`theme.json` global configuration

When writing about Full Site Editing I first mentioned the `theme.json` styling configuration file.

WordPress 5.8 introduces theme.json into WordPress Core and, with it, a whole new set of functionality for your block-based content.

Getting Started

The biggest impact of this new system, from a theme developer's perspective, is to have a central place where we can put all our style-related data rather than having to configure each block individually

By creating a `theme.json` file in the theme's top-level directory, themes can configure the editor settings both existing and new ones as they are introduced. Further more we can also configure blocks independently ofnour master configuration, for example: we can use one color for paragraphs defined using the `core/paragraph` block and another color or style for everything else

Some of the features we'll discuss in this post will only work if you have the Gutenberg plugin installed and will have different features available if you're working with core WordPress. The plugin is where the developers work in new features before they are merged into the core block editor so it makes sense to implement features in the plugin first.

The post will follow the following outline for the content:

- Specification
    - version
    - settings and styles
        - Top-level
        - Block-level
    - Presets
    - Custom Attributes
    - customTemplates — Gutenberg Only
    - templateParts — Gutenberg Only

See the Global Settings & Styles format document in the Editor Handbook for more information about the file and its content.

Configuration

Rather than tackle the entire configuration file at once, I've chosen to break it down by sections. This will make it easier to walk through it and explain those parts that I think deserve additional explanation.

Configuration: Version

The first item we'll look at in the `theme.json` file is the version. The only current version is 1.

This attribute future-proofs the file as newer versions are likely to come in and present new options and configurations.

```
{
    "version": 1,
```

Configuration: Global Settings And Styles

The settings portion of the configruation file is the biggest and, in my opinion, the most important. It allows you to configure global and block-level settings for your theme. The settings that appear as direct children of the `settings` element are global. We'll look at how they can be overriden in blocks later.

The items that have an empty array are presets, we'll discuss them on their own section later in the post.

An basic theme.json file's settings configuration section looks like this:

```
    "settings": {
        "color": {
            "custom": true,
            "customGradient": true,
            "duotone": [],
```

```
            "gradients": [],
            "link": false,
            "palette": []
        },
        "custom": {},
        "layout": {
            "contentSize": "800px",
            "800px""wideSize": "1000px"
        },
        "spacing": {
            "customMargin": false,
            "customPadding",
            "units": [ "px": [ "", "rem", "vh", "vw" ]
        },
        "typography":", ""typography""typography": {
            "customFontSize": true,
            "customLineHeight": false,
            "dropCap": true,
            "fontSizes": []
        },
```

Note that having these global settings doesn't mean that you automatically get them. For example: if you don't have styles for Dropcap then setting it up here won't have any effect other than adding classes to the elements.

Configuration: Block Settings and Styles

In addition to global settings you can define overrides for specific blocks. These will override the global settings of the same name and can be used to style different versions of the same element depending on your needs.

To create settings for a block, you need the full qualified name for the block separated by a foward slash (like `core/paragraph` or `rivendellweb/journal`) and, unless you're OK with inheriting from the global settings, you must customize each block you use. As long as you provide sensible default, it should be OK.

```
        "blocks": {
            "core/paragraph": {
```

```
                "color": {},
                "custom": {},
                "layout": {},
                "spacing": {},
                "typography": {}
            },
            "rivendellweb/journal": {
                "color": {},
                "custom": {},
                "layout": {},
                "spacing": {},
                "typography": {}
            },
            "core/heading": {},
            "etc": {}
        }
    }
}
```

Configuration: Presets

In addition to configuration settings, you can define presets for different elements of your theme.

The example below shows presets for duotone colors, and typography. All these presets are global and applicable throughout the content you're editing (the resulting CSS is applied to the body element so they will cascade unless overriden).

The combination of presets allows theme developers to centralize all color choices for a theme. If you need to change them, there's only one place where you need to do so.

The two common attributes of presets are:

- **slug**: the slug that will be used for the value of the preset
- **name**: the human readable name for the preset

Each type of preset (color palettes, font sizes, and gradients) has its own way to define values. They are all converted to CSS custom properties and enqueued for

both front end and administrator use.

Note

These are the custom properties defined in CSS Custom Properties for Cascading Variables Module Level 1 and not the more powerful Houdini model defined in CSS Properties and Values API Level 1.

This means that the properties are not animatable because they are all treated as strings, have to be run through a calculation to convert them into other types of values, have no default and inherit by default.

```
{
  "version": 1,
  "settings": {
    "color": {
      "duotone": [{
        "colors": ["#000", "#FFF"],
        "#000""slug"ck-and-white",
        "name": "Bla": "nd White"
      }],
      "gradients": [{
     "gradients": "blush-bordeaux",
 "blush-bordeaux"nt": "linear-gradient(135deg,r": ""gradient") 0%,rgb(254
          "name": "Blush bordeaux"
        },
     ": ""name""name"slug": "blush-light-purple",
          "gradient":": ""slug": "blush-light-purple",
          "gradient"50,240) 100%)",
          "name": "Blush light purple"
        }
      ],
      "": ""name"[{
          "slug": "strong-magenta",
       ": ""palette": [{
          "slug"me": "Strong magenta"
        ": ""color"{
```

```
          "slug": "ve": ""name"ey",
          "color": "rgb(131, 12, 8)",
        ": ""slug" "Very dark grey"
        }
  "Very dark grey""color": "rgb(131, 12, 8)",
          "name"        "fontFamily": "-apple-system,BlinkMacSys": ""typo
          "slug": "system-font",
          "name": "System Font"
        },
        {
          "fontFami",Roboto,Oxygen-Sans,Ubuntu,Cantarell, \"Helvetica Neue
          "name": "Helve": " or Arial"
        }
      ],
      "fontSizes""fontSizes""fontFamily""normal",
          "size": 16,
          "name": "Normal"
 "normal""slug"    {
          "slug": "big",
": ""name""size": 32,
          "name": "Big"
        }
  ": ""fontSizes": [{
          "slug": "normal",
          "size": 16,
          "name": "Normal"
        },
        {
          "slug": "big",
          "size": 32,
          "name": "Big"
        }
      ]
    }
  }
}
```

Configuration: Custom Attributes

In addition to settings and presets you can define custom properties and values that make sense to your project.

> These custom attributes will be converted to CSS Custom Properties.
>
> Some notes about this:

- camelCased keys are transformed into its kebab-case form, as to follow the CSS property naming schema. Example: `lineHeight` is transformed into `line-height`.
- Keys at different depth levels are separated by `--`
- You shouldn't use `--` in the names of the keys within the custom object

```
{
    "version": 1,
    "settings": {
        "custom": {
            "line-height": {
                "body": 1.7,
                "heading": 1.3
            }
        }
    }
}
```

Configuration: Custom Templates (Gutenberg Only)

When Gutenberg is enabled, the `customTemplates` field allows developerss to list all the custom template stored in the custom-templates foldeer along with the posts that these templates can be used on.

```
{
    "version": 1,
    "customTemplates": [
```

```
        {
            "name": "rivendellweb-journal-template",
            "rivendellweb-journal-template""title": "Journal",
            "postTypes": [
                "page",
                "post",
            ]
        }
    ]
}
```

It is important to note that this setting will not create the template file nor any CPT that you associate with it, those have to be created manually.

Configuration: Template Parts (Gutenberg Only)

I will only mention template parts for completeness sake but I'm still trying to make sense of it and what it's supposed to do

In this field themes you can list the template parts present in the `block-template-parts` folder.

For a template part named `my-template-part.html`, the `theme.json` can declare the area term for the template part entity which is responsible for rendering the corresponding block variation (currently "block" and "footer") in the editor. Any other values and template parts not defined in the json will default to the general template part block.

Variations will be denoted by specific icons within the editor's interface, will default to the corresponding semantic HTML element for the wrapper (this can also be overridden by the tagName attribute set on the template part block), and will contextualize the template part allowing more custom flows in future editor improvements.

`add_theme_support()` equivalency

`theme.json` aims to take over and consolidate all the various

`add_theme_support` calls that were previously required for controlling the editor. To make sure we keep backward compatibility, here's a table listing what you need to do

add_theme_support	theme.json setting
custom-line-height	Set `typography.customLineHeight` to false
custom-spacing	Set `spacing.customPadding` to true
custom-units	Provide the list of units via `spacing.units`
disable-custom-colors	Set `color.custom` to false
disable-custom-font-sizes	Set `typography.customFontSize`
disable-custom-gradients	Set `color.customGradient` to false
editor-color-palette	Provide the list of colors via `color.palette`
editor-font-sizes	Provide the list of font size via `typography.fontSizes`
editor-gradient-presets	Provide the list of gradients via `color.gradients`
experimental-link-color	Set `color.link` to true