eleventy-plugin-bundle

Little bundles of code, little bundles of joy.

Create minimal per-page or app-level bundles of CSS, JavaScript, or HTML to be included in your Eleventy project.

Makes it easy to implement Critical CSS, in-use-only CSS/JS bundles, SVG icon libraries, or secondary HTML content to load via XHR.

Why?

This project is a minimum-viable-bundler and asset pipeline in Eleventy. It does not perform any transpilation or code manipulation (by default). The code you put in is the code you get out (with configurable `transforms` if you'd like to modify the code).

For more larger, more complex use cases you may want to use a more full featured bundler like Vite, Parcel, Webpack, rollup, esbuild, or others.

But do note that a full-featured bundler has a significant build performance cost, so take care to weigh the cost of using that style of bundler against whether or not this plugin has sufficient functionality for your use case—especially as the platform

matures and we see diminishing returns on code transpilation (ES modules everywhere).

Installation

No installation necessary. Starting with Eleventy `v3.0.0-alpha.10` and newer, this plugin is now bundled with Eleventy.

Usage

By default, Bundle Plugin v2.0 does not include any default bundles. You must add these yourself via `eleventyConfig.addBundle`. One notable exception happens when using the WebC Eleventy Plugin, which adds `css`, `js`, and `html` bundles for you.

To create a bundle type, use `eleventyConfig.addBundle` in your Eleventy configuration file (default `.eleventy.js`):

```
// .eleventy.js
export default function(eleventyConfig)
        {
    eleventyConfig.addBundle("css");
};
```

This does two things:

1. Creates a new `css` shortcode for adding arbitrary code to this bundle
2. Adds `"css"` as an eligible type argument to the `getBundle` and `getBundleFileUrl` shortcodes.

Full options list

```
export default function(eleventyConfig)
        {
    eleventyConfig.addBundle("css", {
        // (Optional) Folder (relative
        to output directory) files will
        write to
        toFileDirectory: "bundle",

        // (Optional) File extension
        used for bundle file output,
        defaults to bundle name
        outputFileExtension: "css",

        // (Optional) Name of shortcode
        for use in templates, defaults
        to bundle name
        shortcodeName: "css",
        // shortcodeName: false, //
        disable this feature.

        // (Optional) Modify bundle
        content
        transforms: [],
```

```
        // (Optional) If two identical
        code blocks exist in non-default
        buckets, they'll be hoisted to
        the first bucket in common.
        hoist: true,

        // (Optional) In 11ty.js
        templates, having a named export
        of `bundle` will populate your
        bundles.
        bundleExportKey: "bundle",
        // bundleExportKey: false, //
        disable this feature.
    });
};
```

Read more about [hoist and duplicate bundle hoisting].

Universal Shortcodes

The following Universal Shortcodes (available in `njk`, `liquid`, `hbs`, `11ty.js`, and `webc`) are provided by this plugin:

- `getBundle` to retrieve bundled code as a string.
- `getBundleFileUrl` to create a bundle file on disk and retrieve the URL to that file.

Here's a [real-world commit showing this in use on the `eleventy-base-blog` project].


```
const postcss = require("postcss");
const postcssNested = require("postcss-
        nested");

export default function(eleventyConfig)
        {
    eleventyConfig.addBundle("css", {
        transforms: [
            async function(content) {
                // this.type returns the
        bundle name.
                // Same as Eleventy
        transforms, this.page is
        available here.
                let result = await
        postcss([postcssNested]).process(conten
        { from: this.page.inputPath, to: null }
                return result.css;
            }
        ]
    });
};
```

Advanced

Limitations

Bundles do not support nesting or recursion (yet?). If this will be useful to you, please file an issue!

```
<style>/* This is bundled. */</style>
<style webc:keep>/* Do not bundle me—
        leave as is */</style>
```

To add JS to a page bundle in WebC, you would use a `<script>` element in a WebC page or component:

```
<script>/* This is bundled. */</script>
<script webc:keep>/* Do not bundle me—
        leave as is */</script>
```

- Existing calls via WebC helpers `getCss` or `getJs` (e.g. `<style @raw="getCss(page.url)">`) have been wired up to `getBundle` (for `"css"` and `"js"` respectively) automatically.
  - For consistency, you may prefer using the bundle plugin method names everywhere: `<style @raw="getBundle('css')">` and `<script @raw="getBundle('js')">` both work fine.
- Outside of WebC, the Universal Filters `webcGetCss` and `webcGetJs` were removed in Eleventy `v3.0.0-alpha.10` in favor of the `getBundle` Universal Shortcode (and respectively).

Modify the bundle output

You can wire up your own async-friendly callbacks to transform the bundle output too. Here's a quick example of `postcss` integration.

Example: Add bundle code in a Markdown file in Eleventy

```
# My Blog Post

This is some content, I am writing
        markup.


em { font-style: italic; }


## More Markdown


strong { font-weight: bold; }
```

Renders to:

```
<h1>My Blog Post</h1>

<p>This is some content, I am writing
        markup.</p>

<h2>More Markdown</h2>
```

Note that the bundled code is excluded!

There are a few more examples below!

Render bundle code

```
<!-- Use this *anywhere*: a layout file,
        content template, etc -->
<style></style>

<!--
You can add more code to the bundle
        after calling
getBundle and it will be included.
-->
* { color: orange; }
```

Write a bundle to a file

Writes the bundle content to a content-hashed file location in your output directory and returns the URL to the file for use like this:

```
<link rel="stylesheet" href="">
```

Note that writing bundles to files will likely be slower for empty-cache first time visitors but better cached in the browser for repeat-views (and across multiple pages, too).


```
<link href="https://
        v1.opengraph.11ty.dev"
        rel="preconnect" crossorigin>
```

Bundle Sass with the Render Plugin

You can render template syntax inside of the `` shortcode too, if you'd like to do more advanced things using Eleventy template types.

This example assumes you have added the Render plugin and the `scss` custom template type to your Eleventy configuration file.

```
h1 { .test { color: red; } }
```

Now the compiled Sass is available in your default bundle and will show up in `getBundle` and `getBundleFileUrl`.

Use with WebC

Starting with `@11ty/eleventy-plugin-webc@0.9.0` (track at issue #48) this plugin is used by default in the Eleventy WebC plugin. Specifically, WebC Bundler Mode now uses the bundle plugin under the hood.

To add CSS to a bundle in WebC, you would use a `<style>` element in a WebC page or component:

```
And now you can use `icon-close` in as
        many SVG instances as you'd like
        (without repeating the heftier
        SVG content).

<svg><use xlink:href="#icon-close"></
        use></svg>
<svg><use xlink:href="#icon-close"></
        use></svg>
<svg><use xlink:href="#icon-close"></
        use></svg>
<svg><use xlink:href="#icon-close"></
        use></svg>
```

React Helmet-style `<head>` additions

```
// .eleventy.js
export default function(eleventyConfig)
        {
    eleventyConfig.addBundle("html");
};
```

This might exist in an Eleventy layout file:

```
<head>

</head>
```

And then in your content you might want to page-specific `preconnect`:

Asset bucketing

```
<!-- This goes into a `defer` bucket
        (the bucket can be any string
        value) -->
em { font-style: italic; }

<!-- Pass the arbitrary `defer` bucket
        name as an additional argument
        -->
<style></style>
<link rel="stylesheet" href="">
```

A `default` bucket is implied:

```
<!-- These two statements are the same
        -->
em { font-style: italic; }
em { font-style: italic; }

<!-- These two are the same too -->
<style></style>
<style></style>
```

Examples

Critical CSS

```
// .eleventy.js
export default function(eleventyConfig)
        {
```

```
    eleventyConfig.addBundle("css");
};
```

Use asset bucketing to divide CSS between the `default` bucket and a `defer` bucket, loaded asynchronously.

(Note that some HTML boilerplate has been omitted from the sample below)

```
<!-- … -->
<head>
    <!-- Inlined critical styles -->
    <style></style>

    <!-- Deferred non-critical styles --
        >
    <link rel="stylesheet" href=""
        media="print"
        onload="this.media='all'">
    <noscript>
        <link rel="stylesheet" href="">
    </noscript>
</head>
<body>
    <!-- This goes into a `default`
        bucket -->
    /* Inline in the head, great with
        @font-face! */

        <!-- This goes into a `defer`
        bucket (the bucket can be any
        string value) -->
    /* Load me later */
```



```
</body>
<!-- … -->
```

Related:

- Check out the demo of Critical CSS using Eleventy Edge for a repeat view optimization without JavaScript.
- You may want to improve the above code with `fetchpriority` when browser support improves.

SVG Icon Library

Here an `svg` is bundle is created.

```
// .eleventy.js
export default function(eleventyConfig)
        {
    eleventyConfig.addBundle("svg");
};

<svg width="0" height="0" aria-
        hidden="true" style="position:
        absolute;">
    <defs></defs>
</svg>

<!-- And anywhere on your page you can
        add icons to the set -->

<g id="icon-close"><path d="…" /></g>
```