**
         * In this method we receive $input_value (from form submit or whatever)
         * and we should return correct and safe value that will be stored in␣
↪database.
         *
         * $input_value can be null.
         * In this case we should use default value from $option['value']
         */

        if (is_null($input_value)) {
            $input_value = $option['value'];
        }

        return (string)$input_value;
    }
    /**
     * @internal
     */
    protected function _get_defaults()
    {
        /**
         * These are default parameters that will be merged with option array.
         * They makes possible that any option has
         * only one required parameter array('type' => 'new').
         */

        return array(
            'value' => ''
        );
    }
}

FW_Option_Type::register('FW_Option_Type_New');
```

```
/**
 * Prefix (namespace) all css rules with ".fw-option-type-{$option_type}"
 * This guarantees that there will be no conflicts with other styles.
 */

.fw-option-type-new input {
    background-color: green;
    color: white;
}

.fw-option-type-new button {
    display: block;
}
```

```
jQuery(document).ready(function ($) {
    var optionTypeClass = '.fw-option-type-new';

    /**
     * Listen to special event that is triggered for uninitialized elements
     */
    fwEvents.on('fw:options:init', function (data) {
        /**
         * data.$elements are jQuery selected elements
         * that contains options html that needs to be initialized
         *
         * Find our uninitialized options by main class
         */
        var $options = data.$elements.find(optionTypeClass +':not(.initialized)');

        /**
         * Add some functionality to our options
         *
         * In this case, we will 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');
    });
});
```

Extensions

Introduction

Extensions are functionalities that add something new to framework or to another extension.

They can be placed in the flowing directories:

- `parent-theme/framework-customizations/extensions/`
- `child-theme/framework-customizations/extensions/`

Note: 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.

Directory Structure

The extensions directory has the following structure:

```
{extension-name}/
-[class-fw-extension-{extension-name}.php] # class FW_Extension_{Extension_Name}␣
↪extends FW_Extension { ... }
-config.php    # Extension specific configurations (should be described in extension
↪'s README.md)
-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     # Documentation
-manifest.php  # Data about extension: version, name, dependencies, etc.
-options/      # Files containing options
| -posts/      # Files containing post types options
| | -post.php
| | -{post-type}.php
| | -...
| -taxonomies/ # Files containing taxonomy tags options
| | -category.php
| | -post_tag.php
| | -{taxonomy}.php
| | -...
| -...
-views/
| -...
-static/
| -js/
| -css/
| -...
-includes/ # All .php files are auto included (no need to require_once)
| -other.php
| -...
--[extensions/] # Directory for sub extensions (if has)
```

Child Extensions

An extension can have child extensions that are located in `extensions/` directory within its own directory. A child extension can be located in the parent theme and child theme on the same relative path. Here are 3 possible cases where an extension can extists and where its child extensions can be placed:

1. If a `hello` extension is located in framework, child extensions can be placed in: framework, parent theme and child theme.

   ```
   -parent-theme/
   | -framework/
   | | -extensions/
   | |   -hello/
   | |     -extensions/
   | |       -hello-child/
   | |       -...
   | -framework-customizations/
   |   -extensions/
   |     -hello/
   |       -extensions/
   |         -hello-child/
   |         -...
   -child-theme/
     -framework-customizations/
       -extensions/
         -hello/
           -extensions/
             -hello-child/
             -...
   ```

2. If a `hello` extension is located in parent theme, child extensions can be placed in: parent theme and child theme.

   ```
   -parent-theme/
   | -framework/
   | -framework-customizations/
   |   -extensions/
   |     -hello/
   |       -extensions/
   |         -hello-child/
   |         -...
   -child-theme/
     -framework-customizations/
       -extensions/
         -hello/
           -extensions/
             -hello-child/
             -...
   ```

3. If a `hello` extension is located in child theme, child extensions can be placed only in child theme.

```
-parent-theme/
| -framework/
| -framework-customizations/
-child-theme/
  -framework-customizations/
    -extensions/
      -hello/
        -extensions/
          -hello-child/
          -...
```

Create Extension

To create an extension, just 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
{
    // ...
}
```

Cookbook

The Cookbook ia 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 Extension

An extension can be disabled by adding it to the blacklist. Blacklist is an array in theme config:

```php
// file: framework-customizations/theme/config.php

$cfg['extensions_blacklist'] = array('extension_name', 'another_extension_name');
```

Disable Child Extensions

Child extensions will not be activated if parent extension will `return false;` from `_init()`.

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

class FW_Extension_Example extends FW_Extension
{
    /**
     * @internal
     */
    protected function _init()
    {
        // ...

        if ($this->something_is_wrong()) {
            return false; // prevent child extensions activation
        }
    }
}
```

Validate Child Extensions

The parent extension has the possibility to check each child extension if it's valid or not. If the child extension is not valid, it will not be activated. To do that, the parent extension must overwrite the `_child_extension_is_valid()` method.

The method should return `true` if child extension is valid, and `false` if not.

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

class FW_Extension_Example extends FW_Extension
{
    /**
     * {@inheritdoc}
     */
    public function _child_extension_is_valid($child_extension_instance)
    {
        // force child extensions to extend some custom class, instead of FW_Extension
        return is_subclass_of($child_extension_instance, 'FW_Ext_Demo_Custom_Class');
    }

    // ...
}
```

Components

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.

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.

- `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('/static.php');

// prints '/.../wp-content/themes/scratch-theme/framework-customizations/
↪theme/static.php'
```

- `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_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');
```

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', 'fw'),
        'desc'  => __('Demo Description', 'fw'),
        'html'  => __('Demo Help Tip', 'fw'),
        '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);
```

Extensions

The core of *Extensions*.

- `get($extension_name)` - get instance of an existing active extension.

```
echo fw()->extensions->get('blog')->get_name();
```

- `locate_path($rel_path, ...)` - search full path of the file by given relative path. Will search in **child theme** then in **parent theme** then in **framework**. These are the places where an extension can be. By default will search in all places, you can specify only the places you want.

Hint: Check in `framework/core/extends/class-fw-extension.php` to see examples of how this method is used.

- `locate_path_URI($rel_path, ...)` - same as *locate_path()* but will return URI to that path instead of full path.

Helpers

Introduction

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

PHP Helpers

General

General PHP helpers:

- `fw_print($value, $die = false)` - styled version of `print_r()`.
- `fw_html_tag($tag, $attr = null, $end = null)` - generate html tag.

  ```
  echo fw_html_tag('script', array('src' => '/demo.js'), true);

  // <script src="/demo.js"></script>
  ```

- `fw_attr_to_html(array $attr_array)` - generate html attributes from array.

  ```
  echo '<div '. fw_attr_to_html(array('id' => 'foo', 'class' => 'bar')) .'>
  ↪</div>';

  // <div id="foo" class="bar" ></div>
  ```

- `fw_akg($keys, &$array_or_object, $default_value = null, $keys_delimiter = '/')` - get array multikey value.

  Note: **MultiKey** is a string composed from multiple array keys, separated by a delimiter character, that represents an array structure. For example

  ```
  'a/b/c'
  ```

  represents

  ```
  array(
      'a' => array(
          'b' => array(
              'c' => null
          )
      )
  )
  ```

```
$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.

```
$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.

```
$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.

```
echo fw_unique_increment(), PHP_EOL;
echo fw_unique_increment(), PHP_EOL;
echo fw_unique_increment(), PHP_EOL;

/*
9370
```

```
9371
9372
*/
```

- `fw_stripslashes_deep_keys($value)` - strip slashes (recursive) from values and keys (if value is array) if `magic_quotes_gpc = On`.
- `fw_addslashes_deep_keys($value)` - add slashes (recursive) to values and keys (if value is array) if `magic_quotes_gpc = On`.
- `fw_current_screen_match($rules)` - check if current `global $current_screen;` (available in admin side) matches the given rules. Used to detect on which admin page you currently are. Thus you can for example enqueue a script only on a target page, not on all admin pages.

```
/**
 * @internal
 */
function _action_enqueue_demo_admin_scripts() {
    // To find out what is the current screen of the current page,␣
↪uncomment next line
    //global $current_screen; fw_print($current_screen);

    $only = array(
        'only' => array(
            array( 'id' => 'dashboard' )
        )
    );

    if (fw_current_screen_match($only)) {
        // enqueue this script only on dashboard page
        wp_enqueue_script(
            'demo-dashboard',
            fw_get_stylesheet_customizations_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',
            fw_get_stylesheet_customizations_directory_uri('/js/demo-
↪excluded.js')
        );
    }
}
add_action('admin_enqueue_scripts', '_action_enqueue_demo_admin_scripts');
```

Note: You can combine `only` and `exclude` in the same rules array.

- `fw_locate_theme_path_uri($rel_path)` - search by relative path, in child then in parent theme directory, and return URI.

```
echo fw_locate_theme_path_uri('/styles.css');

// http://your-site.com/wp-content/themes/child-theme/style.css
```

- `fw_locate_theme_path($rel_path)` - search by relative path, in child then in parent theme directory, and return full path.

```
echo fw_locate_theme_path('/styles.css');

// /var/www/wordpress/public_html/wp-content/themes/child-theme/style.css
```

- `fw_render_view($file_path, $view_variables = array())` - safe render view and return html. In view will be accessible only passed variables, not current context variables.

```
$private = 'Top Secret';

echo fw_render_view(
    get_stylesheet_directory() .'/demo-view.php',
    array('message' => 'Hello')
);

/* demo-view.php
<?php if (!defined('FW')) die('Forbidden');

echo $message;
echo $private;
*/

// Hello
// Notice: Undefined variable: private
```

- `fw_get_variables_from_file($file_path, array $variables)` - extract specified variables from file.

```
$variables = fw_get_variables_from_file(
    get_stylesheet_directory() .'/demo-variables.php',
    array(
        'message' => 'Hi',
        'foo' => 'bar'
    )
);

/* demo-variables.php
<?php if (!defined('FW')) die('Forbidden');

$message = 'Hello';
*/

print_r($variables);

/*
```

```
array(
    'message' => 'Hello',
    'foo' => 'bar'
)
*/
```

- `fw_include_file_isolated($file_path)` - include files isolated and don't give access to current context variables.

```
$private = 'Top Secret';

fw_include_file_isolated(get_stylesheet_directory() .'/demo-isolated.php
↪');

/* demo-isolated.php
<?php if (!defined('FW')) die('Forbidden');

echo $private;
*/

// Notice: Undefined variable: private
```

- `fw_html_attr_name_to_array_multi_key($attr_name)` - convert html `name` attribute to *multi-key*.

```
echo fw_html_attr_name_to_array_multi_key('a[b][c]');

// 'a/b/c'
```

- `fw_is_real_post_save()` - used in 'save_post' action to detect if it's a real post save, not a revision, auto save or something else.
- `fw_current_url()` - generate current page url from `$_SERVER` data.
- `fw_is_valid_domain_name($domain_name)` - check if a domain name is valid.
- `fw_htmlspecialchars($string)` - UTF-8 version of php's `htmlspecialchars()`. Just a shorthand not to write two more parameters for default `htmlspecialchars()` every time.

Note: In php 5.2 `htmlspecialchars()` default encoding is not UTF-8.

- `fw_human_time($seconds)` - convert seconds to human readable time.

```
echo fw_human_time(12345);

// '3 hours'
```

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

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

// 13
// 7
```

- `fw_is_post_edit()` - check if you are currently on a post edit page. It also detects if form submit was made from the post edit page.

- `fw_dirname_to_classname($dirname)` - convert directory name to string to be used as/in class name.

```
echo 'FW_'. fw_dirname_to_classname('hello-world');

// FW_Hello_World
```

- `fw_fix_path($path)` - make sure a path is in unix style, with `/` directory separators.
- `fw_get_stylesheet_customizations_directory()` - Full path to the child-theme/framework-customizations directory.
- `fw_get_stylesheet_customizations_directory_uri()` - URI to the child-theme/framework-customizations directory.
- `fw_get_template_customizations_directory()` - Full path to the parent-theme/framework-customizations directory.
- `fw_get_template_customizations_directory_uri()` - URI to the parent-theme/framework-customizations directory.
- `fw_get_framework_directory()` - Full path to the parent-theme/framework directory.
- `fw_get_framework_directory_uri()` - URI to the parent-theme/framework directory

Cache

Use cache to store frequently accessed data. Cache is just a big array and has one useful feature: it will automatically begin to unset array keys if the php memory is close to full. So it is safe to store in it as much data as you want (of course the maximum allowed by php, by default is ~100Mb).

```
function get_foo_bar() {
    $cache_key = 'foo/bar';

    try {
        /**
         * This will throw an exception if the key was not found
         */
        return FW_Cache::get($cache_key);
    } catch (FW_Cache_Not_Found_Exception $e) {
        $data = _generate_foo_bar_data();

        FW_Cache::set($cache_key, $data);

        return $data;
    }
}
```

Warning: Don't do this:

```
    ...
} catch (FW_Cache_Not_Found_Exception $e) {
    FW_Cache::set($cache_key, _generate_foo_bar_data());

    return FW_Cache::get($cache_key);
}
```

because `FW_Cache::set(...)` can fail or the data that was set can be removed after automatically memory free.

Options

Functions for working with options:

- `fw_extract_only_options(array $options)` - extract only regular options from any array of options.

```
$options = array(
    array(
        'type' => 'box',
        'options' => array(
            'demo-1' => array(
                'type' => 'text'
            )
        )
    ),
    array(
        'type' => 'box',
        'options' => array(
            array(
                'type' => 'tab',
                'options' => array(
                    'demo-2' => array(
                        'type' => 'textarea'
                    )
                )
            )
        )
    )
);

print_r( fw_extract_only_options($options) );

/*
array(
    'demo-1' => array(
        'type' => 'text'
    ),
    'demo-2' => array(
        'type' => 'textarea'
    )
)
*/
```

- `fw_get_options_values_from_input(array $options, $input_array = null)` - extract options values from input array. If no input array is provided, values from `$_POST` will be used.

```
$options = array(
    'demo-1' => array( 'type' => 'text', 'value' => 'default value 1' ),
    'demo-2' => array( 'type' => 'text', 'value' => 'default value 2' ),
);

$input_values = array(
    'demo-1' => 'input value 1',
    'demo-3' => 'input value 3',
);

$values = fw_get_options_values_from_input($options, $input_values);
```

```
print_r($values);

/*
array(
    'demo-1' => 'input value 1',
    'demo-2' => 'default value 2',
)
*/
```

- `fw_prepare_option_value($value)` - by default WordPress offers filters for other plugins to alter database options and post meta. For ex translation plugins use these filters to translate things. If you save your options values in a custom place (like framework does by default, by saving options in a serialized array in database options and post meta) the WordPress filter doesn't know how to work with them.

  Tip: Use this function to pass an option value through filters and translation features that simulates WordPress default behavior. This function is already used in core so you don't have to bother about passing options values through it each time. Use it if you will do something custom and strings will not be translated.

Database

Functions and classes for working with database:

- `FW_WP_Option` - alternative to WordPress **get_option()** and **update_option()** functions.

  Note: Features:

  – Supports *multikeys*

  – The value is stored in two formats: original and prepared. Prepared is used for frontend because it is translated.

- `FW_WP_Post_Meta` - alternative to WordPress **get_post_meta()** and **update_post_meta()**.

  Note: Features:

  – Supports *multikeys*

  – The value is stored in two formats: original and prepared. Prepared is used for frontend because it is translated.

- `fw_get_db_settings_option($option_id, $default_value = null)` - get value from the database of an option from the framework's settings page. Settings options are located in `framework-customizations/theme/options/settings.php`.
- `fw_set_db_settings_option($option_id, $value)` - set a value in the database for an option from the framework's settings 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_extension_data($extension_name, $key, $default_value = null)` - get a value from the database of some data stored by some extension.
- `fw_set_db_extension_data($extension_name, $key, $value)` - extensions uses this function to store custom values in the database.

> **Important:** All values are stored in one wp option. This prevents database spam with wp options for each extension.

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', 'fw');
    }

    return $errors;
}

function _my_form_save() {
    $input_value = FW_Request('demo');

    // do something with value
}
```

```
echo $my_form->render();
// this will output:
// <form ... ><input type="text" name="demo" maxlength="10" value=""></form>
```

JavaScript Helpers

Useful javascript functions and classes. The main helper is `fw`, an object containing constants, methods and classes. To use these helpers, add `fw` to your script dependencies:

```
wp_register_script(..., ..., array('fw'));
```

General

General javaScript helpers:

- `fw.FW_URI` - URI to the framework directory.
- `fw.SITE_URI` - URI to the site root directory.
- `fw.intval(value)` - alternative to php intval(). Returns `0` on failure, instead of `NaN` like parseInt() does.
- `fw.md5(string)` - calculate md5 hash of the string.
- `fw.loading` - show loading on the page.

  Tip: Useful when doing AJAX requests and want to inform your users about that.

  ```
  fw.loading.show();

  setTimeout(function(){
      fw.loading.hide();
  }, 3000);
  ```

  The `show()` and `hide()` methods can be called multiple times. If `show()` is called 10 times, then `hide()` should be called 10 times for loading to disappear. This is done for cases when this helper is used by multiple asynchronous scripts, the loading should not disappear until all scripts complete the work.

- `fw.capitalizeFirstLetter(text)` - capitalizes the first letter of a string.
- `fw.ops(properties, value, obj, delimiter)` - same as `fw_aks(...)` from *PHP Helpers*, but for javascript objects.

  ```
  var obj = {foo: {}};

  fw.ops('foo/bar', 'demo', obj);

  console.log(obj); // {foo: {bar: 'demo'}}
  ```

```
```

- `fw.opg(properties, obj, defaultValue, delimiter)` - same as `fw_akg(...)` from *PHP Helpers*, but for javascript objects.

  ```
  var obj = {foo: {bar: 'hello'}};

  console.log( fw.opg('foo/bar', obj) ); // 'hello'
  ```

- `fw.randomMD5()` - generate random md5.

Options Modal

Modal with *options*. Display html generated from a given options array. After the user completes the form and presses "Save", values are available as a javascript object.

```
var modal = new fw.OptionsModal({
    title: 'Custom Title',
    options: [
        {'test_1': {
            'type': 'text',
            'label': 'Test1'
        }},
        {'test_2': {
            'type': 'textarea',
            'label': 'Test2'
        }}
    ],
    values: {
        'test_1': 'Default 1',
        'test_2': 'Default 2'
    },
    size: 'small' // 'medium', 'large'
});

// listen for values change
modal.on('change:values', function(modal, values) {
    console.log(values);
});

// replace values
modal.set('values', {
    'test_1': 'Custom 1',
    'test_2': 'Custom 2'
});

modal.open();
```

Note: Make sure to enqueue scripts and styles for the options you use in modal. Usually it is done before page is displayed.

```
fw()->backend->enqueue_options_static($modal_options);
```

Events

`fwEvents` is a global object on which you can trigger or listen custom events. This way different scripts can communicate with each other.

```
// script-1.js

fwEvents.on('script-2:message', function(data){
    console.log('script-1 received a message from script-2: '+ data.message);
});

// script-2.js

fwEvents.trigger('script-2:message', {message: 'Hello World!'});
```

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`.

Manifest

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.

Framework

The framework's manifest is located in `framework/manifest.php` and can be accessed like this:

```
fw()->manifest->get('version');
```

It supports the following parameters:

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

$manifest = array();

$manifest['name']          = __('Framework', 'fw');
$manifest['uri']           = 'http://themefuse.com/framework';
$manifest['description']   = __('WordPress Framework', 'fw');
$manifest['version']       = '1.0';
$manifest['author']        = 'ThemeFuse';
$manifest['author_uri']    = 'http://themefuse.com/';
$manifest['requirements'] = array(
    'wordpress' => array(
        'min_version' => '4.0',
        /*'max_version' => '4.99.9'*/
    ),
);
```

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();

$manifest['id']           = 'theme-id';
$manifest['name']         = __('Theme Title', 'fw');
$manifest['uri']          = 'http://themefuse.com/wp-themes-shop/theme-name';
$manifest['description']  = __('Another awesome wordpress theme', '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'*/
    ),
    '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'
        ),*/
    )
);
```

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', 'fw');
$manifest['uri']          = 'http://extension-homepage.com/';
$manifest['description']  = __('Another awesome framework extension', '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'*/
    ),
    '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'
        ),*/
    )
);
```

Built-in Extensions

Introduction

The Unyson framework comes with the following built-in extensions:

- *Shortcodes*
- *Slider*
- *Mega Menu*
- *Sidebars*
- *Styling*
- *Breadcrumbs*
- *SEO*
- *Blog Posts*
- *Backup*
- *Portfolio*
- *Feedback*

Shortcodes

The shortcodes extension makes possible the easy creation of WordPress Shortcodes and their integration with the framework's layout builder.

Directory structure

A shortcode can be created in the following places:

- `framework-customizations/theme/shortcodes/`
- `{some-extension}/shortcodes/`

```
{shortcode-name}/
-class-fw-shortcode-{shortcode-name}.php # optional
-config.php
-options.php # optional
-static/ # optional
| -css/
| | -auto-enqueued-style.css
| | -...
| -img/
| | -layout_builder.png
| | -...
| -js/
|   -auto-enqueued-script.js
|   -...
-views/
  -view.php
  -...
```

The framework will register a new WordPress shortcode with its tag being the shortcode directory name, with hyphens replaced by underscores (`[shortcode_name]` for the above example).

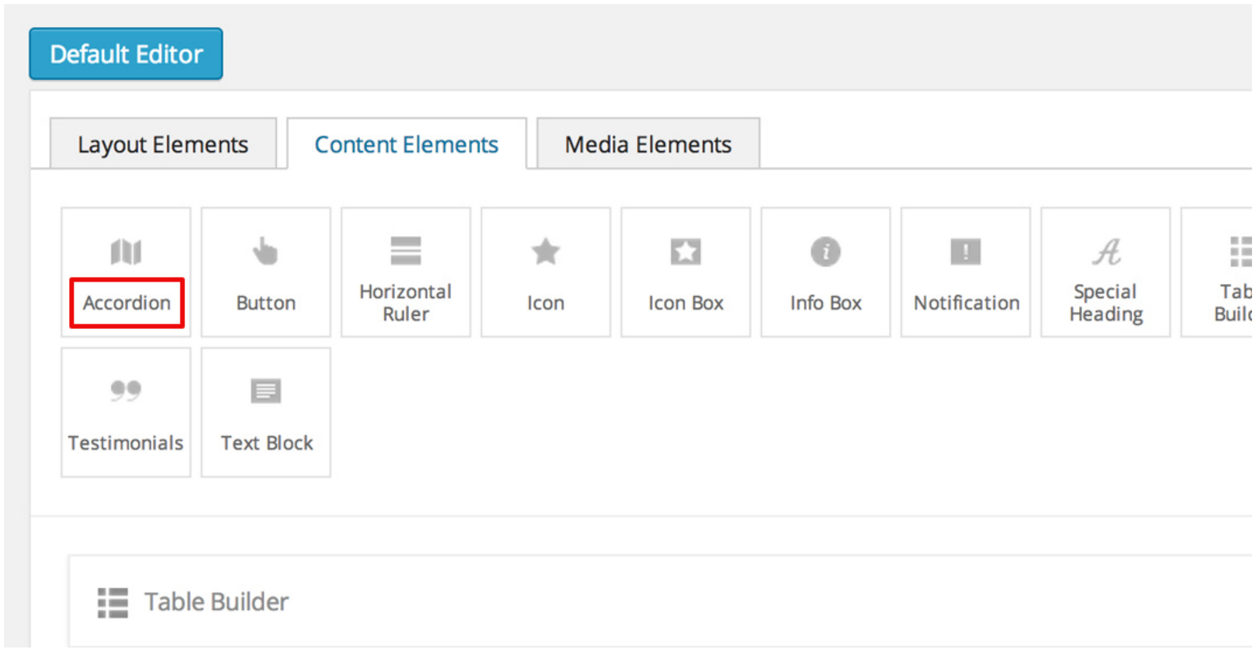
Config File

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:

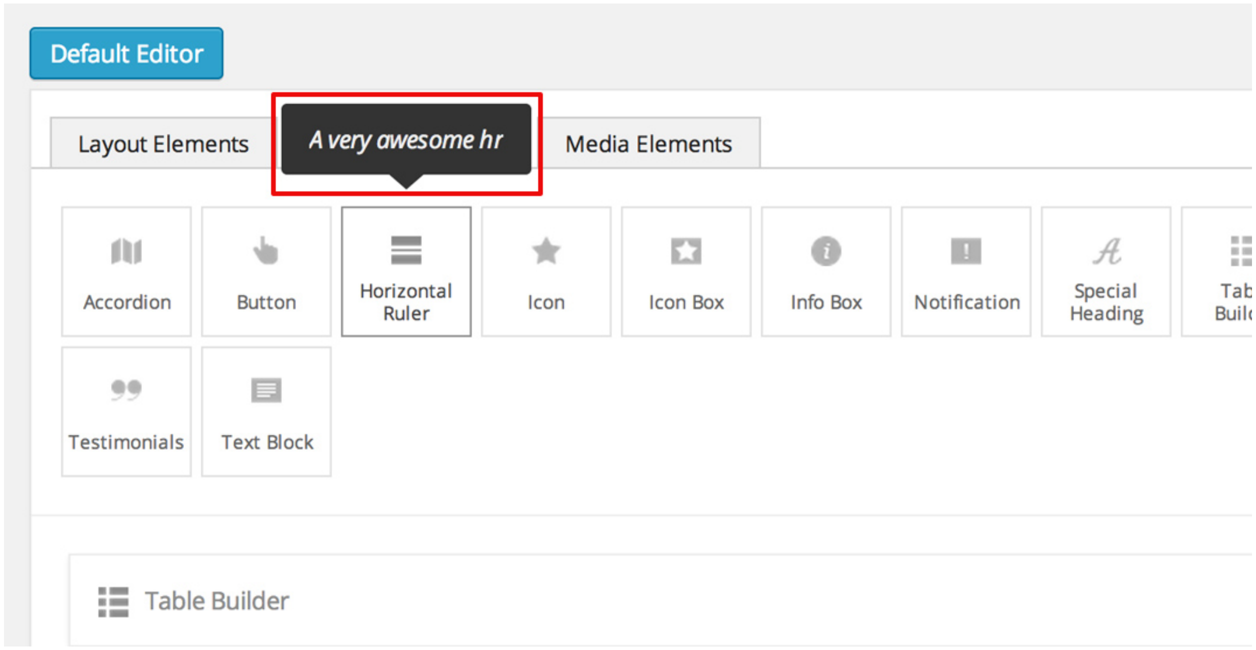
```
$cfg = array(
    'layout_builder' => array(
        'title'         => __('Demo Shortcode', 'fw'),
        'description'   => __('Demo shortcode description', 'fw'),
        'tab'           => __('Demo Elements', 'fw'),
        'popup_size'    => 'small' // can be 'large', 'medium' or 'small'
    )
);
```

For the shortcode to appear in the layout builder, the config array contains a special `layout_builder` key that holds an array with the following data:

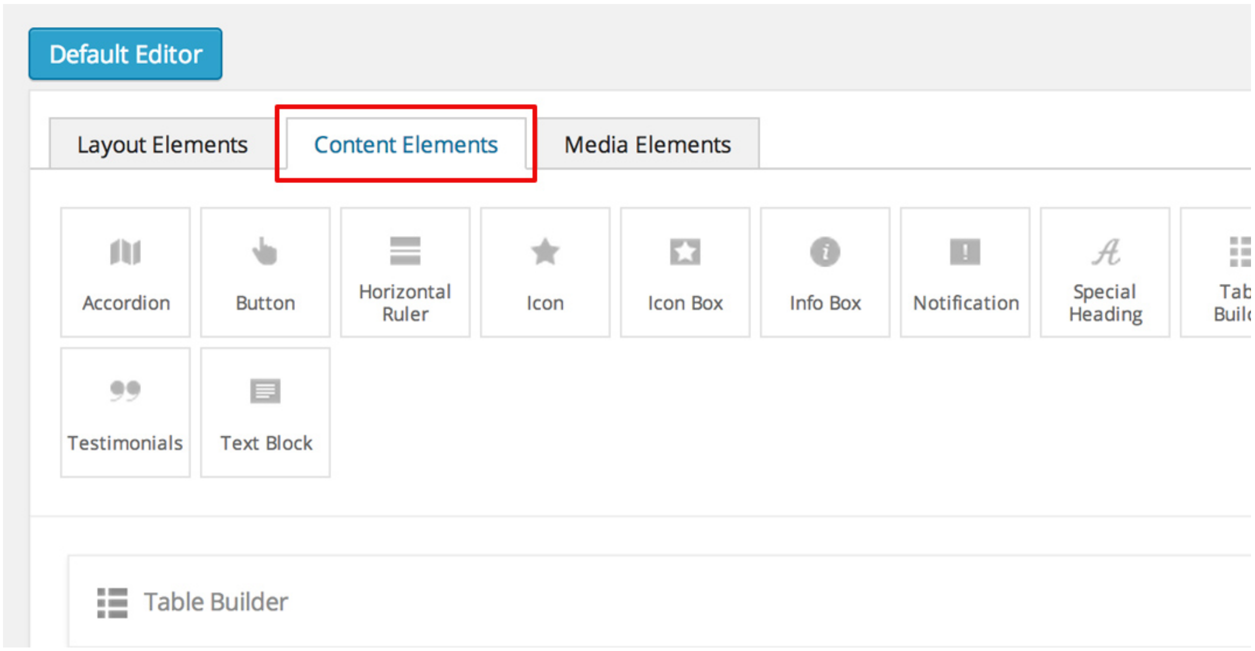
- `title` - the title that will appear in the shortcode box.



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

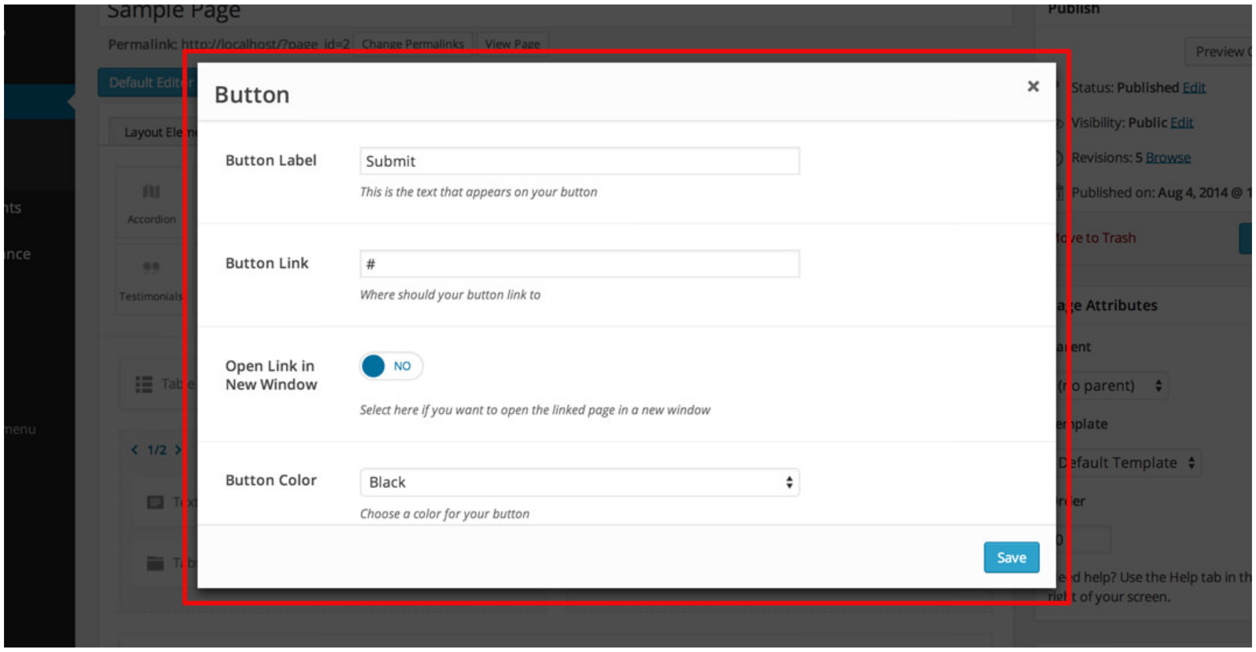


• `tab` - the builder tab in which the shortcode box will appear.



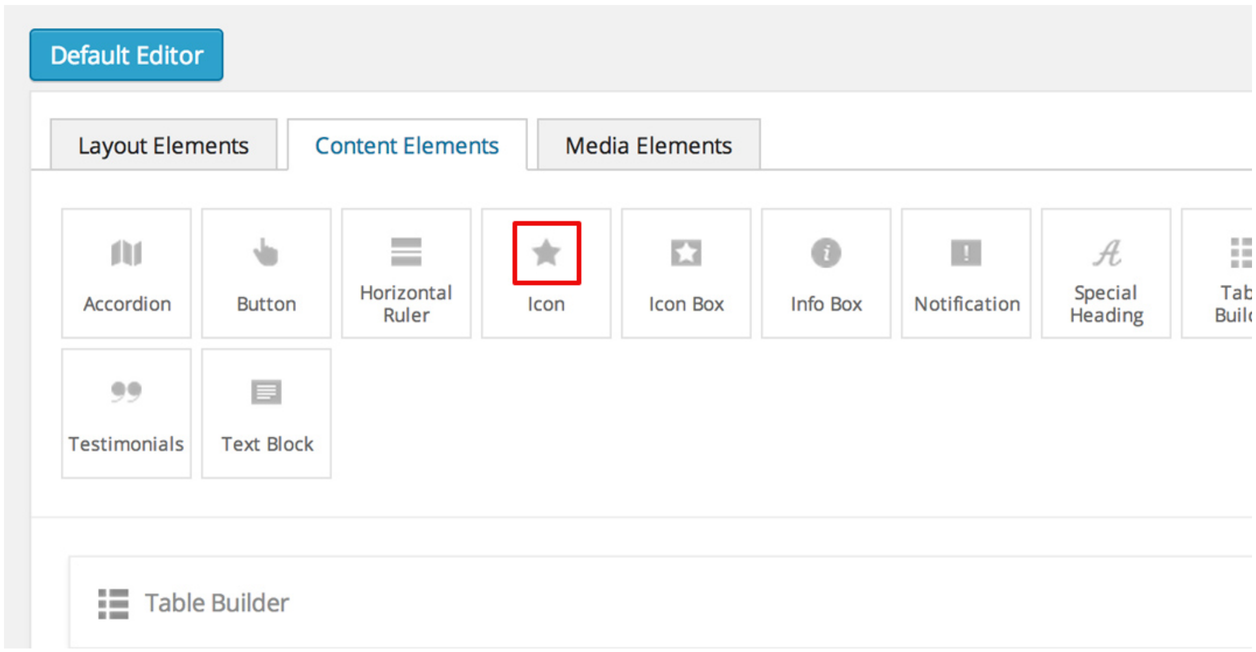
- `popup_size` - the size of the popup in which the *shortcode options* will be displayed.

  Allowed values are `large | medium | small`. This parameter is optional and the default is set to `medium`.


Sample Page
Button
Button Label
Submit
This is the text that appears on your button
Button Link
#
Where should your button link to
Open Link in New Window
NO
Select here if you want to open the linked page in a new window
Button Color
Black
Choose a color for your button
Save

Builder icon

To set an icon for the shortcode box, put an image named `layout_builder.png` inside `{your-shortcode}/static/img/` directory. The image should have the size of 16x16 px.

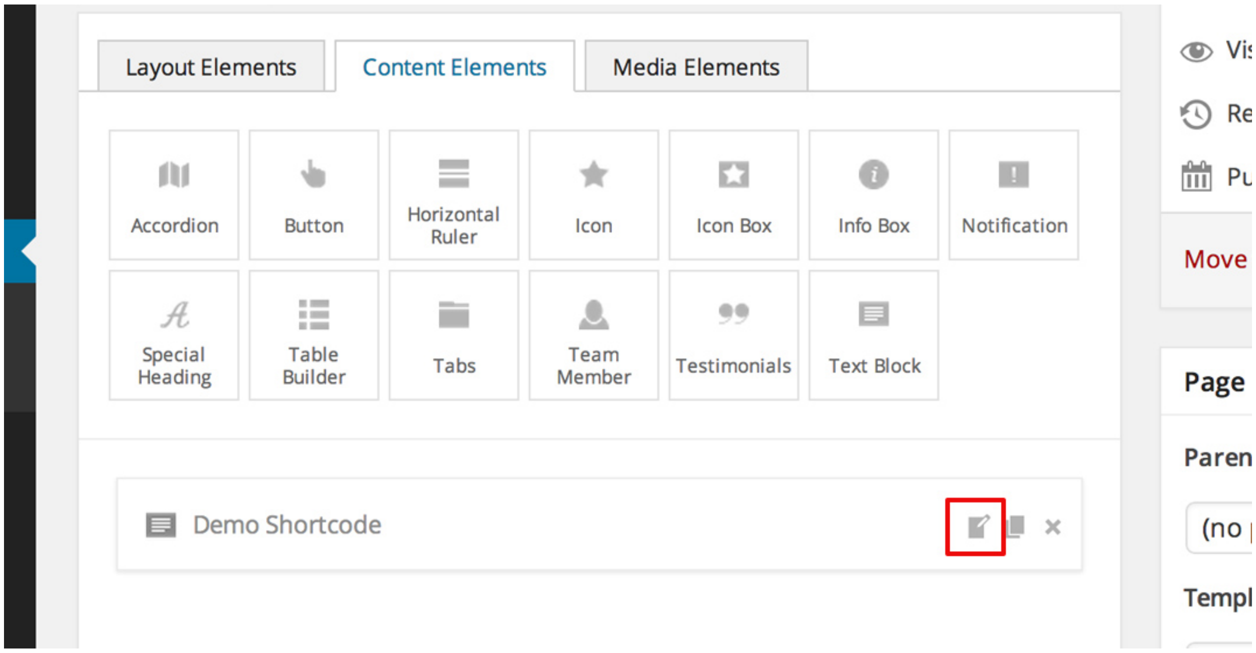


Options file

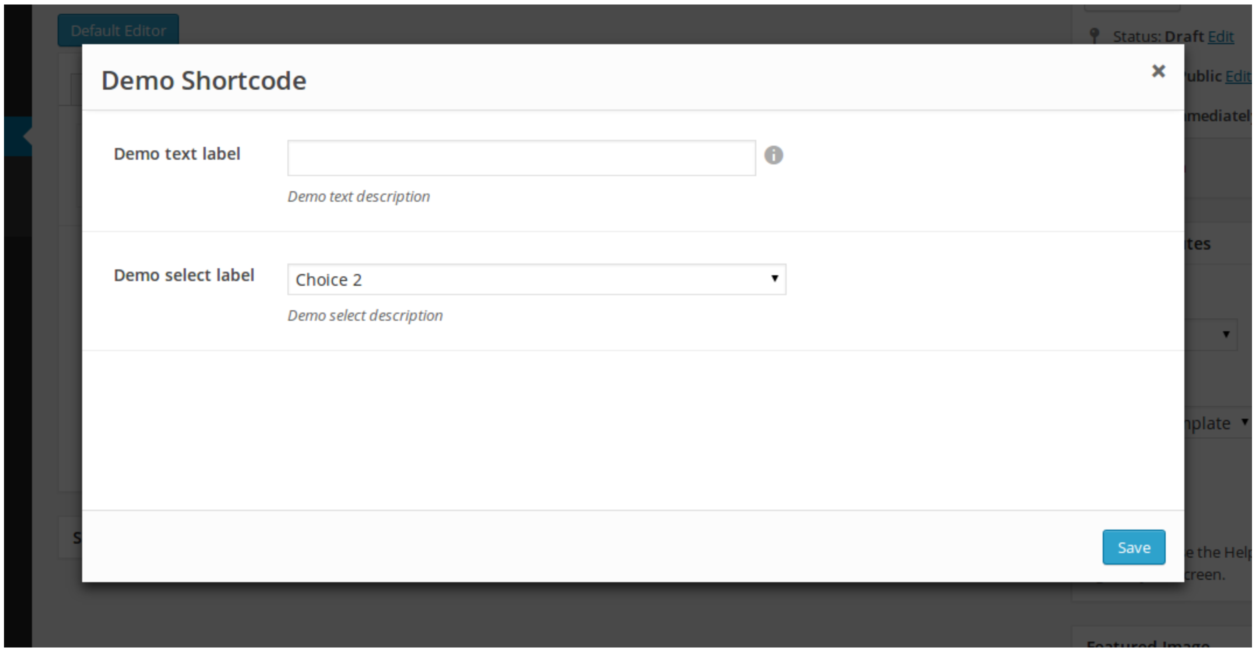
The shortcode directory can contain a file named `options.php` with correctly formed *options*:

```php
$options = array(
    'demo_text'   => array(
        'label'   => __('Demo text label', 'fw'),
        'desc'    => __('Demo text description', 'fw'),
        'help'    => __('Demo text help', 'fw'),
        'type'    => 'text'
    ),
    'demo_select' => array(
        'label'   => __('Demo select label', 'fw'),
        'desc'    => __('Demo select description', 'fw'),
        'type'    => 'select',
        'choices' => array(
            'c1'  => __('Choice 1', 'fw'),
            'c2'  => __('Choice 2', 'fw'),
            'c3'  => __('Choice 3', 'fw')
        ),
        '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:



Demo Shortcode

Demo text label

Demo text description

Demo select label: Choice 2

Demo select description

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 files

When rendering the *default view*, the framework will enqueue all `css` files from `{your-shortcode}/static/css/` and all `js` files from `{your-shortcode}/static/js/` directories in alphabetical order.

Note: Files from subdirectories inside both `{your-shortcode}/static/css/` and `{your-shortcode}/static/js/` will not be enqueued by default.

Check out the *cookbook section* for tips on how to do that.

Attention: All of the above is valid only in the case that the `handle_shortcode()` method from the *class file* was not overriden.

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. However, some of that default functionality can be overridden 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_path()` - returns the path to the shortcode folder. Useful for loading views or checking if files exist.
- `get_uri()` - returns the uri to the shortcode folder. Useful for enqueuing styles and scripts, or forming the `src` attribute of an `<img>` tag for an image from `static/img/`.

- `get_config($key = null)` - returns the shortcode's whole config array, or just a specified key value.
- `get_options()` - returns the shortcode's options array, if there is any.

The methods that are most prone to be overriden 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.).
- `handle_shortcode($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

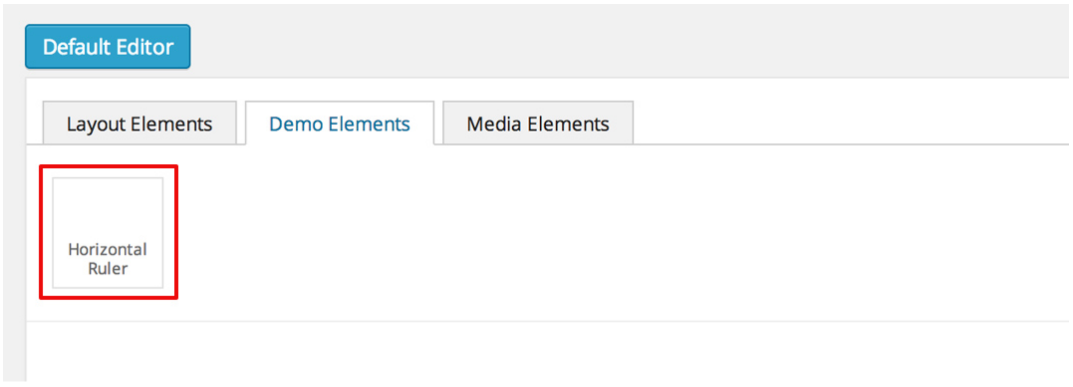
Creating a simple shortcode

This example will go through creating the `[hr]` (horizontal ruler) shortcode in a few simple steps:

1. Create a `hr` folder in `framework-customizations/theme/shortcodes/`.
2. Create a *config file* inside `framework-customizations/theme/shortcodes/`:

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

$cfg = array(
    'layout_builder' => array(
        'title'       => __('Horizontal Ruler', 'fw'),
        'description' => __('Creates a \'hr\' html tag', 'fw'),
        'tab'         => __('Demo Elements', 'fw'),
    )
);
```

Note: At this point the shortcode should appear in the **Demo Elements** tab of the layout builder as shown bellow:



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

3. Create the *view file* in `framework-customizations/theme/shortcodes/hr/views/`:

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

<hr>
```

The [hr] shorcode is completed! The directory structure of the shortcode is as shown bellow:

```
framework-customizations/
-theme/
  -shortcodes/
    -hr/
      -config.php
      -views/
        -view.php
```

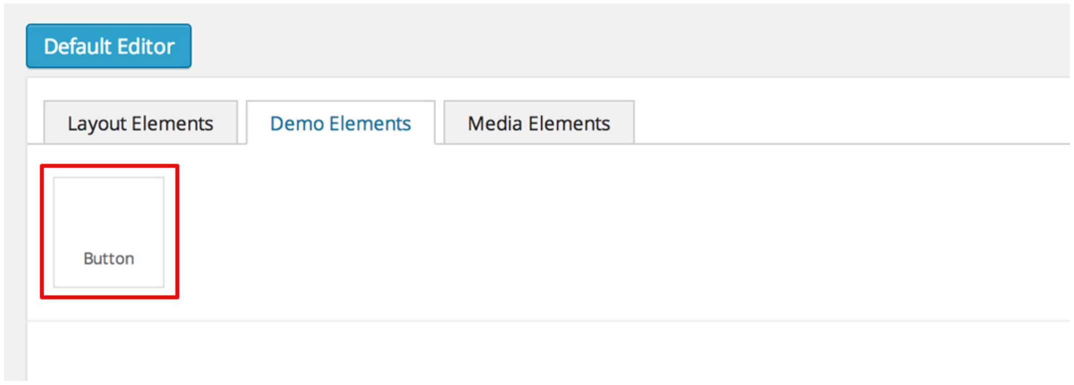
Creating a shortcode with options

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

1. Create a button folder in framework-customizations/theme/shortcodes/
2. Create a *config file* inside framework-customizations/theme/button/:

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

$cfg = array(
    'layout_builder' => array(
        'title'          => __('Button', 'fw'),
        'description'    => __('Creates a button with choosable label,␣
↪size and style', 'fw'),
        'tab'            => __('Demo Elements', 'fw'),
    )
);
```

Note: At this point the shortcode should appear in the **Demo Elements** tab of the layout builder as shown bellow:

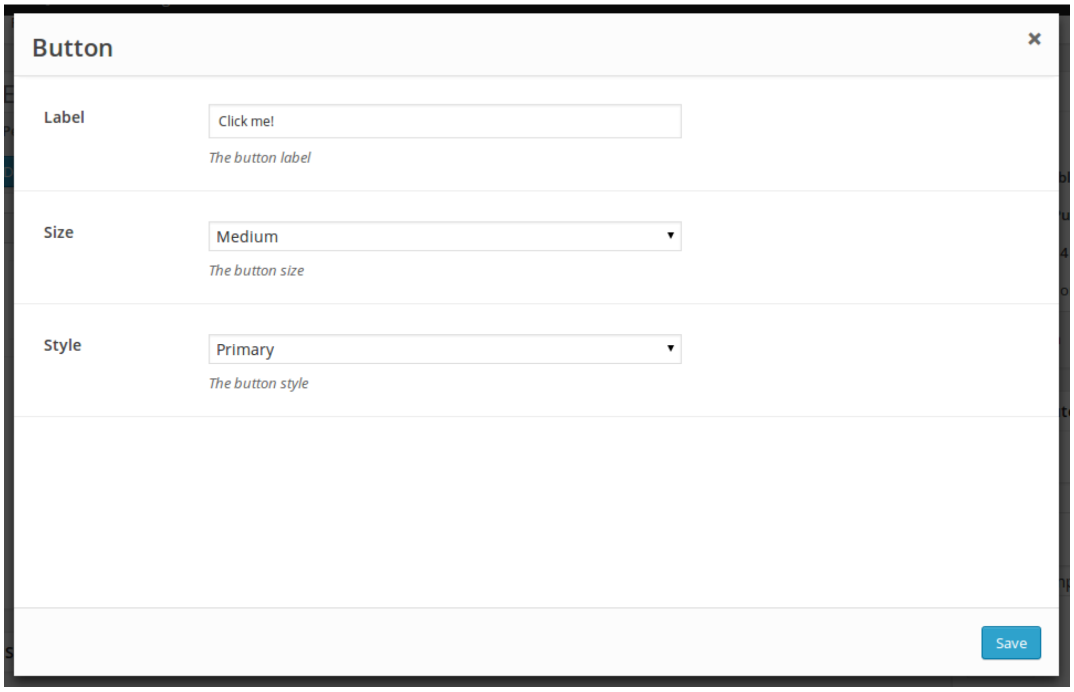


Tip: To add an icon to the shortcode see the *icon section*.

3. Create an *options file* inside framework-customizations/theme/shortcodes/button/ with the options for **label**, **size** and **style**:

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

$options = array(
    'label' => array(
        'label'   => __('Label', 'fw'),
        'desc'    => __('The button label', 'fw'),
        'type'    => 'text',
        'value'   => __('Click me!', 'fw')
    ),
    'size' => array(
        'label'   => __('Size', 'fw'),
        'desc'    => __('The button size', 'fw'),
        'type'    => 'select',
        'choices' => array(
            'big'    => __('Big', 'fw'),
            'medium' => __('Medium', 'fw'),
            'small'  => __('Small', 'fw')
        ),
        'value'   => 'medium'
    ),
    'style' => array(
        'label'   => __('Style', 'fw'),
        'desc'    => __('The button style', 'fw'),
        'type'    => 'select',
        'choices' => array(
            'primary'   => __('Primary', 'fw'),
            'secondary' => __('Secondary', 'fw')
        )
    )
);
```

Now, when clicking the shortcode inside the canvas area of the layout builder a pop-up window containting the options will appear:



Button

Label: Click me!
The button label

Size: Medium
The button size

Style: Primary
The button style

4. Create the *view file* in `framework-customizations/theme/shortcodes/button/views/`. 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 bellow:

```
framework-customizations/
-theme/
  -shortcodes/
    -button/
      -config.php
      -options.php
      -views/
        -view.php
```

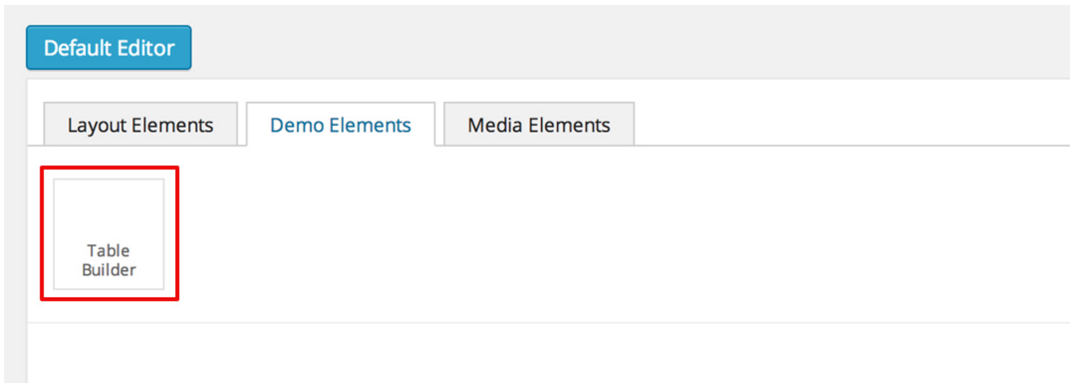
Creating an advanced shortcode with a custom class

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/theme/shortcodes/`.
2. Create *a config file* inside `framework-customizations/theme/table-builder/`:

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

$cfg = array(
    'layout_builder' => array(
        'title'       => __('Table Builder', 'fw'),
        'description' => __('Creates custom tables', 'fw'),
        'tab'         => __('Demo Elements', 'fw'),
        'popup_size'  => 'large'
    )
);
```

Note: At this point the shortcode should appear in the **Demo Elements** tab of the layout builder as shown bellow:



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 its needs.

   (a) Create a `table-builder` option type in `framework-customizations/theme/shortcodes/table-builder/includes/fw-option-type-table-builder/`

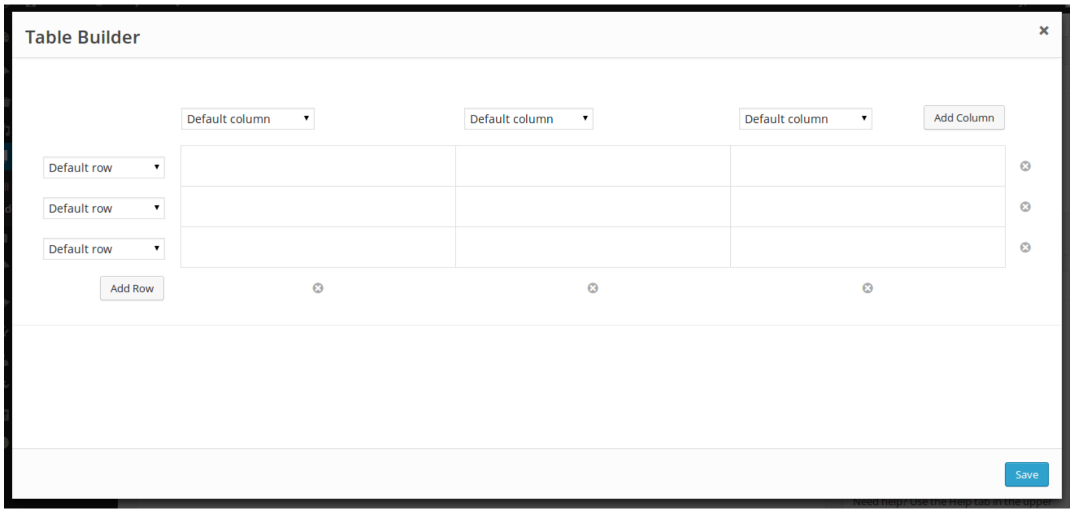
   (b) Create a *custom class* for the shortcode and override the `_init()` method, to load the custom option type class file.

```
<?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->get_path() . '/includes/fw-option-type-
↪table-builder/class-fw-option-type-table-builder.php';
    }

    // ...

}
```

   (c) Create an *options file* inside `framework-customizations/theme/shortcodes/table-builder/` 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* in `framework-customizations/theme/shortcodes/table-builder/views/` and make use of the custom option type value.

The `[table_builder]` shorcode is completed! The directory structure of the shortcode is as shown bellow:

```
framework-customizations/
-theme/
  -shortcodes/
    -table-builder/
      -class-fw-shortcode-table-builder.php
      -config.php
      -options.php
      -views/
      | -view.php
      -includes/
        -fw-option-type-table-builder/
          -class-fw-option-type-table-builder.php
          -static/
          -views/
```

Slider

The slider extension allows you to create all kinds of sliders for your website.

Directory Structure

The slider extension directory has the following structure:

```
slider/
-...
-extensions/
  -{slider-type}/
  -...
  -{slider-type}/
    -class-fw-extension-{slider-type}.php # optional
    -config.php
    -options/ # optional
    | -options.php
    | -...
    -static/ # optional
    | -css/
    | | -auto-enqueued-style.css
    | | -...
    | -img/
    | | -preview.jpg
    | | -thumb.jpg
    | | -...
    | -js/
    |   -auto-enqueued-script.js
    |   -...
    -views/
      -{slider-type}.php
      -...
```

Create a simple slider type

To create simple slider type, create a *child extension*. In our case the slider type is `bx-slider`, so the child extension directory will be `framework-customizations/extensions/slider/extensions/bx-slider`.

Configuration

The configuration file `config.php` contains the following parameters:

```
/**
 * Specify available population methods.
 *
 * There are 4 built-in population methods:
 *  'posts'      Populate with posts
 *  'categories' Populate with categories
 *  'tags'       Populate with tags
 *  'custom'     Populate with custom content provided by the user
 */
$cfg['population_methods'] = array('posts', 'categories', 'tags', 'custom');

/**
 * Specify what media types the slider supports.
 *
 * There are 2 media types available:
 *  'image' Supports images
 *  'video' Supports videos
 */
$cfg['multimedia_types'] = array('image');
```

Template

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

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

$unique_id = 'bx-slider-'. fw_unique_increment();
?>
<?php if (isset($data['slides'])): ?>
    <script type="text/javascript">
        jQuery('document').ready(function () {
            jQuery('#<?php echo $unique_id ?>').bxSlider();
        });
    </script>
    <ul id="<?php echo $unique_id ?>" class="bxslider">
        <?php foreach ($data['slides'] as $slide): ?>
            <li>
                <?php if ($slide['multimedia_type'] === 'video') : ?>
                    <?php echo fw_oembed_get($slide['src'], $dimensions); ?>
                <?php else: ?>
                    <img src="<?php echo fw_resize($slide['src'], $dimensions['width
↪'], $dimensions['height']); ?>"
                         alt="<?php echo esc_attr($slide['title']) ?>"
                         width="<?php echo esc_attr($dimensions['width']); ?>"
                         height="<?php echo $dimensions['height']; ?>"/>
                <?php endif; ?>
            </li>
        <?php endforeach; ?>
    </ul>
<?php endif; ?>
```

The `$data` variable that is available in view, has the following structure:

```
$data = array(
    'slides' => array(
        array(
            'title' => 'Slide Title',
            'multimedia_type' => 'video|image',
            'src'   => 'Slide src',
            'desc'  => 'Slide Description',
            'extra' => array(
                /**
                 * This array can be empty, it depends on population method
                 * or if user set extra options for population method
                 */
                'extra-slide-key' => 'Extra slide value',
                ...
            )
        ),
        ...
    ),
    'settings' => array(
        'title'             => 'Slider Title',
        'slider_type'       => '{slider-type}',
        'population_method' => 'posts|categories|tags|custom',
        'post_id'           => 10, // ID of the slider (slider is a custom post)
        'extra' => array(
            /**
             * This array can be empty.
             * Or will have something in it
             * if user set custom options for slider in options/options.php
             */
            'extra-slider-key' => 'Extra slider values',
            ...
        )
    )
);
```

Options

Optionally, if your slider have extra *options*, you can create 2 types of option files within `options/` directory:

- `options.php` - extra options shown after default options on add and edit slider page.
- `{population-method}.php` - extra options for concrete population method, shown after default options on edit slider page.

Static files

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

- `static/images/` - directory for images. This directory has 2 special images that you should create:
  - `thumb.png` - small image with frontend preview of this slider type. Is displayed on the admin side in Slider Type choices.
  - `preview.png` - a bigger image with frontend preview of this slider type. It is displayed when the user hovers the `thumb.png` in the WordPress admin.
- `static/css/` - directory for styles. They will be automatically enqueued in frontend.
- `static/js/` - directory for scripts. They will be automatically enqueued in frontend.

Note: Styles and scripts are enqueued in alphabetical orders. You cannot set dependencies for them. So if you want for e.g. `c.js` to be enqueued before `b.js`, you must rename it, or prefix it with some number or letter `a-c.js`.

Create advanced slider type

If you want to create an advanced slider with your own extra logic, you must create a class file named `class-fw-extension-{slider-type}.php` within the slider type directory.

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

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

class FW_Extension_Bx_Slider extends FW_Slider
{
    /**
     * @internal
     */
    public function _init()
    {
    }
}
```

Then you can take a look at the `FW_Slider` methods to learn what are they doing and decide which one you will overwrite.

Frontend render

There are two ways you can display a slider in frontend:

1. **Builder shortcode** - the main slider extension automatically creates a `[slider]` shortcode which is available in *builder* in the **Media Elements** tab.
2. **Render from code** - the slider extension has a public method that you can use to render a slider on frontend.

```
fw()->extensions->get('slider')->render_slider(10, array(
    'width'  => 300,
    'height' => 200
));
```

Mega Menu

The Mega Menu extension gives the end-user the ability to construct advanced navigation menus.

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
        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-exclamation" href="#">Mega Menu 1</a>
        <div class="mega-menu">
            <ul class="sub-menu mega-menu-row">
                <li class="mega-menu-col">
                    <a href="#">Just Links</a>
                    <ul class="sub-menu">
                        <li>
                            <a href="#">Menu Item 1</a>
                        </li>
                        <li>
                            <a href="#">Menu Item 2</a>
                        </li>
                        <li>
                            <a href="#">Menu Item 3</a>
                        </li>
                        <li>
                            <a href="#">Menu Item 4</a>
                        </li>
                        <li>
                            <a href="#">Menu Item 5</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-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-wrench" href="#">Menu Item 2</a>
                        </li>
                        <li class="menu-item-has-icon">
                            <a class="fa-italic" href="#">Menu Item 3</a>
                        </li>
                        <li class="menu-item-has-icon">
                            <a class="fa-ellipsis-v" href="#">Menu Item 4</a>
                        </li>
                        <li class="menu-item-has-icon">
                            <a class="fa-home" href="#">Menu Item 5</a>
                            <p>Suspendisse potenti. Morbi a elit non mauris tempor␣
↪consequat. Praesent dapibus malesuada ligula, a fermentum leo euismod nec. Nunc␣
↪porta ligula id velit interdum congue. In mi augue, sodales a convallis id,␣
↪accumsan vitae nisi. Mauris id laoreet quam, vel hendrerit enim. Nunc ultricies␣
↪diam id neque vulputate, eu egestas est convallis. Nullam sed nisi vehicula turpis␣
↪pharetra rutrum. Nunc scelerisque sodales elit, nec elementum nisl varius vel.␣
↪Aliquam accumsan tellus a tortor porta mollis.</p>
                        </li>
                    </ul>
                </li>
            </ul>
        </div>
    </li>
    <li class="menu-item-has-icon">
        <a class="fa-info-circle" href="#">Home</a>
        <ul class="sub-menu sub-menu-has-icons">
            <li class="menu-item-has-icon">
                <a class="fa-info-circle" href="#">Page 2</a>
            </li>
            <li class="menu-item-has-icon">
                <a class="fa-info-circle" href="#">Page 3</a>
                <ul class="sub-menu sub-menu-has-icons">
                    <li class="menu-item-has-icon">
                        <a class="fa-key" href="#">Page 4</a>
                    </li>
                    <li class="menu-item-has-icon">
                        <a class="fa-briefcase" href="#">Page 5</a>
                    </li>
                    <li class="menu-item-has-icon">
                        <a class="fa-gavel" href="#">Page 6</a>
                        <ul class="sub-menu sub-menu-has-icons">
                            <li class="menu-item-has-icon">
                                <a class="fa-globe" href="#">Page 7</a>
                            </li>
```

```
                                <li>
                                    <a href="#">Page 8</a>
                                </li>
                            </ul>
                        </li>
                    </ul>
                </li>
            </ul>
        </li>
</ul>
```

Sidebars

A way to customize WordPress pages with dynamic sidebars.

Configuration

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

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

$cfg = array(
    'sidebar_positions' => array(
        'position-id' => array(
            /**
             * Image from: framework-customizations/extensions/sidebars/images/
             * (required)
             */
            'icon_url' => 'picture.png',
            /**
             * Number of sidebars on page.
             * The maximum number is 4.
             * (optional)
             * (default 0)
             */
            'sidebars_number' => 0
        ),
        // other positions ...
    ),
    /**
     * Array that will be passed to register_sidebar($args)
     * Should be without 'id' and 'name'.
     * Will be used for all dynamic sidebars.
     */
    'dynamic_sidebar_args' => array(
        'before_widget' => '<div id="%1$s" class="widget %2$s">',
        'after_widget'  => '</div>',
        'before_title'  => '<h3>',
        'after_title'   => '</h3>',
    ),
);
```

Helpers

- `fw_ext_sidebars_show($color)` - display sidebar in frontend. The parameter `$color` is the color of the sidebar selected from the WordPress admin and can be: `blue`, `yellow`, `green` or `red`.
- `fw_ext_sidebars_get_current_position()` - can be called in the frontend to find out current position name. It returns `position-id` from `$cfg['sidebar_positions']`, or null.
- `fw_ext_sidebars_get_current_preset()` - can be called in the frontend to find out the sidebar's settings for current page template.

```
// file: sidebar-content.php

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

$current_position = fw_ext_sidebars_current_position_get();

if ($current_position !== 'position-id') {
    echo fw_ext_sidebars_show('green');
}

if ($current_position === 'position-id-2') {
    echo fw_ext_sidebars_show('blue');
}

if ($current_position === 'position-id-3') {
    echo fw_ext_sidebars_show('yellow');
}
```

Filters

- `fw_ext_sidebars_post_types` - use this filter to change/remove post types that are used in the extension.

```
function _filter_remove_post_type_from_sidebars($post_types_list) {
    unset($post_types_list['post_type_name']);

    return $post_types_list;
}
add_filter('fw_ext_sidebars_get_post_types', '_filter_remove_post_type_
↪from_sidebars' );
```

- `fw_ext_sidebars_taxonomies` - use this filter to change/remove taxonomies that are used in the extension.

```
function _filter_remove_taxonomy_from_sidebars($taxonomy_list) {
    unset($taxonomy_list['taxonomy_name']);

    return $taxonomy_list;
}
add_filter('fw_ext_sidebars_get_taxonomies', '_filter_remove_taxonomy_
↪from_sidebars');
```

Styling

A way to allow users to control the design of some front end elements. This extension creates a page in **Appearance > Customize** menu.

Option Types

- `style` - generates the styling settings. It should be used only once in settings options and has the following structure:

```
'theme_style' => array(
    'label' => false,
    'type'  => 'style',
    /**
     * Must contain a list of predefined styles
     * that contains the name, icon and value for each block element
     */
    'predefined' => $predefined = include_once( 'includes/predefined-
↪styles.php' ),
    /**
     * Must contain the initial value for each element from each block
     */
    'value' => $predefined['black']['blocks'],
    /**
     * Each key contains the necessary settings to stylize one or more␣
↪blocks of the site
     */
    'blocks' => array(
        'block-1' => array(
            'title' => __( 'Header', 'fw' ),
            /**
             * Elements that can be controlled
             * Allowed elements are: 'h1', 'h2', 'h3', 'h4', 'h5', 'h6',
↪'p', 'links', 'links_hover', 'background'
             */
            'elements' => array( 'h1', 'links', 'links_hover', 'background
↪' ),
            'css_selector' => array(
                '#masthead',
                '.primary-navigation .mega-menu',
                '.primary-navigation .mega-col',
                '.primary-navigation .mega-row',
            ),
            /**
             * Additional options that will be displayed in this block,␣
↪before standard elements
             */
            'before' => array(
                'menu_border' => array(
                    'label' => __('Menu Border', 'fw'),
                    'desc'  => false,
                    'type'  => 'color-picker',
                    'value' => '#cccccc',
                )
            ),
            /**
             * Addition options that will be displayed in this block,␣
↪after standard elements
             */
            'after' => array(
                // ...
            )
        ),
        'block-2' => array(
            // ...
        ),
    ),
)
```

An example of `before` option usage:

```
/**
 * @internal
 */
function _action_print_additional_css() {
    $theme_style = fw_ext_styling_get('theme_style', '');
    $menu_border = (!empty($theme_style['blocks']['block-1']['before'][
↪'menu_border']))
        ? $theme_style['blocks']['header']['before']['menu_border']
        : '#cccccc';

    if (!empty($quick_css)) {
        echo ''.
        '<style type="text/css">'.
            '.primary-menu .mega-menu .mega-menu-col,'.
            '.primary-menu .mega-menu .mega-menu-row {'.
            '    border-color: '. $menu_border .';'.
            '}'.
        '</style>';
    }
}
add_action( 'wp_head', '_action_print_additional_css', 100 );
```

Configure

The options that appear on the customization page are located in the extension's directory `/options/settings.php` and can be overwritten by copying the file to `framework-customizations/extensions/styling/options/settings.php` in the child theme.

Helpers

- `fw_ext_styling_get($option, $default = null)` - a safe way of getting the value of an option from the styling extension.

```
function _action_print_quick_css() {
    $quick_css = fw_ext_styling_get('quick_css', '');

    if (!empty($quick_css)) {
        echo '<style type="text/css">' . $quick_css . '</style>';
    }
}
add_action( 'wp_head', '_action_print_quick_css', 100 );
```

Breadcrumbs

Option Type

In order to configure the Breadcrumbs, an option type was created that contains all the necessary configuration options for this extension.

Note: This option is for internal use only. Do not use it for posts/terms options as it will not affect breadcrumbs functionality. It works only in the settings options, and should be used only once.

Usage example:

```
'breadcrumbs-option' => array(
    'type'  => 'breadcrumbs',
    'label' => false,
    'desc'  => false,
)
```

Helpers

- `fw_ext_breadcrumbs_render($separator = '>')` - use this function to render breadcrumbs in your template

```
<h3>My page</h3>
<?php echo fw_ext_breadcrumbs_render( '>>' ) ?>
<!-- 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' );
```

SEO

The SEO extension 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:

- **Settings Options** - a tab named **SEO** where all sub-extensions add their options. Also in the **SEO** tab there is a sub-tab named **General** with a box named **General Settings**.
- **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_admin_options` - use to add your own tab in Settings Options **SEO** tab.
- `fw_ext_seo_general_tab_admin_options` - use to add your own box in Settings Options **SEO > General** tab.
- `fw_ext_seo_general_setting_admin_options` - use to add your own options in Settings Options **SEO > General > General Settings** box.
- `fw_ext_seo_post_type_options` - add options in post options **SEO** box.
- `fw_ext_seo_taxonomy_options` - add options in term options **SEO** section.

All filters have the same parameter `$options` array.

```
/**
 * @internal
 */
function _filter_set_my_framework_titles_metas_tab( $options ) {
    $options['my_id_tab'] = array(
        'title'   => __( 'My Options', 'fw' ),
        'type'    => 'tab',
        'options' => array(
            'my_id_title' => array(
                'label' => __( 'Title', 'fw' ),
                'desc'  => __( 'Set title', 'fw' ),
                'type'  => 'text',
                'value' => ''
            ),
            'my_id_description' => array(
                'label' => __( 'Description', 'fw' ),
                'desc'  => __( 'Set description', 'fw' ),
                'type'  => 'textarea',
                'value' => ''
            ),
        )
    );

    return $options;
}
add_filter( 'fw_ext_seo_admin_options', '_filter_set_my_framework_titles_
↪metas_tab' );
```

Tags

The SEO extension has a list of built in SEO tags, but in some cases you'll want to add your own. To add a new SEO tag you have to use the `fw_ext_seo_init_tags` filter. This is the format for a SEO tag:

```
'tag_name' => array(
    'desc'  => __( 'My new tag', 'fw' ),
    'value' => '',
)
```

`tag_name` must be unique. This tag will be available as `%%tag_name%%`.

Add new tag

```
/**
 * @internal
 */
function _filter_add_my_seo_tag($tags) {
    $tags['mytag'] = array(
        'desc'  => __( 'My new tag', 'fw' ),
        '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', 'fw');
    }

    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 `fw_ext_seo_init_location` action.

Blog Posts

This extension is used to change WordPress default post labels from **Posts** to **Blog Posts**, and the categories taxonomy from **Categories** to **Blog Categories**. Considering the increased use of custom posts, we felt the name Posts doesn't explain exactly what the menu item means so we choose to change it from **Posts** to **Blog Posts**.

Note: Users will be able to modify the titles from framework `.pot` file.

Backup

The Backup extension was designed to achieve the following tasks:

1. **Backup**: Make a Full or Database only copy of the site periodically or on demand.
2. **Restore**: Do a restore from a previously made backup.

3. **Demo Install**: Make an archive of the currently selected theme with all of its settings (database plus uploads directory).

   Auto Install: Gives the ability to use these settings at the time the theme was activated.

4. **Migration**: Move a WordPress site from one place to another.

Backup and Restore

These two are the basic features of the **Backup** extension.

The backup process works the following way:

1. All the necessary data is collected and stored in a zip archive. For **Full Backup** this is the database content and all of the files under the `ABSPATH` directory. For **Database Backup** this is the database content only.
2. The backup archive is moved from a temporary location into a persistent one. By default in the `uploads/backup` directory. But it can be anywhere, for e.g. Dropbox or Amazon S3.
3. Information about this archive is registered in the database, so next time the Backup page will display it in the **Backup Archive** list.

The restore process works in the following way:

1. The archive is fetched from a persistent location and extracted to a temporary directory.
2. If it contains a database dump, then the current database will be cleared and restored from the dump file. If there are WordPress files in it, the whole directory under `ABSPATH` will be removed and restored from the archive.

The Backup extension can be extended in only one way, by writing a custom **Storage Layer**.

Storage Layer

Storage Layer is a way for the **Backup Extension** to stores backup archives. To create one, create a sub-extension that implements `FW_Backup_Storage_Interface`.

For an example of implementation take a look at the **backup-storage-local** extension.

Demo Install

Demo Install is the process of making an archive of the currently active theme, packed with all of its settings (database plus uploads directory). These settings are stored in the `auto-install` directory under the theme parent directory.

Warning: This feature by default is turned off and is enabled only when the `WP_DEBUG` constant is defined and its value is true.

If it's enabled, a **Create Demo Install** button should appear on Backup page.

Auto Install

Auto Install is the reverse process of **Demo Install**.

This feature is enabled only when current theme contains the `auto-install` directory in it.

If it's enabled, an **Auto Install** page will appear under the **Tools** menu. That page displays a button **Import Demo Content** and by clicking on it, all tables from the database will be dropped and replaced by the `auto-install/database.sql` file. Also the `uploads` directory will be replaced with the `auto-install/uploads` directory.

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 site
2. Moving it in the `uploads/backup/` directory on the new site

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

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
    )
);
```

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', 'fw'),
        'plural'   => __('Custom Projects', 'fw'),
    );

    return $labels;
}
add_filter('fw_ext_projects_post_type_name', '_filter_portfolio_labels');
```

- `fw_ext_portfolio_category_name` - portfolio taxonomy labels (plural and singular)

```
/**
 * @internal
 */
function portfolio_tax_labels_names( $labels ) {
    $labels = array(
        'singular' => __( 'Custom Category', 'fw' ),
        'plural'   => __( 'Custom Categories', 'fw' ),
    );

    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:

- `archive.php` - portfolio archive.
- `taxonomy.php` - portfolio taxonomy.
- `single.php` - portfolio single post.

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.

Activation

To activate the reviews for a particular post type, add this code to your `framework-customizations/theme/hooks.php`:

```
if (!function_exists('_action_theme_activate_feedback')):
    function _action_theme_activate_feedback() {
        add_post_type_support($post_type, 'fw-feedback');
    }
endif;
add_action('init', '_action_theme_activate_feedback');
```

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.
- `fw_ext_feedback_stars_load_view()` - renders the view that displays information about the votes. *Usually used in the comments view.*