gatsby-source-filesystem

A Gatsby source plugin for sourcing data into your Gatsby application from your local filesystem.

The plugin creates `File` nodes from files. The various "transformer" plugins can transform `File` nodes into various other types of data e.g. `gatsby-transformer-json` transforms JSON files into JSON data nodes and `gatsby-transformer-remark` transforms markdown files into `MarkdownRemark` nodes from which you can query an HTML representation of the markdown.

Install

```
npm install gatsby-source-filesystem
```

How to use

```
// In your gatsby-config.js
module.exports = {
  plugins: [
    // You can have multiple instances of this plugin
    // to read source nodes from different locations on your
    // filesystem.
    //
    // The following sets up the Jekyll pattern of having a
    // "pages" directory for Markdown files and a "data" directory
    // for `.json`, `.yaml`, `.csv`.
    {
      resolve: `gatsby-source-filesystem`,
      options: {
        name: `pages`,
        path: `${__dirname}/src/pages/`,
      },
    },
    {
      resolve: `gatsby-source-filesystem`,
      options: {
        name: `data`,
        path: `${__dirname}/src/data/`,
        ignore: [`**/\.*`], // ignore files starting with a dot
      },
    },
  ],
}
```

Options

In addition to the name and path parameters you may pass an optional `ignore` array of file globs to ignore.

They will be added to the following default list:

```
**/*.un~
**/.DS_Store
**/.gitignore
**/.npmignore
**/.babelrc
**/yarn.lock
**/node_modules
../**/dist/**
```

To prevent concurrent requests overload of `processRemoteNode`, you can adjust the 200 default concurrent downloads, with `GATSBY_CONCURRENT_DOWNLOAD` environment variable.

How to query

You can query file nodes like the following:

```
{
  allFile {
    edges {
      node {
        extension
        dir
        modifiedTime
      }
    }
  }
}
```

To filter by the `name` you specified in the config, use `sourceInstanceName`:

```
{
  allFile(filter: { sourceInstanceName: { eq: "data" } }) {
    edges {
      node {
        extension
        dir
        modifiedTime
      }
    }
  }
}
```

Helper functions

`gatsby-source-filesystem` exports three helper functions:

- `createFilePath`
- `createRemoteFileNode`
- `createFileNodeFromBuffer`

createFilePath

When building pages from files, you often want to create a URL from a file's path on the file system. E.g. if you have a markdown file at `src/content/2018-01-23-an-exploration-of-the-nature-of-reality/index.md`, you might want to turn that into a page on your site at `example.com/2018-01-23-an-exploration-of-the-nat`
`createFilePath` is a helper function to make this task easier.

```
createFilePath({
  // The node you'd like to convert to a path
  // e.g. from a markdown, JSON, YAML file, etc
  node,
  // Method used to get a node
  // The parameter from `onCreateNode` should be passed in here
  getNode,
  // The base path for your files.
  // It is relative to the `options.path` setting in the `gatsby-source-filesystem` entries
  // Defaults to `src/pages`. For the example above, you'd use `src/content`.
  basePath,
  // Whether you want your file paths to contain a trailing `/` slash
  // Defaults to true
  trailingSlash,
})
```

Example usage

```
const { createFilePath } = require(`gatsby-source-filesystem`)

exports.onCreateNode = ({ node, getNode, actions }) => {
  const { createNodeField } = actions
  // Ensures we are processing only markdown files
  if (node.internal.type === "MarkdownRemark") {
    // Use `createFilePath` to turn markdown files in our `data/faqs` directory into `/faqs/
    const relativeFilePath = createFilePath({
      node,
      getNode,
      basePath: "data/faqs/",
    })

    // Creates new query'able field with name of 'slug'
    createNodeField({
      node,
      name: "slug",
      value: `/faqs${relativeFilePath}`,
    })
  }
}
```

createRemoteFileNode

When building source plugins for remote data sources such as headless CMSs, their data will often link to files stored remotely that are often convenient to download so you can work with them locally.

The `createRemoteFileNode` helper makes it easy to download remote files and add them to your site's GraphQL schema.

While downloading the assets, special characters (regex: `/:|\/|\*|\?|"|<|>|\||\\/g`) in filenames are replaced with a hyphen "-". When special characters are found a file hash is added to keep files unique e.g `a:file.jpg` becomes `a-file-73hd.jpg` (as otherwise `a:file.jpg` and `a*file.jpg` would overwrite themselves).

```
createRemoteFileNode({
  // The source url of the remote file
  url: `https://example.com/a-file.jpg`,

  // The id of the parent node (i.e. the node to which the new remote File node will be lin
  parentNodeId,

  // Gatsby's cache which the helper uses to check if the file has been downloaded already.
  getCache,

  // The action used to create nodes
  createNode,

  // A helper function for creating node Ids
  createNodeId,

  // OPTIONAL
  // Adds htaccess authentication to the download request if passed in.
  auth: { htaccess_user: `USER`, htaccess_pass: `PASSWORD` },

  // OPTIONAL
  // Adds extra http headers to download request if passed in.
  httpHeaders: { Authorization: `Bearer someAccessToken` },
  // OPTIONAL
  // Sets the file extension
  ext: ".jpg",
})
```

Example usage The following example is pulled from gatsby-source-wordpress. Downloaded files are created as `File` nodes and then linked to the WordPress Media node, so it can be queried both as a regular `File` node and from the `localFile` field in the Media node.

```
const { createRemoteFileNode } = require(`gatsby-source-filesystem`)

exports.downloadMediaFiles = ({
  nodes,
  getCache,
  createNode,
  createNodeId,
  _auth,
}) => {
  nodes.map(async node => {
    let fileNode
    // Ensures we are only processing Media Files
    // `wordpress__wp_media` is the media file type name for WordPress
    if (node.__type === `wordpress__wp_media`) {
      try {
        fileNode = await createRemoteFileNode({
          url: node.source_url,
          parentNodeId: node.id,
          getCache,
          createNode,
          createNodeId,
          auth: _auth,
        })
      } catch (e) {
        // Ignore
      }
    }

    // Adds a field `localFile` to the node
    // ___NODE appendix tells Gatsby that this field will link to another node
    if (fileNode) {
      node.localFile___NODE = fileNode.id
    }
  })
}
```

The file node can then be queried using GraphQL. See an example of this in the gatsby-source-wordpress README where downloaded images are queried using gatsby-transformer-sharp to use in the component gatsby-image.

Retrieving the remote file name and extension The helper tries first to retrieve the file name and extension by parsing the url and the path provided (e.g. if the url is `https://example.com/image.jpg`, the extension will be inferred as `.jpg` and the name as `image`). If the url does not contain an extension, we use the `file-type` package to infer the file type. Finally, the name and the extension *can* be explicitly passed, like so:

```
createRemoteFileNode({
  // The source url of the remote file
  url: `https://example.com/a-file-without-an-extension`,
  parentNodeId: node.id,
  getCache,
  createNode,
  createNodeId,
  // if necessary!
  ext: ".jpg",
  name: "image",
})
```

createFileNodeFromBuffer

When working with data that isn't already stored in a file, such as when querying binary/blob fields from a database, it's helpful to cache that data to the filesystem in order to use it with other transformers that accept files as input.

The `createFileNodeFromBuffer` helper accepts a `Buffer`, caches its contents to disk, and creates a file node that points to it.

The name of the file can be passed to the `createFileNodeFromBuffer` helper. If no name is given, the content hash will be used to determine the name.

Example usage

The following example is adapted from the source of `gatsby-source-mysql`:

```
// gatsby-node.js
const createMySqlNodes = require(`./create-nodes`)

exports.sourceNodes = async ({ actions, createNodeId, getCache }, config) => {
  const { createNode } = actions
  const { conn, queries } = config
  const { db, results } = await query(conn, queries)

  try {
    queries
      .map((query, i) => ({ ...query, ___sql: results[i] }))
      .forEach(result =>
        createMySqlNodes(result, results, createNode, {
          createNode,
          createNodeId,
          getCache,
        })
      )
    db.end()
  } catch (e) {
    console.error(e)
    db.end()
  }
}

// create-nodes.js
const { createFileNodeFromBuffer } = require(`gatsby-source-filesystem`)
const createNodeHelpers = require(`gatsby-node-helpers`).default

const { createNodeFactory } = createNodeHelpers({ typePrefix: `mysql` })

function attach(node, key, value, ctx) {
  if (Buffer.isBuffer(value)) {
    ctx.linkChildren.push(parentNodeId =>
      createFileNodeFromBuffer({
        buffer: value,
        getCache: ctx.getCache,
        createNode: ctx.createNode,
        createNodeId: ctx.createNodeId,
      })
    )
    value = `Buffer`
  }

  node[key] = value
}

function createMySqlNodes({ name, __sql, idField, keys }, results, ctx) {
  const MySqlNode = createNodeFactory(name)
  ctx.linkChildren = []

  return __sql.forEach(row => {
    if (!keys) keys = Object.keys(row)

    const node = { id: row[idField] }
    for (const key of keys) {
      attach(node, key, row[key], ctx)
    }

    node = ctx.createNode(node)

    for (const link of ctx.linkChildren) {
      link(node.id)
    }
  })
}

module.exports = createMySqlNodes
```

Troubleshooting

In case that due to spotty network, or slow connection, some remote files fail to download. Even after multiple retries and adjusting concurrent downloads, you can adjust timeout and retry settings with these environment variables:

- `GATSBY_STALL_RETRY_LIMIT`, default: `3`
- `GATSBY_STALL_TIMEOUT`, default: `30000`
- `GATSBY_CONNECTION_TIMEOUT`, default: `30000`