Typography

- Enabling and disabling typography options
- Custom font families
  - Registering system fonts
  - Registering web fonts (font faces)
- Custom font sizes
  - Enabling fluid typography
  - Registering custom font size presets

Typography is a wide-ranging subject in web design, and no single piece of documentation could do it justice. There are book-length works that dive into the finer details and articles aplenty across the web that teach best practices, tips, and tricks at length.

This guide will specifically get you up to speed with the available settings available via `settings.typography` in `theme.json`.

Typography is also tightly related to spacing in your theme, so you will need to familiarize yourself with the spacing settings to get the most out of both guides. You will also take what you learn from this documentation and apply it to your `theme.json` styles.

Enabling and disabling typography options

There are several settings that are merely for enabling or disabling elements in the user interface. Each of these properties accept a boolean value, meaning that you can set them to either `true` or `false`.

Not every block will support every typography setting. Some are specific to a single block. For example, the `dropCap` property only works for Paragraph blocks. Others are widely used by almost any block with text, such as `customFontSize`.

These `settings.typography` properties let you enable or disable features in the interface:

- `customFontSize`: Whether to let users input custom font sizes. Defaults to `true`.
- `dropCap`: Whether users can enable a drop-cap for the first letter of the Paragraph block. Defaults to `true`.
- `fontStyle`: Whether users can select a custom font style. Defaults to `true`.
- `fontWeight:` Whether users can select a custom font weight. Weight ranges map to the standard **Thin** through **Black** font weights. Defaults to `true`.
- `letterSpacing`: Whether users can input a custom letter spacing value. Defaults to `false`.
- `lineHeight`: Whether users can input a custom line height for text. There is no way to register line-height presets, so this option enables a completely custom input. Defaults to `false`.
- `textColumns`: Whether to show a columns option for the block's text. Defaults to `false`.
- `textDecoration`: Whether the user can set the text decoration for a block's text. The available options are **None**, **Underline**, and **Strikethrough**. Defaults to `true`.
- `textTransform`: Whether the user can change the letter case for a block's text. The available options are **None**, **Uppercase**, **Lowercase**, and **Capitalize**. Defaults to `true`.
- `writingMode`: Whether to enable the text **Orientation** in the interface, allowing users to choose between **Horizontal** and **Vertical** text. Defaults to `false`.

Here is what these typography settings look like in the default WordPress `theme.json`:

```
{
	"version": 2,
	"settings": {
		"typography": {
			"customFontSize": true,
			"dropCap": true,
			"fontStyle": true,
			"fontWeight": true,
			"letterSpacing": true,
			"lineHeight": false,
			"textColumns": false,
			"textDecoration": true,
			"textTransform": true,
			"writingMode": false
		}
	}
}
```

In this screenshot, take note of the **Typography** panel for the Paragraph block, which has every setting enabled:

WordPress post editor with a Heading and multiple Paragraph blocks inserted. In the right sidebar, n

Depending on your theme's audience, you will want to enable only the features that its users will need. For example, you may have a client who only needs to configure the font size. It wouldn't make sense to show them every setting if it gets in the way of their workflow.

A note on drop caps: the current implementation feature is notoriously buggy with different font families, font sizes, and line heights. If you leave this feature enabled, be sure that you test that drop caps look good within your theme's design. You may need to add custom CSS so that it matches your theme.

Custom font families

WordPress lets you register as many font families as you want via the `settings.typography.fontFamilies` property in `theme.json`. You can add support for both system fonts (those that live on the visitor's computer) or web fonts (custom fonts bundled with your theme). You'll learn how to register both of these in this section.

These appear as options under the **Typography > Font** field in the inspector panel for blocks that support selecting a font:

WordPress post editor with a Cover block with inner blocks nested. In the sidebar, the Font dropdow

The `settings.typography.fontFamilies` property is an empty array by default. You can register fonts by passing in font family objects with these properties:

- **`name`**: *(Required)* The human-readable title for the font family, which can be translated.
- **`slug`**: *(Required)* The slug for the font family, which will be appended to a generated CSS custom property: `--wp--preset--font-family--{slug}`.
- **`fontFamily`**: *(Required)* A valid value that will map to the CSS `font-family` value. Generally, this will be a font stack (a list of families that the browser will try to use in order).
- **`fontFace`**: *(Optional)* An array of font faces that are mapped to the `@font-face` CSS at-rule. You only need to use this when bundling custom web fonts with your theme.

Here is what the font family setting looks like in the default WordPress `theme.json`:

```
{
	"version": 2,
	"settings": {
		"typography": {
			"fontFamilies": []
		}
	}
}
```

In the following subsections, you will first learn how to register system fonts. Then, you will take the next step and register a custom web font.

Registering system fonts

System fonts are more straightforward than web fonts. You only need to know which fonts you want to use. For this exercise, let’s assume that you want to register two fonts that you will use in your theme:

- **Primary:** a transitional serif font stack that will look good across modern devices.
- **Secondary:** the user’s system UI font.

Add this code to your `theme.json` to register each of the fonts:

```
{
	"$schema": "https://schemas.wp.org/trunk/theme.json",
	"version": 2,
	"settings": {
		"typography": {
			"fontFamilies": [
				{
					"name": "Primary",
					"slug": "primary",
					"fontFamily": "Charter, 'Bitstream Charter', 'Sitka Text', Cambria, serif"
				},
				{
					"name": "Secondary",
					"slug": "secondary",
					"fontFamily": "system-ui, sans-serif"
				}
			]
		}
	}
}
```

These options will now appear in the **Typography > Font** dropdown for any blocks that support the feature.

WordPress will also generate two CSS custom properties with the `--wp--preset--font-family--{$slug}` format in both the editor and on the front end:

```
body {
--wp--preset--font-family--primary: Charter, 'Bitstream Charter', 'Sitka Text', Cambria,
serif;
--wp--preset--font-family--secondary: system-ui, sans-serif;
}
```

You can reference these as `var:preset|font-family|$slug` when you begin using them as Styles in `theme.json`. You can also reference them using `var( --wp--preset--font-family--{$slug} )` directly in CSS.

It is generally recommended to use a semantic naming scheme for your font family slugs so that they are more future proof when switching between child themes, style variations, and even from theme to theme. These examples use `primary` and `secondary` since those are widely used terms. It's best to avoid naming your slugs so that they match the current font family.

Registering web fonts (font faces)

Registering custom web fonts works in much the same way as system fonts. The big difference is that you need to add the `fontFace` property to your font family objects.

`fontFace` must be an array of font face objects. Each object accepts these properties that map to descriptors for the `@font-face` CSS at-rule:

- **`fontFamily`:** A valid CSS `font-family` descriptor.
- **`fontWeight`:** A range of CSS `font-weight` values.
- **`fontStyle`:** A valid CSS `font-style` value.
- **`fontStretch`:** A valid CSS font stretch value.
- **`src`:** An array of font file URLs where the font files are located (this can be used to support multiple formats, but only one is required).

The `src` property is unique in that it allows you to reference a URL that is relative to the `theme.json` file in your theme. Use the `"file:./path/to/file.ext"` format to reference a font bundled with your theme.

Let's try an example. Building off the code from the previous section, suppose you wanted to replace the system UI font with the Open Sans web font.

This example will also assume you have downloaded and converted the Open Sans font to the modern `.woff2` format, which is widely supported by most browsers. You've also placed these files in your theme's `/assets/fonts` folder like this:

- `/assets`
  - `/fonts`
    - `/open-sans.woff2`

- `/open-sans-italic.woff2`

With your web fonts bundled in your theme and ready to go, add this code to your `theme.json` file to register them:

```
{
	"$schema": "https://schemas.wp.org/trunk/theme.json",
	"version": 2,
	"settings": {
		"typography": {
			"fontFamilies": [
				{
					"name": "Primary",
					"slug": "primary",
					"fontFamily": "Charter, 'Bitstream Charter', 'Sitka Text', Cambria, serif"
				},
				{
					"name": "Secondary",
					"slug": "secondary",
					"fontFamily": "'Open Sans', sans-serif",
					"fontFace": [
						{
							"fontFamily": "Open Sans",
							"fontWeight": "300 800",
							"fontStyle": "normal",
							"fontStretch": "normal",
							"src": [ "file:./assets/fonts/open-sans.woff2" ]
						},
						{
							"fontFamily": "Open Sans",
							"fontWeight": "300 800",
							"fontStyle": "italic",
							"fontStretch": "normal",
							"src": [ "file:./assets/fonts/open-sans-italic.woff2" ]
						}
					]
				}
			]
		}
	}
}
```

```
}
```

Like with system fonts, WordPress will generate CSS custom properties for web fonts. Here is what the CSS output looks like:

```
body {
--wp--preset--font-family--primary: Charter, 'Bitstream Charter', 'Sitka Text', Cambria,
serif;
--wp--preset--font-family--secondary: 'Open Sans', sans-serif;
}
```

Custom font sizes

WordPress lets you register any number of preset font sizes that your theme users can choose from. By registering custom sizes, you can more easily maintain consistent typography across the theme.
Custom font sizes appear as options in the **Typography > Size** panel in the inspector controls for blocks that support the feature:

WordPress post editor showing multiple paragraphs with different font sizes.

Note that if you have more than five custom sizes, the UI will show a dropdown select instead of the button group shown in the above screenshot.

There are two `theme.json.settings.typography` sub-properties related font sizes:

- **`fluid`:** Whether to enable fluid font sizes that scale up on large screens and down on smaller screens. This can be overridden for individual custom sizes. Defaults to `false`.
- **`fontSizes`:** An array of font size objects that you can use to customize the available presets that users can choose from. WordPress registers the default sizes of `small`, `medium`, `large`, and `x-large`.

Here is what these properties look like in the default WordPress `theme.json`:

```
{
"version": 2,
"settings": {
"typography": {
"fluid": false,
"fontSizes": [
{
"name": "Small",
```

```
□□□□□"slug": "small",
□□□□□"size": "13px"
□□□□},
□□□□{
□□□□□"name": "Medium",
□□□□□"slug": "medium",
□□□□□"size": "20px"
□□□□},
□□□□{
□□□□□"name": "Large",
□□□□□"slug": "large",
□□□□□"size": "36px"
□□□□},
□□□□{
□□□□□"name": "Extra Large",
□□□□□"slug": "x-large",
□□□□□"size": "42px"
□□□□}
□□□]
□□}
□}
}
```

As you can see, WordPress disables fluid font sizes by default and registers four static sizes of its own. In almost all cases, you will want to overwrite these with sizes that match your theme's design.

Like other presets that you can set in `theme.json`, WordPress automatically generates CSS custom properties for font sizes using the `--wp--preset--font-size--{$slug}` format. For the default sizes, this CSS is output in the editor and on the front end:

```
body {
□--wp--preset--font-size--small: 13px;
□--wp--preset--font-size--medium: 20px;
□--wp--preset--font-size--large: 36px;
□--wp--preset--font-size--x-large: 42px;
}
```

WordPress also generates custom CSS classes for each font size preset using the `.has-{$slug}-font-size` naming scheme. The default sizes produce these classes:

```
.has-small-font-size{
	font-size: var(--wp--preset--font-size--small) !important;
}

.has-medium-font-size{
	font-size: var(--wp--preset--font-size--medium) !important;
}

.has-large-font-size{
	font-size: var(--wp--preset--font-size--large) !important;
}

.has-x-large-font-size{
	font-size: var(--wp--preset--font-size--x-large) !important;
}
```

Enabling fluid typography

In modern web design, you will almost always utilize fluid typography. This allows your font sizes to scale up or down with the viewport or container.

WordPress uses a viewport-based system for fluid typography. But you don’t have to rely on this if you prefer to use something different, such as a container-based system, and can leave it disabled.

Suppose you wanted to use the default core sizes but wanted them all to be fluid instead of static values. You can enable this by setting `settings.typography.fluid` to `true` in your `theme.json` file:

```
{
	"version": 2,
	"settings": {
		"typography": {
			"fluid": true
		}
	}
}
```

WordPress will now generate fluid sizes for registered font sizes:

```
body {
	--wp--preset--font-size--small: 13px;
```

```
--wp--preset--font-size--medium: clamp(14px, 0.875rem + ((1vw - 3.2px) * 0.852), 20px);
--wp--preset--font-size--large: clamp(22.041px, 1.378rem + ((1vw - 3.2px) * 1.983), 36px);
--wp--preset--font-size--x-large: clamp(25.014px, 1.563rem + ((1vw - 3.2px) * 2.413), 42px);
}
```

`14px` is the default minimum font-size limit. Because the `small` size is below that limit, it remained at its static size of `13px`.

Registering custom font size presets

You will undoubtedly want to register font sizes that match your theme's design instead of sticking with the defaults. To do this, you must pass an array of font size objects to the `settings.typography.fontSizes` property in `theme.json`.

Each of these objects accepts several properties:

- **`name`:** The human-readable title for the size, which can be translated.
- **`size`:** A valid CSS size. This can be a number and unit, a custom fluid size using `clamp()`, or a reference to another custom CSS property.
- **`slug`:** The slug for the size, which will be appended to a generated CSS custom property: `--wp--preset--spacing--{slug}`.
- **`fluid`:** A boolean value to enable or disable fluid typography for this specific size, overwriting the global property. Alternatively, it can be an object containing:
  - **`min`:** The minimum value that the font size can scale down to. Must be a valid CSS size.
  - **`max`:** The maximum value that the font size can scale up to. Must be a valid CSS size.

Let's register a simple set of custom sizes with static values first. Add this code to your `theme.json` file:

```
{
"$schema": "https://schemas.wp.org/trunk/theme.json",
"version": 2,
"settings": {
"typography": {
"fontSizes": [
{
"name": "Small",
"size": "1rem",
"slug": "sm"
},
```

```
        {
            "name": "Medium",
            "size": "1.25rem",
            "slug": "md"
        },
        {
            "name": "Large",
            "size": "1.5rem",
            "slug": "lg"
        }
    ]
  }
}
```

Just like with the default size presets, WordPress will automatically generate the CSS for each in both the editor and on the front end:

```
body {
  --wp--preset--font-size--sm: 1rem;
  --wp--preset--font-size--md: 1.25rem;
  --wp--preset--font-size--lg: 1.5rem;
}
```

Now, let's take this one step further. In the previous example, you registered Small, Medium, and Large sizes. Suppose that you wanted the Small size to maintain its value regardless of the browser's viewport width. But you want the Medium and Large size to scale along with the viewport.

Using the `fluid` property, let's configure these on a size-by-size basis. Add this code to your `theme.json`:

```
{
  "$schema": "https://schemas.wp.org/trunk/theme.json",
  "version": 2,
  "settings": {
    "typography": {
      "fluid": true,
      "fontSizes": [
        {
          "name": "Small",
```

```
□□□□□"size": "1rem",
□□□□□"slug": "sm",
□□□□□"fluid": false
□□□□},
□□□□{
□□□□□"name": "Medium",
□□□□□"size": "1.25rem",
□□□□□"slug": "md",
□□□□□"fluid": {
□□□□□□"min": "1rem",
□□□□□□"max": "1.5rem"
□□□□□}
□□□□},
□□□□{
□□□□□"name": "Large",
□□□□□"size": "1.5rem",
□□□□□"slug": "lg",
□□□□□"fluid": {
□□□□□□"min": "1.25rem",
□□□□□□"max": "2rem"
□□□□□}
□□□□}
□□□]
□□}
□}
}
```

WordPress will generate these CSS custom properties:

```
body {
□--wp--preset--font-size--sm: 1rem;
□--wp--preset--font-size--md: clamp(1rem, 1rem + ((1vw - 0.2rem) * 1.136), 1.5rem);
□--wp--preset--font-size--lg: clamp(1.25rem, 1.25rem + ((1vw - 0.2rem) * 1.705), 2rem);
}
```

There aren’t many limits to how you can customize this for your theme. The above offers an intro to registering custom font sizes, but you have full control over how this works for your theme.

For a deeper understanding of fluid font sizes, read Intrinsic design, theming, and rethinking how to design with WordPress on the Developer Blog.

□□□ #2
□ □□□ □□□ 2026-08-29 15:36:32 UTC
□ □□□ □□□ 2026-08-29 15:36:32 UTC