

Usage Documentation

1. Installation

Installation on Wordpress

1. Download the latest version. Unzip the files. Drag and drop the unzipped folder into the “/wp-content/plugins” folder.
2. CHMOD 777 “cache”, “files_flutter”, “modules” and “purifier_lib/HTMLPurifier/DefinitionCache/Serializer/HTML” folders under “/wp-content/plugins/Flutter”. You should be all set.

Installation on Wordpress MU

1. Download the latest version. Unzip the files. Drag and drop the unzipped folder into the “/wp-content/mu-plugins” folder.
2. CHMOD 777 “cache”, “files_flutter”, “modules” and “purifier_lib/HTMLPurifier/DefinitionCache/Serializer/HTML” folders under “/wp-content/mu-plugins/Flutter”.
3. Copy the file /wp-content/mu-plugins/Flutter/wpmu/Flutter.php to /wp-content/mu-plugins/. You should be all set.

Flutter: Differences between Standard and MU Versions

Using Flutter on Wordpress MU is the same as using Flutter on normal Wordpress. The only differences are:

1. The top admin is the only one who can create and edit modules and panels.
2. Blogs admins can use panels to create new posts.
3. Blogs admins can use modules to create the blog Layout. Each user has his own instance of blog Layout.

2. Getting Started with Flutter Write Panels

What are Write Panels?

When you open Write > Post menu, you are limited to two fields only, the post title and the post content. Flutter Write Panel feature allows you to extend wordpress write menus with new fields such as checkboxes, text boxes, radio buttons ... etc. Each field has a unique

name, given that name you can retrieve the value of that field for the current post at any time. This is done by calling a special function named **get()**. You can use the **get()** function directly in the theme files, or you can use it in Modules.

Creating a Write Panel

1. Go to **Flutter > Write Panels** and click **Create Custom Write Panel** button,
2. Type **“Test Panel”** as the name for the write panel.
3. Click **Finish**. You should be forwarded to the panel edit page where you can start adding fields to the panel.
4. Click **Create a Field**, type **“show_me”** as the name, **“Show Me”** as the label and choose the type to be Checkbox. Click **Continue**.
5. Go to **Write** menu, you should see a new sub-menu named **Test Panel**. Choose this new sub-menu, you should see the normal Write Post page but with a new checkbox field named **Show Me**. Try creating a post with your new panel.

Using fields

Now, we will edit the theme in order to show only the posts that have the show_me checkbox checked.

1. Edit the file **index.php** in your theme folder.
2. Add the line **query_posts(\$query_string.'x_show_me=true');** just before the line **if (have_posts())**.
3. Now open you blog home page, you should see the posts that have the show_me checkbox checked only.

3. Getting Started with Flutter Layout

Flutter Layout tool allows you to define where to place the modules on the blog page.

Creating a Module

1. Go to **Flutter > Modules** and click **Create a Module** button,
2. Type “**module1**” as the name for the module and click finish. You should be forwarded to the module page.

Creating Layout-Compatible Theme Since ‘FreshPost’ we’ve re-approached the way we create fields for write panels

In order to use Flutter Layout features, you must have a Layout-Compatible theme. In the following steps, we will change the wordpress default theme into a Layout-Compatible theme.

1. Make a copy of the folder **default** under **/wp-content/themes/** and rename it to **default-layout**
2. Open the file **style.css** under **default-layout** folder and change the first line to: **Theme Name: WordPress Default (Layout-Compatible)**
3. Create a file named **canvas.php**. This file defines the layout . Copy the following code to it:

```
<!-- Canvas Page: Home Page | home -->
<div id="home" style="display: none">
<p class="canvastitle">Homepage Template:</p>
<!-- Main Body -->
<div class="body_column">
<h5 class="title">Main Content</h5>
<?php canvas_define_zone('home_banner'); ?>
</div>
</div>
```

4. **Open the file index.php** and add the following line right after **<div id="content">** tag (this line defines our layout location):

```
canvas_zone('home_banner');
```

5. Our new theme should be now ready. In order to activate our new theme, go back to wordpress, choose **Design > Themes** and choose our theme from the list of available themes.

Adding Modules to Themes

1. Go to **Flutter > Layout**, you should see a box named **module1**. Drag this box and drop it in the empty box under **Main Content**.
2. **Click Publish Changes. View the blog.** You should see a text at the top of the page saying: "This is some text generated by module1 module".

3. Flutter Reference

Retrieving Fields value

Flutter offers 3 functions that you can use in the theme files in order to get the value of a custom field associated to the current post:

- **get(\$fieldName, \$groupIndex=1, \$fieldIndex=1, true)** - Get the naked value of the field.
- **get_image(\$fieldName, \$groupIndex=1, \$fieldIndex=1)** - Wraps the value of the field in an image tag. The field must be of type **Image**.
- **get_audio(\$fieldName, \$groupIndex=1, \$fieldIndex=1)** - Wraps the value of the field in an mp3 swf player. The field must be of type **Audio**.

Where:

\$fieldName - string containing field name

\$groupIndex - integer representing the index of the group in case the group is duplicated.

\$fieldIndex - integer representing the index of the field in case the field is duplicated.

\$readyForEIP - if true and the field type is textbox or multiline textbox, the resulting value will be wrapped in a div that is ready for EIP.

In addition, Flutter offers two more functions to get the number of group/fields duplicates.

- **getFieldDuplicates (\$fieldName, \$groupIndex)** - Get number of field duplicates given field name and group duplicate index. The function returns 1 if there are no duplicates (just the original field), 2 if there is one duplicate and so on.
- **getGroupDuplicates (\$fieldName)** - Get number of group duplicates given field name. The function returns 1 if there are no duplicates (just the original group), 2 if there is one duplicate and so on. \$fieldName is the name of any field in the group

Using Fields Values in query_posts Function

Flutter allows you to filter posts based on custom fields values. There are two ways to filter posts:

- **Filtering posts with a specified value** - Assuming you have a field named “my_field”, add a the prefix 'x' to it and add it normally to query_posts. E.g.
`query_posts($query_string.'x_my_field=true');`
- **Ordering posts based on a custom field**- Assuming you have a field named “my_custom_order”, query_posts should be:
`query_posts($query_string.'customorderby=x_my_custom_order&order=ASC')`

You can find more information about query_posts here:

http://codex.wordpress.org/Template_Tags/query_posts .

Canvas.php structure

In order for a theme to be Layout-Compatible, it must contain a file named canvas.php. This file describes the layout of the pagesSince ‘FreshPost’ we’ve re-approached the way we create fields for write panels and how canvases are placed in the layout. A canvas is simply an area where you can add one or more modules. To define a layout of a page, you must define the page using the following line:

```
<!-- Canvas Page: Title of your page | DOM ID -->
```

followed by a div with the same DOM ID. Inside this div, you should define all the canvases that should appear on that page. For example, the following code defines a page named Home page with one canvas:

```
<!-- Canvas Page: Home Page | home -->
<div id="home" style="display: none">
<p class="canvastitle">Homepage Template:</p>
<!-- Main Body -->
<div class="body_column">
<h5 class="title">Main Content</h5>
<?php canvas_define_zone('home_banner'); ?>
</div>
</div>
```

In order to show the canvas on the theme, use the function `canvas_zone()` in the place where you want to show the canvas.

Modules and Layout

When a module is placed on a canvas in the Layout page, it means that Flutter will load that module whenever this page is opened. Loading a module means the Flutter will include the file `default.php` located in `Flutter/modules/{Module Name}/{Template Name}/{Template Size}/`. The layout page allows you to define which template and size to grab by opening the Module Settings overlay page (click the three small lines at the right of the module box in the canvas, if you put the mouse over this icon, a tooltip will be displayed with the text “Module Options”).

As a developer, you should write the logic that renders the module in the file `default.php`, you can create several template folders to offer you users different templates. The folder `{Template Size}` could be one of the following:

1. A number specifying the module width

2. 'small', 'medium', 'large' or 'full'

In addition to template names and sizes, Flutter allows you to define parameters for the modules that are accessible from `default.php` and configurable from Module Settings overlay window. You can define parameters in the file `Flutter/modules/{Module Name}/configure.xml`. Assume you have a parameter named `mod_par1`, you can simply get its value inside `default.php` by calling the function `mod_var('mod_par1')`.

configure.xml

In order for Flutter Layout feature to work properly, a file named `configure.xml` must exist in the folder **Flutter/modules/{module name}**, this file contains description of the module as well as a definition of its parameters. By default, whenever you create a new module on Flutter, a basic templates folder is created with one parameter named **the_text**.

`Configure.xml` contains the following fields:

author – the name of the author

description – the description of the module

`Configure.xml` can contain 0 or more variables, each variable has the following fields:

name – The name of the variable.

description – the description of the variable.

type – could be text, textarea, integer, boolean, radio, select, image, file or list.

Modules: Variable Options

Normally, when creating a dropdown (variable type: select) or radio button field (variable type: radio), the module author must specify a number of options for the user to select from. For example:

```
<variable>
  <name>
    sort_column
  </name>
  <description>
    Sort categories by
  </description>
  <type>
    Radio
  </type>
  <default>
    ID
  </default>
  <numoptions>
    2
  </numoptions>
  <options>
    <option>
      <displaytext>
        Category ID
      </displaytext>
      <value>
        ID
      </value>
    </option>
    <option>
      <displaytext>
        Category Name
      </displaytext>
      <value>
        name
      </value>
    </option>
  </options>
</variable>
```

</variable>

In this snippet, we have defined a **radio** type variable named "**sort_column**". Inside the options tag we have two options, one called category ID and one called category name. Their values are "**ID**" and "**name**" respectively.

Modules: Using the Database to Your Advantage

Let's say you want to develop a module that will only pull posts from a single category. You could ask the user for the category ID (which they may not know how to find) or category name (which they may enter incorrectly) to skip. This could be problematic if the end-user doesn't know much about the inner workings of Wordpress.

It would be simpler to provide the user with a list of available categories to choose from, which Canvas will gladly do for you. Using similar syntax, we can tell Canvas to automatically generate a list of options using the Wordpress tables as a guide.

The syntax you will be using to call database data is structured like this: **table_name->displaytext:value**

This construct is placed within the value tag of our option definition (no displaytext tag is needed). The **table_name** variable is the table name without the prefix. To query the categories table on Wordpress MU, for instance, we'd use **terms** table. The **displaytext** variable corresponds to the displaytext tag used within the option definition, and is used to generate the text label for your dropdown or radio element. It must be a column name within the table. In our example, we'll use **name**. The value variable corresponds to the value tag used within the option definition, and will provide the text used within the "value" definition of the form element.

In addition, you can further specify a WHERE condition between **<params>** tag

To create a dropdown menu containing all of the categories under the category c1, we'd use the following:

<variable>

```
<name>
cat2
</name>
<description>
Categories under category 'c1'
</description>
<type>
Select
```



```

</type>
<options>
<option>
<value>
terms->name:term_ID
</value>
<params>
1 = (SELECT count(*) FROM %prefix%term_taxonomy myTBL WHERE
parent = (SELECT term_id FROM %prefix%terms myTBL1 WHERE myTBL1.name = 'c1')
AND myTBL.term_id = TBL.term_id)
</params>
</option>
</options>
</variable>

```

If you want to use the same column for both the displaytext and the value, you can eliminate the value definition as such: **terms->name**. In this case, the column named in displaytext will be used for both variable values and labels.

EIP (Edit-n-place)

Flutter offers a feature named EIP (edit-n-place). EIP allows authors to edit the post directly from the blog page. When the blog page is opened and the current logged-in user is an author, he will be able to directly edit the post simply by clicking the post title or the post content. Once clicked, the title/content area becomes editable and a save button appears. The author can edit the title/content and click save when he finishes editing.

Since it is not trivial to know exactly the place of title/content that the designer wants to be editable, Flutter needs some help from the designer. In order to make a title editable, the designer must add a class to the tag enclosing the title in the theme file. The class name should be retrieved using the function **EIP_title()**, and for the content a similar function is used named **EIP_content()**.

In addition, you can use EIP feature for fields of type textbox and multiline textbox. When you use `get()` function, it will return the value of the textbox field wrapped in a `div`.

For example, to make a title (which is enclosed in an `<h2>` tag) EIP-compatible, use the following line :

```
<h2 class="<?php EIP_title(); ?>"> <?php the_title(); ?> </h2>
```

And to make a content (which is enclosed in a `<div>` tag) EIP-compatible, use the following line :

```
<div class="<?php EIP_content(); ?>"> <?php the_content(); ?>
</div>
```

4. Flutter API

canvas_define_zone

Description:

Used inside canvas.php in the theme folder to define an area where modules can be dropped in.

Prototype:

```
canvas_define_zone($zone_name, $template_name =  
"default", $template_size =  
MODULE_TEMPLATE_SIZE_LARGE, $allow_other_sizes =  
TRUE, $default_module = 'mod1', $allow_other_modules =  
TRUE, $styles='')
```

Parameters:

\$canvas_name – (String) The canvas name

\$template_name – (String) The name of the template to grab.
The default value is "default".

\$template_size – (Integer) Grabs the specified version of the
template. The default value is
MODULE_TEMPLATE_SIZE_LARGE. The value of
\$template_size could be

- MODULE_TEMPLATE_SIZE_SMALL,
MODULE_TEMPLATE_SIZE_MEDIUM,
MODULE_TEMPLATE_SIZE_LARGE, or
MODULE_TEMPLATE_SIZE_FULL.
- The maximum width of the module that can be placed in this
canvas.

\$allow_other_sizes – (Boolean) If you tried to put a version of
a module larger the zone version, and this value is set to True,
it will just give a warning. Otherwise, it will give error. The
default value is true.

\$default_module – (String) This will bring up a certain module by default.

\$allow_other_modules – (Boolean) If set to True, then the admin can put another module in the space. Otherwise, you shouldn't be able to move the module. The default value is true.

\$styles – (String) A string that will be printed in the style attribute of the canvas div.

canvas_zone

Description:

Used inside any theme file in order to display a canvas whenever this page is displayed.

Prototype:

canvas_zone(\$canvas_name)

get_module

Description:

Used inside any theme file in order to grab the specified module directly without the need to define it in the canvas.php.

Prototype:

get_module(\$module_name, \$template_name, \$template_size)

mod_var

Description:

Used inside the file default.php in the module folder in order to get the specified module parameter.

Prototype:

mod_var(\$parameter_name)