Revisiting Gutenber blocks part 2: Further thoughts and ideas

The previous post (Revisiting Gutenberg blocks part 1: Building and Styling the blocks) discussed how to build a block and reviewed both old and new techniques for creating Gutenberg blocks.

This post will revisit related areas of block development in more detail.

Block variations

Some times it may be easier to create additional styles for an existing core element than create your own. Gutenberg calls this block variations.

Authors can also choose whether they want to incorporate core blocks. It is highly likely that you will need to customize the block styles to match the core blocks with the theme but that will alwaays be easier than creating. Gutenberg calls this block style variations.

The following core blocks are available:

- Text
    - Paragraph
    - List
    - Heading
    - Subhead
    - Table
    - Button
    - Classic (HTML) Block
    - Text Columns
- Media
    - Image
    - Cover Image
    - Image Gallery
    - Audio
    - Video

        - Embeds
        - Shortcodes
    - Quotes
        - Quote
        - Pullquote
        - Verse
    - Code
        - Code
        - Preformatted
        - Custom HTML
        - Shortcodes
        - Embeds
    - Layout
        - Columns
        - Text Columns
        - Separator
        - Read More

See Gutenberg Blocks for more information about core blocks and how to use them.

The variations plugin doesn't require a build step, it's a combination of PHP and CSS.

The PHP script enqueues the script that defines the variations and the stylesheet that holds the actual CSS for the variations we create.

```
<?php
function rivendellweb_enqueue_variations() {
    wp_enqueue_script(
        'rivendellweb-script',
        plugins_url( './src/block-variations.js', __FILE__ ),
        array( 'wp-blocks', 'wp-dom-ready', 'wp-edit-post' ),
        filemtime( plugin_dir_path( __FILE__ ) . './src/block-variations.j
    );
}
add_action( 'enqueue_block_editor_assets', 'rivendellweb_enqueue_variation

function rivendellweb_variation_styles() {
```

```
  wp_enqueue_style(
    'rivendellweb_variations_css',
    plugins_url( './src/block-variations.css', __FILE__ ) );
}
add_action( 'enqueue_block_assets', 'rivendellweb_variation_styles' );
```

The Javascript file contains only block style registrations for our custom styles. It uses `blocks.regiterBlockStyle` to give WordPress the following informaiton:

- What block will use the style
- What's the name of the style
- What's the label for the style

At this time we're not removing existing styles from blocks, only adding new ones.

```
wp.blocks.registerBlockStyle( 'core/paragraph', {
  name: 'lede-paragraph',
  label: 'First Paragraph',
'lede-paragraph'// Blockquote variations
wp.blocks.registerBlockStyle( 'core/quote', {
  name: 'fancy-quote',
  label: 'Fancy Quote',
} );

wp.blocks.registerBlockStyle( 'core/quote', {
  name: 'top-bottom-quote',
  label: 'Top and Bottom',
})

wp.blocks.registerBlockStyle( 'core/quote', {
  name: 'red-quote',
  label: 'Red Quote',
})
```

Aas mentioned earlier, the CSS defines the actual styles for our block variations using the `.is-style{variation-name}` CSS class syntax.

```css
.is-style-lede-paragraph {
  font-size: 1.5em !important;
}

.is-style-fancy-quote {
  border-left: 4px solid red;
}

.is-style-top-bottom-quote {
  border-top: 4px solid black;
  border-bottom: 4px solid black;
  border-left: 0;
}

p {
  margin: 1em 0;
}

.is-style-red-quote {
  border-left: 0;
  color: red;
  font-size: 5vmax;
}
```

Block style variations provide a way to customize blocks without changing the blocks themselves; variations are also clean, they don't require a build process since they use PHP to enqueue assets.

One thing that tripped me when writing this, is that variations appear once you've inserted the block on the post or page.

Block patterns

The idea behind block patterns is that we can group blocks that we use together into a single layout that we can reuse where needed.

The steps to create a Block Pattern are as follows:

1. Create the pattern and copy the content

We create the pattern by setting up the blocks we want as part of the pattern in Gutenberg. This will create the markup that we need to paste into the pattern definition by copying it from the editor and escaping it, like we'll do in the next step.

2. Escape the block content

Until we can create a built-in conversion tool, we'll have to use an online tool like escape string, or the native PHP json_encode function to convert the code into a string suitable to add to the pattern.

One incomplete solution is to create a heredoc to contain the full Gutenberg template.

We then encode the template to a string using json_encode() to create the string.

Lastly, we then use preg_replace() to make sure we replace all instances of `\/` with `/`

```
<?php
$video_text_content = <<<EN<<<END
 // Content of the template goes here
END;nverted_json = json_encode($video_text_content);

$template_string = preg_replace('(\\\/)','/',$converted_json);
'(\\\/)'
```

We can run this as an external script in the command line or, if we choose to, we can incorporate it into our block pattern definitions or maybe add it to a WordPress utility library to be created at a later time.

3. Register the pattern

The pattern registration is fairly simple. We first check if the WP_Block_Patterns_Registry an only add the pattern if it does (it doesn't make sense to register the pattern if it doesn't).

register_block_pattern() tells WordPress that we're creating a new block pattern. It takes two parameters: a pattern name and an array of pattern properties.

The possible pattern properties are:

- `title` **(required)**: A human-readable title for the pattern
- `content` **(required)**: Stringified content string for the pattern
- `description`: A visually hidden text used to describe the pattern in the inserter
- `categories`: One or more pattern categories for the pattern to appear in
- `keywords`: Aliases or keywords that for the pattern
- `viewportWidth`: Width of the pattern in the inserter.

The final step is to hook the pattern registration function to the init hook to make sure it actually works.

```
<?php
function rivendellweb_register_block_patterns() {
  if ( class_exists( 'WP_Block_Patterns_Registry' ) ) {

    register_block_pattern(
      'rivendellweb/video-text',
        'title'       => __( 'Video with text description', 'textdomain' )
        'description' => __( 'Youtube video with text to the right.', 'Blo
        'content'     => $template_string,
        'categories'  => array( 'Rivendellweb Patterns' ),
    );
  }
}

add_action( 'init', 'rivendellweb_register_block_patterns' );
```

The function can register multiple patterns at the same time.

4. (Optional) Create custom categories for the pattern

Rather than put our custom patterns in the default categories, we can create one

or more custom categories for our patterns that will share space with the default ones (although not in alphabetical order).

The callback function, `rivendellwebr_register_pattern_categories()` allows us to register one or more pattern categories using register_block_pattern_category()

```
<?php
function rivendellwebr_register_pattern_categories() {
  if ( class_exists( 'WP_Block_Patterns_Registry' ) ) {

    register_block_pattern_category(
      'journal',
      array('label' => __( 'Journal', 'Block pattern category', 'textdoma
    );

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

So now we have as many custom patterns in as many custom pattern categories as our project needs.

Additional resources

Rich Tabor's How to Build Block Patterns for the WordPress Block Editor provides an indepth look at building patterns.

Getting Custom Post Types to Talk to Gutenberg

I've created several custom post types that worked in the classic editor. When I brought them to my Gutenberg playground they would not work or they would not work as I expected them to.

It took me a while to realize that you must be really picky on how you configure the Custom Post Type (CPT) to work as Gutenberg is far less forgiving than the classic editor when using register_post_type()

I want my CPT to do three things

1. Work with the REST API
2. Work with the classic editor
3. Work with Gutenberg

To address point one, we need to include `show_in_rest` in the post registration

To address points two and three we need to include the full support declaration for what part of the editor we want to support.

I normally add the following items to the `supports` array:

- `title`
- `editor`
- `excerpt`
- `author`
- `thumbnail`
- `comments`
- `revisions`
- `custom-fields`
- `permalinks`
- `featured_image`

```
<?php
function rivendellweb_custom_book_type() {
  register_post_type( 'book'book'// WordPress CPT Options Start
  array(
      'labels' => array(
          'name' => __( 'Books' ),
          'singular_name' => __( 'Book' )
      ),
      'has_archive' => true,
      'public' => true,
      'rewrite' => array('slug' => 'book'),
      'show_in_rest' => true,
      'supports' => array('title',
          'editor',
          'excerpt',
          'author',
```

```
            'thumbnail',
            'comments',
            'revisions',
            'custom-fields',
            'permalinks',
            'featured_image'
        )
    )
);
}
add_action( 'init', 'rivendellweb_custom_book_type' );
```

Further tweaks to block styles

There are a few things that I would still want to customize for the blocks and the theme using them.

Custom color palettes

```
<?php
add_theme_support(
'edit'editor-font-sizes'ay(
  array(
      'name' => __( 'Small', 'rivendellweb-blocks' ),
 'name'size' => 10,
      'slug' => 'small'
  ),
  array(
      'name' => __( 'Regular', 'rivendellweb-blocks' ),
      'size' => 16,
' => 10,
      ''regular'
  )
  'regular'// truncated to save space
);
```

Like we did with the custom fonts sizes we do the same thing with a custom color

palette.

```
/**
 * Theme Setup
 *
 */
function rivendellweb_setup() {
  // Disable Custom Colors
  add_theme_support( 'disable-custom-colors' );

  'disable-custom-colors'// Editor Color Paletteeditor-color-palette', ar
    array(
      'name'  => __( 'Blue', 'rivendellweb' ),
      'slug'  => 'blue',
      'color' => '#59BACC',
    ),
    ar', array(
    array(
      '#59BACC',ivendellweb' ),
      'slug'  => 'green',
      'color' => '#58AD69',
    ),
    array(
      'name'  => __( 'O' ),
      '#58AD69',' ),
      'slug'  => 'orange',
      'color' => '#FFBC49',
    ),
    array(
      'name'  => __( 'Red', 'rivend' ),
      '#FFBC49',g'  => 'red',
      'color' => '#E2574C',
    ),
  ) );
}
add_action( 'after_setup_theme', 'rivendellweb_se'  => '#E2574C',
    ),
  ) );
```

```
}
add_action( 'after_setup_theme', 'rivendellweb_setup' );
```

Fonts and typography

Templates

External block templates

External metdata declaration

The block type metadata provides an external means to declare our block API. It is stored in a `block.json` file. An example, taken from the Block metadata developer docs looks like this:

```
{
  "apiVersion": 2,
  "name": "my-plugin/notice",
  "my-plugin/notice""title"egory": "text"": ""category"[ "core/group" ],
  "core/group"r",
  "description": "": ""description""icon": "star",
  "description"ywords": [ "alert", "message" ],
  "textdomain":": [ ""keywords": [ "alert", "message" ],
  "textdomain"type": "string",
": ""attributes""html",
"html""message"": ".messa": ""type": "string",
      "source": "html",
      "selector" "message"
  },
  "usesCon"message""providesContext": {
    "my-plugin/message": "message"
  },
  "usesContext"e": "default", "lab": ""supports": {
    "align": true
  },
  "styles""other", ""other""name": "default", "label": "Default", "isDefau
     "message": "Thi": ": "other", "label"  },
```

```
  "editorScript": "file": ""editorScript": {
    "attributes": {
      "message"
  "editorStyle": "file:./build/ind": ""editorScript": "file:./build/style
}
"file:./build/style.css""script": "file:./build/script.js",
  "editorStyle": "file:./build/index.css",
  "style": "file:./build/style.css"
}
```

Look at the Block metadata developer docs for more information about the content of the file and how it works.

Full page designs

Block design: Journal