Gatsby Node APIs

Introduction

Gatsby gives plugins and site builders many APIs for building your site. Code in the file `gatsby-node.js`/`gatsby-node.ts` is run once in the process of building your site. You can use its APIs to create pages dynamically, add data into GraphQL, or respond to events during the build lifecycle. To use the Gatsby Node APIs, create a file named `gatsby-node.js`/`gatsby-node.ts` in the root of your site. Export any of the APIs you wish to use in this file.

You can author the file in JavaScript or TypeScript.

Every Gatsby Node API gets passed a set of helper functions. These let you access several methods like reporting, or perform actions like creating new pages.

An example `gatsby-node.js` file that implements two APIs, `onPostBuild`, and `createPages`.

```
const path = require(`path`)
// Log out information after a build is done
exports.onPostBuild = ({ reporter }) => {
  reporter.info(`Your Gatsby site has been built!`)
}
// Create blog pages dynamically
exports.createPages = async ({ graphql, actions }) => {
  const { createPage } = actions
  const blogPostTemplate = path.resolve(`src/templates/blog-post.js`)
  const result = await graphql(`
    query {
      allSamplePages {
        edges {
          node {
            slug
            title
          }
        }
      }
    }
  `)
```

```
  result.data.allSamplePages.edges.forEach(edge => {
    createPage({
      path: `${edge.node.slug}`,
      component: blogPostTemplate,
      context: {
        title: edge.node.title,
      },
    })
  })
}
```

The TypeScript and Gatsby documentation shows how to set up a `gatsby-node` file in TypeScript.

Async vs. sync work

If your plugin performs async operations (disk I/O, database access, calling remote APIs, etc.) you must either return a promise (explicitly using `Promise` API or implicitly using `async/await` syntax) or use the callback passed to the 3rd argument. Gatsby needs to know when plugins are finished as some APIs, to work correctly, require previous APIs to be complete first. See Debugging Async Lifecycles for more info.

```
// Async/await
exports.createPages = async () => {
  // do async work
  const result = await fetchExternalData()
}

// Promise API
exports.createPages = () => {
  return new Promise((resolve, reject) => {
    // do async work
  })
}

// Callback API
exports.createPages = (_, pluginOptions, cb) => {
  // do async work
  cb()
}
```

If your plugin does not do async work, you can return directly.