> *Note: this feature is available with* `react-scripts@0.5.0` *and higher.*

Changing the HTML

The `public` folder contains the HTML file so you can tweak it, for example, to set the page title. The `<script>` tag with the compiled code will be added to it automatically during the build process.

Adding Assets Outside of the Module System

You can also add other assets to the `public` folder.

Note that we normally encourage you to `import` assets in JavaScript files instead. For example, see the sections on adding a stylesheet and adding images and fonts. This mechanism provides a number of benefits:

- Scripts and stylesheets get minified and bundled together to avoid extra network requests.
- Missing files cause compilation errors instead of 404 errors for your users.
- Result filenames include content hashes so you don't need to worry about browsers caching their old versions.

However there is an **escape hatch** that you can use to add an asset outside of the module system.

If you put a file into the `public` folder, it will **not** be processed by webpack. Instead it will be copied into the build folder untouched. To reference assets in the `public` folder, you need to use an environment variable called `PUBLIC_URL`.

Inside `index.html`, you can use it like this:

```
<link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
```

Only files inside the `public` folder will be accessible by `%PUBLIC_URL%` prefix. If you need to use a file from `src` or `node_modules`, you'll have to copy it there to explicitly specify your intention to make this file a part of the build.

When you run `npm run build`, Create React App will substitute `%PUBLIC_URL%` with a correct absolute path so your project works even if you use client-side routing or host it at a non-root URL.

In JavaScript code, you can use `process.env.PUBLIC_URL` for similar purposes:

```
render() {
  // Note: this is an escape hatch and should be used sparingly!
  // Normally we recommend using `import` for getting asset URLs
  // as described in "Adding Images and Fonts" above this section.
  return <img src={process.env.PUBLIC_URL + '/img/logo.png'} />;
}
```

Keep in mind the downsides of this approach:

- None of the files in `public` folder get post-processed or minified.
- Missing files will not be called at compilation time, and will cause 404 errors for your users.
- Result filenames won't include content hashes so you'll need to add query arguments or rename them every time they change.

When to Use the `public` Folder

Normally we recommend importing stylesheets, images, and fonts from JavaScript. The `public` folder is useful as a workaround for a number of less common cases:

- You need a file with a specific name in the build output, such as `manifest.webmanifest`.
- You have thousands of images and need to dynamically reference their paths.
- You want to include a small script like `pace.js` outside of the bundled code.
- Some libraries may be incompatible with webpack and you have no other option but to include it as a `<script>` tag.

Note that if you add a `<script>` that declares global variables, you should read the topic Using Global Variables in the next section which explains how to reference them.