External Project Templates

- Project Template Configuration
  - pluginTemplatesPath
  - blockTemplatesPath
  - assetsPath
  - defaultValues
    - transformer examples
  - variants

Are you looking for a way to share your project configuration? Creating an external project template hosted on npm or located in a local directory is possible. These npm packages can provide custom `.mustache` files that replace default files included in the tool for the WordPress plugin or/and the block. It's also possible to override default configuration values used during the scaffolding process.

Project Template Configuration

Providing the main file (`index.js` by default) for the package that returns a configuration object is mandatory. Several options allow customizing the scaffolding process.

`pluginTemplatesPath`

This optional field allows overriding file templates related to **the WordPress plugin shell**. The path points to a location with template files ending with the `.mustache` extension (nested folders are also supported). When not set, the tool uses its own set of templates.

Example:

```
const { join } = require( 'path' );

module.exports = {
    pluginTemplatesPath: join( __dirname, 'plugin-templates' ),
```

```
};
```

`blockTemplatesPath`

This optional field allows overriding file templates related to **the individual block**. The path points to a location with template files ending with the `.mustache` extension (nested folders are also supported). When not set, the tool uses its own set of templates.

Example:

```
const { join } = require( 'path' );

module.exports = {
    blockTemplatesPath: join( __dirname, 'block-templates' ),
};
```

`assetsPath`

This setting is useful when your template scaffolds a WordPress plugin that uses static assets like images or fonts, which should not be processed. It provides the path pointing to the location where assets are located. They will be copied to the `assets` subfolder in the generated plugin.

Example:

```
const { join } = require( 'path' );

module.exports = {
    assetsPath: join( __dirname, 'plugin-assets' ),
};
```

`defaultValues`

It is possible to override the default template configuration using the `defaultValues` field.

Example:

```
module.exports = {
    defaultValues: {
        slug: 'my-fantastic-block',
        title: 'My fantastic block',
```

```
        dashicon: 'palmtree',
        version: '1.2.3',
    },
};
```

The following configurable variables are used with the template files. Template authors can change default values to use when users don't provide their data.

Project:

- `wpScripts` (default: `true`) – enables integration with the `@wordpress/scripts` package and adds common scripts to the `package.json`.
- `wpEnv` (default: `false`) – enables integration with the `@wordpress/env` package and adds the `env` script to the `package.json`.
- `customScripts` (default: {}) – the list of custom scripts to add to `package.json` . It also allows overriding default scripts.
- `npmDependencies` (default: `[]`) – the list of remote npm packages to be installed in the project with `npm install` when `wpScripts` is enabled.
- `npmDevDependencies` (default: `[]`) – the list of remote npm packages to be installed in the project with `npm install --save-dev` when `wpScripts` is enabled.
- `customPackageJSON` (no default) – allows definition of additional properties for the generated package.json file.

Plugin header and readme fields (learn more about header requirements and readmes):

- `pluginURI` (no default) – the home page of the plugin.
- `version` (default: `'0.1.0'`) – the current version number of the plugin.
- `requiresAtLeast` (default: `'6.8'`) – the lowest WordPress version that the plugin will work on.
- `requiresPHP` (default: `'7.4'`) – the minimum required PHP version for use with this plugin.
- `testedUpTo` (default: `'6.8'`) – the highest WordPress version that the plugin has been tested against.
- `author` (default: `'The WordPress Contributors'`) – the name of the plugin author(s).
- `license` (default: `'GPL-2.0-or-later'`) – the short name of the plugin's license.
- `licenseURI` (default: `'https://www.gnu.org/licenses/gpl-2.0.html'`) – a link to the full text of the license.
- `domainPath` (no default) – a custom domain path for the translations (more info).
- `updateURI:` (no default) – a custom update URI for the plugin (related dev note).

Block metadata (learn more):

- `folderName` (default: `src`) – the location for the `block.json` file and other optional block files generated from block templates included in the folder set with the `blockTemplatesPath` setting.

- `$schema` (default: `https://schemas.wp.org/trunk/block.json`) – the schema URL used for block validation.
- `apiVersion` (default: `2`) – the block API version (related dev note).
- `slug` (no default) – the block slug used for identification in the block name.
- `namespace` (default: `'create-block'`) – the internal namespace for the block name.
- `title` (no default) – a display title for your block.
- `description` (no default) – a short description for your block.
- `dashicon` (no default) – an icon property that makes it easier to identify a block (available values).
- `category` (default: `'widgets'`) – blocks are grouped into categories to help users browse and discover them. The categories provided by core are `text`, `media`, `design`, `widgets`, `theme`, and `embed`.
- `textdomain` (defaults to the `slug` value) – the text domain used to make strings translatable (more info).
- `attributes` (no default) – block attributes (more details).
- `supports` (no default) – optional block extended support features (more details.
- `editorScript` (default: `'file:./index.js'`) – an editor script definition.
- `editorStyle` (default: `'file:./index.css'`) – an editor style definition.
- `style` (default: `'file:./style-index.css'`) – a frontend and editor style definition.
- `render` (no default) – a path to the PHP file used when rendering the block type on the server before presenting on the front end.
- `customBlockJSON` (no default) – allows definition of additional properties for the generated block.json file.
- `transformer` (default: `( view ) => view`) – a function that receives all variables generated by the create-block tool and returns an object of values. This function provides the ability to modify existing values and add new variables.

`transformer` examples

This examples adds a generated value to the slug variable.

```
transformer: ( view ) => {
    const hex = getRandomHexCode();
      return {
         ...view,
         slug: `${ view.slug }-${ hex }`,
      };
},
```

This example creates a new custom variable that can be used in the associated mustache templates as `{{customVariable}}`

```
transformer: ( view ) => {
      return {
         ...view,
         customVariable: `Custom Value`,
      };
},
```

`variants`

Variants are used to create variations for a template. Variants can override any `defaultValues` by providing their own.

```
module.exports = {
    defaultValues: {
        slug: 'my-fantastic-block',
        title: 'My fantastic block',
        dashicon: 'palmtree',
        version: '1.2.3',
    },
    variants: {
        primary: {},
        secondary: {
            title: 'My fantastic block - secondary variant',
        },
    },
};
```

Variants are accessed using the `--variant` flag, i.e `--variant secondary`.

If no variant is provided, the first variant is used if any are defined.

Mustache variables are created for variants that can be used to conditionally output content in files. The format is `{{isVARIANT_NAMEVariant}}`.

```
{{#isPrimaryVariant}}
This content is only rendered if `--variant primary` is passed.
{{/isPrimaryVariant}}

{{#isSecondaryVariant}}
This content is only rendered if `--variant secondary` is passed.
{{/isSecondaryVariant}}
```

Variants can also define their own files by defining `pluginTemplatesPath`, `blockTemplatesPath`, or `assetsPath`. If these are defined, they will override the paths defined by the project template. In the case that a variant doesn’t need some of the files defined by the template, `null` can be passed to the appropriate variable to skip scaffolding those files.

```
module.exports = {
    defaultValues: {
        slug: 'my-fantastic-block',
        title: 'My fantastic block',
        dashicon: 'palmtree',
        version: '1.2.3',
    },
    variants: {
        primary: {},
        secondary: {
            title: 'My fantastic block - secondary variant',
            blockTemplatesPath: join(
                __dirname,
                'custom-path',
                'block-templates'
            ),
            assetsPath: null, // Will not scaffold any assets files even if defined by the
main template.
        },
    },
};
```

□□□ #2
□ □□□ □□□ 2026-10-31 18:23:01 UTC
□ □□□ □□□ 2026-10-31 18:23:03 UTC