Core Gatsby APIs

Themes are packaged Gatsby sites shipped as plugins, so you have access to all of Gatsby's APIs for modifying default configuration settings and functionality.

- Gatsby Config
- Actions
- Node Interface
- ... and more

If you're new to Gatsby you can get started by following along with the guides for building out a site. Converting it to a theme will be straightforward later on since themes are prepackaged Gatsby sites.

Configuration

Plugins can now include a `gatsby-config` in addition to the other `gatsby-*` files. We typically refer to plugins that include a `gatsby-config.js` as a theme (more on that in theme composition). A typical `gatsby-config.js` in a user's site that uses your theme could look like this. This example passes in two options to `gatsby-theme-name` : `postsPath` and `colors` .

```
module.exports = {
  plugins: [
    {
      resolve: "gatsby-theme-name",
      options: {
        postsPath: "/blog",
        colors: {
          primary: "tomato",
        },
      },
    },
  ],
}
```

You can access options that are passed to your theme in your theme's `gatsby-config` . You can use options to make filesystem sourcing configurable, accept different nav menu items, change branding colors from the default, and anything else you want to make configurable.

To take advantage of the options that are passed in when configuring your theme in a user's site, return a function in your theme's `gatsby-config.js` . The argument the function receives is the options the user passed in.

```
module.exports = themeOptions => {
  console.log(themeOptions)
  // logs `postsPath` and `colors`

  return {
    plugins: [
      // ...
    ],
  }
}
```

While using the usual object export (`module.exports = {}`) in your theme means that you can run the theme standalone as its own site, when using a function in your theme to accept options you will need to run the theme as part of an example site. See how the [theme authoring starter]() handles this using Yarn Workspaces.

Accessing options elsewhere

Note that because themes are plugins you can also access the options in any of the lifecycle methods that you're used to. For example, in your theme's `gatsby-node.js` you can access the options as the second argument to `createPages` :

```
exports.createPages = async ({ graphql, actions }, themeOptions) => {
  console.log(themeOptions)
}
```

Shadowing

Since themes are usually deployed as npm packages that other people use in their sites, you need a way to modify certain files, such as React components, without making changes to the source code of the theme. This is called *Shadowing*.

Shadowing is a filesystem-based API that allows us to replace one file with another at build time. For example, if you had a theme with a `Header` component you could replace that `Header` with your own by creating a new file and placing it in the correct location for Shadowing to find it.

Overriding

Taking a closer look at the `Header` example, let's say you have a theme called `gatsby-theme-amazing` . That theme uses a `Header` component to render navigation and other miscellaneous items. The path to the component from the root of the npm package is `gatsby-theme-amazing/src/components/header.js` .

You might want the `Header` component to do something different (maybe change colors, maybe add additional navigation items, really anything you can think of). To do that, you create a file in your site at `src/gatsby-theme-amazing/components/header.js` . You can now export any React component you want from this file and Gatsby will use it instead of the theme's component.

> 💡 *Note: you can shadow components from other themes using the same method. Read more about advanced applications in [latent shadowing]().*

Extending

In the last section we talked about completely replacing one component with another. What if you want to make a smaller change that doesn't require copy/pasting the entire theme component into your own? You can take advantage of the ability to extend components.

Taking the `Header` example from before, when you write your shadowing file at `src/gatsby-theme-amazing/components/header.js` , you can import the original component and re-export it as such, adding your own overridden prop to the component.

```
import Header from "gatsby-theme-amazing/src/components/header"

// these props are the same as the original component would get
```

```
export default function MyHeader(props) {
  return <Header {...props} myProp="true" />
}
```

Taking this approach means that when you upgrade your theme later you can also take advantage of all the updates to the `Header` component because you haven't fully replaced it, just modified it.

What path should be used to shadow a file?

Until Gatsby has tooling to automatically handle shadowing, you will have to manually locate paths in a theme and create the correct shadowing paths in your site.

Luckily, the way to do that is only a few steps. Take the `src` directory from the theme, and move it to the front of the path, then write a file at that location in your site. Looking back on the `Header` example, this is the path to the component in your theme:

```
gatsby-theme-amazing/src/components/header.js
```

and here is the path where you would shadow it in your site:

```
<your-site>/src/gatsby-theme-amazing/components/header.js
```

Shadowing only works on imported files in the `src` directory. This is because shadowing is built on top of webpack, so the module graph needs to include the shadowable file.

Since you can use multiple themes in a given site, there are many potential places to shadow a given file (one for each theme and one for the user's site). In the event that multiple themes are attempting to shadow `gatsby-theme-amazing/src/components/header.js`, the last theme included in the plugins array will win. The site itself takes the highest priority in shadowing.

Theme composition

Gatsby themes can compose horizontally and vertically. Vertical composition refers to the classic "parent/child" relationship. A child theme declares a parent theme in the child theme's plugins array.

```
module.exports = {
  plugins: [`gatsby-theme-parent`],
}
```

Horizontal composition is when two different themes are used together, such as `gatsby-theme-blog` and `gatsby-theme-notes`.

```
module.exports = {
  plugins: [`gatsby-theme-blog`, `gatsby-theme-notes`],
}
```

Themes at their core are an algorithm that merges multiple `gatsby-config.js` files together into a single config your site can use to build with. To do that you need to define how to combine two `gatsby-config.js` s together.

Before you can do that, you need to flatten the parent/child relationships into a single array. This results in the final ordering when considering which shadowing file to use if multiple are available.

The first example results in a final ordering of `['gatsby-theme-parent', 'gatsby-theme-child']` (parents always come before their children so that children can override functionality), while the second example results in `['gatsby-theme-blog', 'gatsby-theme-notes']`.

Once you have the final ordering of themes you merge them together using a reduce function. This reduce function specifies the way each key in `gatsby-config.js` will merge together. Unless otherwise specified below, the last value wins.

- `siteMetadata` and `mapping` both merge deeply using Lodash's `merge` function. This means a theme can set default values in `siteMetadata` and the site can override them using the standard `siteMetadata` object in `gatsby-config.js`.
- `plugins` are normalized to remove duplicates, then concatenated together.