Block Variations

- What are block variations?
  - Block variations vs. block styles
- Working with block variations
  - Setup: Loading JavaScript
  - Registering block variations
  - Unregistering block variations

The Block Variations API is a powerful feature that allows you to extend any registered blocks. Basically, it gives you the ability to create alternate versions of a block's settings. This can make it easier for your users to insert blocks without the hassle of going through all of the setup they might need for a specific scenario.

The Block Variations API is often pitched as a plugin developer API. In many cases, plugins are the best place for variations to live. But there are also use cases for theme creators.

Several WordPress features use the term "variations" in their name. This can be confusing at times. The biggest thing to note is that block variations are not the same thing as global style variations and block style variations.

What are block variations?

The simplest explanation of block variations is that they are literally variations on an existing block. Let's take a look at an example to explore this further.

The Social Icon block is a simple block to add a link to some social site, but WordPress bundles many different variations of this base block to cover a wide range of social sites:

WordPress post editor with several social icons shown in a row.

Without the Block Variations API, each of those variations would need to be individually-coded blocks. There are many reasons that would be a bad idea. So instead of writing a block for each social icon, developers only need to make a few configuration changes to create an alternate

version of the original block.

There are many, many ways that you could potentially change a block to make it different from its default configuration. This can be something as simple as setting up a Spacer block with your theme's preferred default spacing. Or it can be as complex as a custom Query Loop block with a complex posts query for the front-page template.

While it is more advanced than some theme features, the API does give you the flexibility to make more complex themes than what is available out of the box.

Block variations vs. block styles

An oft-repeated question is: *When should I create a block variation vs. a block style?*

The answer is almost always the same as the answer to another question: *Can these changes be made through* `theme.json` *styles or CSS?* If yes, you should almost always create a custom block style.

But if you need to change a block's settings so that it is set up differently when a user inserts an instance of it into a post, a custom block variation is probably the best route to take.

Working with block variations

Block variations require that you work with JavaScript instead of PHP. The examples below do not require a build process, so you can use them as-is.

But the more you work with WordPress' JavaScript packages, the more likely you'll want to incorporate build tools into your workflow. It's possible to use the `@wordpress/scripts` package, originally meant block development, with your theme. For a deeper dive into how this works, read Beyond block styles, part 1: using the WordPress scripts package with themes.

In the next sections, you will learn how to work with block variations by building a simple variation on the WordPress Spacer block, as shown in this screenshot:

WordPress post editor with a Table of Contents block, Spacer block, and demo content.

For a deeper dive into block variations, check out these tutorials on the WordPress Developer Blog:

- An introduction to block variations
- Building a book review grid with a Query Loop block variation

Setup: Loading JavaScript

The first step for working with block variations is to create an empty file for adding your custom JavaScript to. This can be at any location you choose, but the code example below will attempt to find a `block-variations.js` in your theme's `/assets/js` folder.

So your theme structure should look similar to this:

- `/assets`
  - `/js`
    - `/block-variations.js`
- ...other files and folders

To load that file in the editor, you must call the `wp_enqueue_script()` function within a callback added to the `enqueue_block_editor_assets` action hook.

Add this code to your theme's `functions.php` file:

```
add_action( 'enqueue_block_editor_assets', 'themeslug_enqueue_block_variations' );

function themeslug_enqueue_block_variations() {
    wp_enqueue_script(
        'themeslug-block-variations',
        get_theme_file_uri( 'assets/js/block-variations.js' ),
        array(
            'wp-blocks',
            'wp-dom-ready',
            'wp-i18n'
        ),
        wp_get_theme()->get( 'Version' ),
        true
    );
}
```

Take note of the dependencies array (third parameter) within `wp_enqueue_script()`. This includes the `wp-blocks`, `wp-dom-ready`, and `wp-i18n` scripts, which are needed for the JavaScript code in the below examples.

Registering block variations

To register block variations, you must use the `registerBlockVariation()` JavaScript function. Here is a look at the function's signature:

```
const registerBlockVariation = ( blockName, variation )
```

The function accepts two parameters:

- **`blockName`:** The name of the block (including namespace) to register the variation for.
- **`variation`:** An optional object for configuring the variation, which may include any of the following properties:
  - **`name`:** A unique, machine-readable slug for your variation.
  - **`title`:** A human-readable title for your variation that should be internationalized.
  - **`description`:** A human-readable description of the variation that should be internationalized if added.
  - **`category`:** The slug of a registered block type category
  - **`keywords`:** An array of keywords to help users discover the variation when searching.
  - **`icon`:** An icon to use to visualize the variation. Either a string or object is accepted.
  - **`attributes`:** An object that overrides the block's attributes.
  - **`innerBlocks`**: An array to handle the initial configuration of nested blocks.
  - **`example`:** An object that provides structured data for the block preview. Can be set to `undefined` to disable the preview.
  - **`scope`:** A list of scopes where the block can be used. Available options are: `block`, `inserter`, and `transform`.
  - **`isDefault`:** Whether to set the variation as the default variation of the block. Defaults to `false`.
  - **`isActive`:** A function or array of block attributes used to determine if the variation is active when the block is selected.

To learn more about block variations, read the Block Variations API documentation in the Block Editor Handbook.

To build your custom variation, the most important thing is to think about the attributes that you want to control that will set this variation apart from the default block. That's what makes a variation a *variation*.

For Spacer block variations, this is likely to be the `height` attribute. Suppose you wanted it to have a default of `180px`. For your variation, you'd need to set `attributes.height` and also check if `180px` is the value of the attribute in the `isActive` callback function.

Try this by pasting this code in your `block-variations.js` file:

```
const { registerBlockVariation } = wp.blocks;
const { __ } = wp.i18n;

registerBlockVariation( 'core/spacer', {
    name:       'themeslug/spacer',
    title:      __( 'Theme Name: Spacer', 'themeslug' ),
```

```
    keywords:   [ 'space', 'spacer', 'spacing' ],
    attributes: {
        height: '180px'
    },
    isActive: ( blockAttributes ) =>
        blockAttributes.height && '180px' === blockAttributes.height
} );
```

Each block will have its own attributes, so you will need to dive into the block's code to determine all of the available attributes you may want to overwrite for your variations. That is something beyond what can be covered in this doc, but the above code block should give you a solid starting point.

Unregistering block variations

To unregister a block variation, you must use the `unregisterBlockVariation()` JavaScript function. Here is a look at the function's signature:

```
const unregisterBlockVariation = ( blockName, variationName )
```

The function accepts two parameters:

- **`blockName`:** The name of the block (including namespace) for the variation you want to unregister.
- **`variationName`:** The name of the variation to unregister.

Suppose that you wanted to remove the Spacer block variation that you just added from the previous section. All you need to do is plug in the block and variation names.

Add this code at the end of your `block-variations.js` file to test it:

```
wp.domReady( () => {
    wp.blocks.unregisterBlockVariation(
        'core/spacer',
        'themeslug/spacer'
    );
} );
```

One important thing to note when unregistering variations is that you should wrap them in a `wp.domReady()` call. This is to ensure that the unregistering process happens later in the loading process after variations have already been registered.

□□□ #2
□ □□□ □□□ 2026-10-31 18:40:23 UTC

□ □□□ □□□ 2026-10-31 18:40:25 UTC