Layouts Theme Integration

Layouts add drop-and-drop functionality to WordPress themes, which use the Bootstrap CSS. Using Layouts, end-users can rearrange pages, move blocks, resize them and add new blocks. All this, without having to learn how the grid system works or how cells are coded.

Layouts comes with a simple API for registering custom cells. The API allows theme authors to turn their existing functionality into draggable cells, which end-users can easily manipulate.

Cells have:

1. A GUI that allows users to enter settings
2. Storage for cell options
3. Render on the site's front-end pages

Examples for cell types, which themes may define:

- Sliders
- Feature highlights
- Testimonials
- Showcase
- Products

By defining their cell types, themes retain their uniqueness. They use the drag-and-drop engine from Layouts to allow users to build sites that include the unique features in the theme.

Adapting Themes to use Layouts

Modifying a theme to support layouts requires 4 steps:

1. Create cell types for elements to display in the front end. eg, header, menu, footer, etc
2. Modify or add php templates to render layouts
3. Create layouts and save these to the theme directory
4. Loading the pre-created layouts from the theme directory

Creating Cell Types

A cell is a place for displaying content. It can be positioned and re-sized in the layout editor. It gets displayed in the front end inside a div.

Registering a new Cell type

```
<?php register_dd_layout_cell_type( $cell_type, $args ) ?>
```

Returns **true** on success and **false** on failure. If it returns false then the $cell_type is already used.
This should be called during the “init” action.

Parameters.

$cell_type - string identifier. This is used to identify the new cell type.

$args - an array of arguments.

Arguments:

- **name** - The name of the cell type.
- **icon-css** - css class name for the icon. This is displayed in the popup when creating new cell types. (optional)
- **icon-url** - url for the icon. Can be used instead of icon-css. (optional)
- **category** - Category used to group cell types together. (optional - defaults to “General cells” category.
- **category-icon-css** - css class name for the icon for the category. This is displayed in the popup when creating new cell types. (optional)
- **category-icon-url** - url for the icon. Can be used instead of category-icon-css. (optional)
- **description** - A description of the cell type. This is displayed in the popup when creating new cell types.
- **preview-image-url** - An image to be displayed in the popup when creating new cell types.
- **button-text** - The text for the button that is displayed in the popup when creating new cell types.
- **dialog-title-create** - The dialog title to be displayed when creating a new cell of this type
- **dialog-title-edit** - The dialog title to be displayed when editing a cell of this type
- **dialog-template-callback** - The function name of a callback function that supplies the user interface for creating or editing the cell type. Can be left blank if the cell type has no UI.

- **cell-content-callback** - The function name of a callback function that returns the HTML to be rendered in the front end. This function will receive the $cell_settings that the user has entered via the cell edit dialog.
- **cell-template-callback** - The function name of a callback function that returns the HTML for displaying the cell in the editor.
- **cell-class** - The class name or names to add when the cell is output. Separate class names with a space. (optional)
- **register-styles** - An array of styles required for the cell in the layouts editor. Each array item should be an array with values the same as the wp_register_sytle function. Only the first two are required - array(**$handle**, **$src,** $deps, $ver, $media)
- **register-scripts** - An array of scripts required for the cell in the layouts editor. Each array item should be an array with values the same as the wp_register_script function. Only the first two are required - array(**$handle, $src**, $deps, $ver, $in_footer)

Example:

This example creates a simple text cell. It allows user to enter a line of text and then displays it in the front end when the cell is rendered. This can be found in /example-cell-type/simple-text-cell.php

```
function register_simple_text_cell_init() {
    register_dd_layout_cell_type (
        'simple-text-cell',
        array ( 'name' => __('Simple text cell', 'theme-context'),
                'icon-css' => 'icon-font',
                'description' => __('A cell that displays simple text.', 'theme-context'),
                'category' => __('Example cells', 'theme-context'),
                'category-icon-css' => 'icon-sun',
                'button-text' => __('Assign Simple text cell', 'theme-context'),
                'dialog-title-create' => __('Create a new Simple text cell', 'theme-context'),
                'dialog-title-edit' => __('Edit Simple text cell', 'theme-context'),
                'dialog-template-callback' => 'simple_text_cell_dialog_template_callback',
                'cell-content-callback' => 'simple_text_cell_content_callback',
                'cell-template-callback' => 'simple_text_cell_template_callback',
                'cell-class' => 'simple_text_cell',
                'register-styles' => array(array('simple-text-cell-style', get_template_directory_uri() . 'my-style.css')),
                'register-scripts' => array(array('simple-text-cell-script', get_template_directory_uri() . 'my-style.js'))

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

```php
/* Callback function that returns the user interface for the cell dialog.
 * Notice that we are using 'text_data' for the input element name.
 * This can then be accessed during frontend render using
 * $cell_settings['text_data']
 */

function simple_text_cell_dialog_template_callback() {
    ob_start();
    ?>

        <p>
            <?php
                /*
                 * Use the the_ddl_name_attr function to get the
                 * name of the text box. Layouts will then handle loading and
                 * saving of this UI element automatically.
                 */
            ?>
            <h3>
                    <?php the_ddl_cell_info('name'); ?>
            </h3>

            <label for="<?php the_ddl_name_attr('text_data'); ?>">Cell content:</label>
            <input type="text" name="<?php the_ddl_name_attr('text_data'); ?>">
        </p>

    <?php

    return ob_get_clean();
}

// Callback function for displaying the cell in the editor.
function simple_text_cell_template_callback() {

   // This should return an empty string or the attribute to display.
   // In this case display the ‘text_data’ attribute in the cell template.

   return 'text_data';
}

// Callback function for display the cell in the front end.
function simple_text_cell_content_callback($cell_settings) {
```

```
    // Get the text using the 'text_data' key.

    $content = $cell_settings['text_data'];

    return $content;
}
```

Load cell template files in your theme:

We recommend to create separate files for different cells. Each of these files can include the functions for displaying the cell in the Layout GUI and rendering it.

After creating the cell files you'll want to upload them to make them visible in your theme.

To do so you can simply include them in your functions.php file.

You can use this helper function in order to do that programmatically:

```
if( class_exists( 'WPDD_Layouts' ) && !function_exists( 'include_ddl_layouts' ) )
{
	function include_ddl_layouts( $tpls_dir = '' )
	{
		$dir_str = dirname(__FILE__) . $tpls_dir;
		$dir = opendir( $dir_str );
		while( ( $currentFile = readdir($dir) ) !== false )
		{
			if ( $currentFile == '.' || $currentFile == '..' || $currentFile[0] == '.' )
			{
				continue;
			}

			include $dir_str.$currentFile;
		}
		closedir($dir);
	}
	include_ddl_layouts('/inc/layouts/');
}
```

The function takes as an argument the path of the layouts to read.

Inserting Layouts to PHP templates

In your PHP templates, add a call to **the_ddlayout**, to render a layout. The call receives the layout slug as an argument.

To create a complete PHP template, which uses a layout, you should include:

- <head> info </head> and related things
- open the the <body> tag
- call **the_ddlayout** to render a layout
- output any footer info
- close the </body> tag

Example page template using a layout (page-layouts.php):

```
<?php
/*
Template Name: Layouts page template
*/
get_header('layouts'); ?>

   <?php
     the_ddlayout('default-page-layout');
   ?>

<?php get_footer('layouts');
```

Simplified header to match (header-layouts.php):

```
<?php
/**
 * The template for displaying the header.
 *
 */
?>
<!DOCTYPE html>
<html>
<head>

    <meta charset="<?php bloginfo( 'charset' ); ?>" />
    <link rel="stylesheet" type="text/css" media="all" href="<?php bloginfo( 'stylesheet_url' ); ?>"
/>
    <?php
        wp_head();
    ?>
</head>

<body <?php body_class(); ?>>
```

Simplified footer to match (footer-layouts.php):

```
<?php
/**
 * The template for displaying the footer.
 *
 */
?>

<?php wp_footer(); ?>

</body>
</html>
```

Exporting layouts and using in the theme:

Exporting:

Layouts can be exported from the “Theme export” submenu under Layouts. The layouts can be exported to the default “theme-dd-layouts” directory or exported and downloaded as a zip file.

Importing:
Layouts can be imported from the theme directory using the ddl_import_layouts_from_theme_dir function. You can pass the full path to the a directory that contains the layouts to load or leave it blank to import from the default ‘theme-dd-layouts’ subdirectory in the theme.

You’ll do this on init with high priority since the $wpddlayout global object would not be defined before init - the priority should be > 9.

For php < 5.3:

```
<?php
add_action( 'init', 'import_layouts', 99 );
function import_layouts()
{
        ddl_import_layouts_from_theme_dir($theme_layouts_dir);
} ?>
```

For php >= 5.3

```
<?php
add_action( 'init', function(){
        ddl_import_layouts_from_theme_dir($theme_layouts_dir);
}, 99 );
```

Handling Dialog Events:

Sometimes a cell type might require special handling when the cell dialog editor opens and closes. These can be handled via events in Javascript.

The following events are fired:
When the dialog opens, an event named CELL_TYPE.dialog-open is triggered.
When the dialog closes, an event named CELL_TYPE.dialog-close is triggered.

For our simple test cell example the cell type is **simple-text-cell**.

simple-text-cell.dialog-open and simple-text-cell.dialog-close are triggered.

```
// Custom events for simple-text-cell.php

jQuery(function($) {

    // On simple text cell dialog open
    $(document).on('simple-text-cell.dialog-open', function() {
        alert('Dialog Open event has been triggered for Simple Text Cell example');
    });

    // On simple text cell dialog close
    $(document).on('simple-text-cell.dialog-close', function() {
        alert('Dialog Close event has been triggered for Simple Text Cell example');
    });

});
```

Fields API:

The fields API can be used to display values of fields in the front end. The field API can only be used in the **cell-content-callback** function.

Fields can be either simple, with a single value for the field, or repeating.

Simple fields

Displaying cell values:
<?php **the_ddl_field**($field_name); ?>

Testing cell values:
<?php if (**get_ddl_field**($field_name) == 'some value') { …. } ?>

Repeating fields

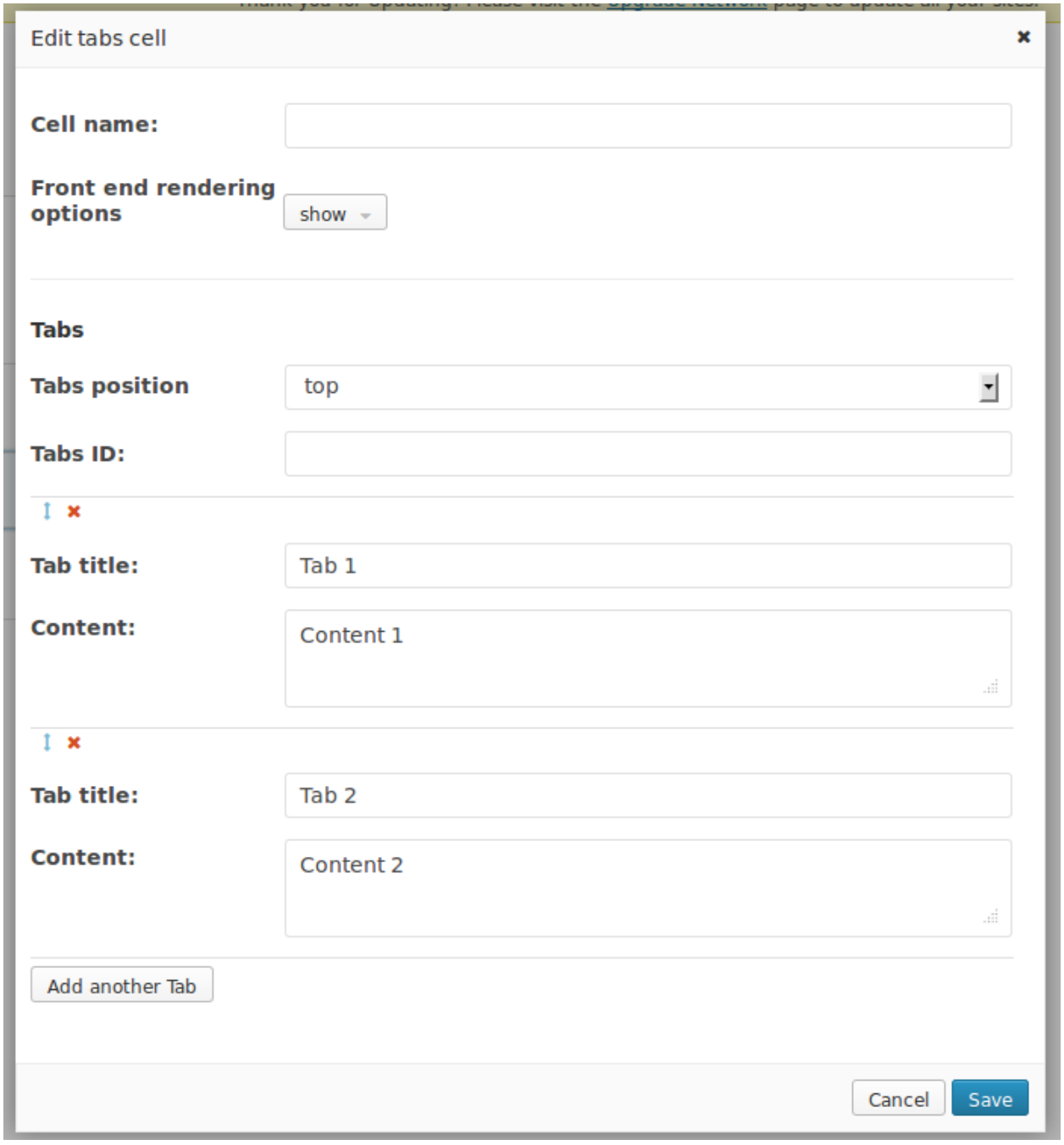
Sometimes you need multiple repeating values for a field or a group of fields. An example of this would be a tab cell where the user can have a number of tabs and content for each tab.

<?php **ddl_repeat_start(**$group_name**,** $button_label**,** $max_items **);** ?>

Specifying the repeating fields in the dialog content callback.

```
<?php ddl_repeat_start( 'tabs', __('Add another Tab', 'theme-context' ), 5 ); ?>
        <p>
                <label for="<?php the_ddl_name_attr('title'); ?>">Tab title:</label>
                <input type="text" name="<?php the_ddl_name_attr('title'); ?>">
        </p>
        <p>
                <label for="<?php the_ddl_name_attr('content'); ?>">Content:</label>
                <textarea name="<?php the_ddl_name_attr('content'); ?>"></textarea>
        </p>
<?php ddl_repeat_end(); ?>
```

The fields between the ddl_repeat_start and ddl_repeat_end will be repeated in the User Interface. Layouts will handle the repeating fields and allow addition, re-ordering and deletion of the fields.


Edit tabs cell
Cell name:
Front end rendering options
show
Tabs
Tabs position
top
Tabs ID:
Tab title:
Tab 1
Content:
Content 1
Tab title:
Tab 2
Content:
Content 2
Add another Tab
Cancel
Save

Rendering repeating fields in the front end:

<?php **has_ddl_repeater**($group_name); ?>
Checks to see if the group has any more items.

<?php **the_ddl_repeater**($group_name); ?>
Moves to the next item in the group.

<?php **the_ddl_repeater_index**($group_name); ?>
Displays the repeater index for the group

<?php $index = **get_ddl_repeater_index**($group_name); ?>
Gets the repeater index for the group

<?php **ddl_rewind_repeater**($group_name); ?>
Resets the loop for the group

<?php **the_ddl_sub_field**($field_name); ?>
Displays the sub field for the current item in the group

<?php $field = **get_ddl_sub_field**($field_name); ?>
Gets the sub field for the current item in the group

Typical loop example.

```
<?php if ( has_ddl_repeater('tabs') ) : ?>
<ul>
    <?php while ( has_ddl_repeater('tabs') ) :  the_ddl_repeater('tabs');?>
        <li>
            <h2><?php the_ddl_sub_field('title') ?></h2>
            <p><?php the_ddl_sub_field('content') ?></p>
        </li>
    <?php endwhile; ?>
</ul>
<?php endif; ?>
```

Embedding layouts in a theme:

The embedded layouts zip file should be unzipped to the theme root directory. The embedded layouts code will be in the “embedded-layouts” subdirectory.

To include the code into a theme, the following code needs to be added near the top of the theme’s functions.php file:

```
require_once dirname(__FILE__) . '/embedded-layouts/ddl-theme.php';
```