There are two types of plugins that work within Gatsby's data system, "source" and "transformer" plugins.

- **Source** plugins "source" data from remote or local locations into what Gatsby calls nodes.
- **Transformer** plugins "transform" data provided by source plugins into new nodes and/or node fields.

The purpose of this doc is to:

1. Define what a Gatsby transformer plugin is, and
2. Walk through a simplified reimplementation of an existing plugin, to demonstrate how to create a transformer plugin.

For a step-by-step process, check out the tutorial on Creating a Remark Transformer Plugin.

What do transformer plugins do?

Transformer plugins "transform" data of one type into another type. You'll often use both source plugins and transformer plugins in your Gatsby sites.

This loose coupling between the data source and transformer plugins allow Gatsby developers to quickly assemble complex data transformation pipelines with little work.

How do you create a transformer plugin?

Just like a source plugin, a transformer plugin is a normal npm package. It has a `package.json` file with optional dependencies as well as a `gatsby-node.js` file where you implement Gatsby's Node.js APIs.

`gatsby-transformer-yaml` is transformer plugin that looks for new nodes with a media type of text/YAML (e.g. a `.yaml` file) and creates new YAML child node(s) by parsing the YAML source into JavaScript objects.

Check out this example of rebuilding a simplified `gatsby-transformer-yaml` directly in a site. Say you have a default Gatsby starter site which includes a `src/data/example.yml` file:

```
- name: Jane Doe
  bio: Developer based in Somewhere, USA
- name: John Smith
  bio: Developer based in Maintown, USA
```

Make sure the data is sourced

First, in `gatsby-config.js`, use the `gatsby-source-filesystem` plugin to create File nodes.

```
module.exports = {
  plugins: [
    {
      resolve: `gatsby-source-filesystem`,
      options: {
        path: `./src/data/`,
      },
    },
  ],
}
```

These are exposed in your GraphQL schema which you can query:

```
query {
  allFile {
    edges {
      node {
        internal {
          type
          mediaType
          description
          owner
        }
      }
    }
  }
}
```

Now you have a `File` node to work with:


```
{
  "data": {
    "allFile": {
      "edges": [
        {
          "node": {
            "internal": {
              "contentDigest": "c1644b03f380bc5508456ce91faf0c08",
              "type": "File",
              "mediaType": "text/yaml",
              "description": "File \"src/data/example.yml\"",
              "owner": "gatsby-source-filesystem"
            }
          }
        }
      ]
    }
  }
}
```


Transform nodes of type `text/yaml`

Now, transform the newly created `File` nodes by hooking into the `onCreateNode` API in `gatsby-node.js`.

Convert YAML into JSON for storage in Gatsby nodes

If you're following along in an example project, install the following packages:

```
npm install js-yaml lodash
```

Now, in `gatsby-node.js`:

```
const jsYaml = require(`js-yaml`)

async function onCreateNode({ node, loadNodeContent }) {
  // only log for nodes of mediaType `text/yaml`
  if (node.internal.mediaType !== `text/yaml`) {
    return
  }

  const content = await loadNodeContent(node)
  const parsedContent = jsYaml.load(content)
}

exports.onCreateNode = onCreateNode
```

File content:

```
- id: Jane Doe
  bio: Developer based in Somewhere, USA
- id: John Smith
  bio: Developer based in Maintown, USA
```

Parsed YAML content:

```
[
  {
    "id": "Jane Doe",
    "bio": "Developer based in Somewhere, USA"
  },
  {
    "id": "John Smith",
    "bio": "Developer based in Maintown, USA"
  }
]
```

Now you'll write a helper function to transform the parsed YAML content into new Gatsby nodes:

```
function transformObject(obj, id, type) {
  const yamlNode = {
    ...obj,
    id,
    children: [],
    parent: null,
    internal: {
      contentDigest: createContentDigest(obj),
      type,
    },
  }
  createNode(yamlNode)
}
```

Above, you create a `yamlNode` object with the shape expected by the `createNode` action.

Creating the transformer relationship

You then need to create a link between the parent node (file) and the child node (YAML content) using the `createParentChildLink` function after adding the parent node's id to the `yamlNode`:

```
function transformObject(obj, id, type) {
  const yamlNode = {
    ...obj,
    id,
    children: [],
    parent: node.id, // highlight-line
    internal: {
      contentDigest: createContentDigest(obj),
      type,
    },
  }
  createNode(yamlNode)
  createParentChildLink({ parent: node, child: yamlNode }) // highlight-line
}
```

Another example of a transformation relationship is the `gatsby-source-filesystem` plugin used with the `gatsby-transformer-remark` plugin. This combination transforms a parent `File` node's markdown string into a `MarkdownRemark` node. The remark transformer plugin adds its newly created child node as a child of the parent node using the action `createParentChildLink`. Transformation relationships like this are used when a new node is *completely* derived from a single parent node. E.g. the markdown node is derived from the parent `File` node and would not exist if the parent `File` node hadn't been created.

Because all children nodes are derived from their parent, when a parent node is deleted or changed, Gatsby deletes all of the child nodes (and their child nodes, and so on). Gatsby does so with the expectation that they'll be recreated again by transformer plugins. This is done to ensure there are no nodes left over that were derived from older versions of data but should no longer exist.

For examples of other plugins creating transformation relationships, you can see the `gatsby-transformer-remark` plugin (from the above example) or the `gatsby-transformer-sharp` plugin.

Create new nodes from the derived data

In your updated `gatsby-node.js`, you'll then iterate through the parsed YAML content, using the helper function to transform each into a new node:

```
const jsYaml = require(`js-yaml`)
const _ = require(`lodash`)

async function onCreateNode({
  node,
  actions, // highlight-line
  loadNodeContent,
  createNodeId, // highlight-line
  createContentDigest, // highlight-line
}) {
```

```
  // highlight-start
  function transformObject(obj, id, type) {
    const yamlNode = {
      ...obj,
      id,
      children: [],
      parent: node.id,
      internal: {
        contentDigest: createContentDigest(obj),
        type,
      },
    }

    createNode(yamlNode)
    createParentChildLink({ parent: node, child: yamlNode })
  }
  // highlight-end

  const { createNode, createParentChildLink } = actions

  if (node.internal.mediaType !== `text/yaml`) {
    return
  }

  const content = await loadNodeContent(node)
  const parsedContent = jsYaml.load(content)

  // highlight-start
  parsedContent.forEach((obj, i) => {
    transformObject(
      obj,
      obj.id ? obj.id : createNodeId(`${node.id} [${i}] >>> YAML`),
      _.upperFirst(_.camelCase(`${node.name} Yaml`))
    )
  })
  // highlight-end
}

exports.onCreateNode = onCreateNode
```

Query for the transformed data

Now you can query for your new nodes containing our transformed YAML data:

```
query {
  allExampleYaml {
    edges {
      node {
        id
        name
        bio
      }
```

```
    }
  }
}
```


```
{
  "data": {
    "allExampleYaml": {
      "edges": [
        {
          "node": {
            "id": "3baa5e64-ac2a-5234-ba35-7af86746713f",
            "name": "Jane Doe",
            "bio": "Developer based in Somewhere, USA"
          }
        },
        {
          "node": {
            "id": "2c733815-c342-5d85-aa3f-6795d0f25909",
            "name": "John Smith",
            "bio": "Developer based in Maintown, USA"
          }
        }
      ]
    }
  }
}
```


Check out the full source code of `gatsby-transformer-yaml`.

Using the cache

Sometimes transforming properties costs time and resources. In order to avoid recreating these properties at each run, you can profit from the global cache mechanism Gatsby provides.

Cache keys should at least contain the contentDigest of the concerned node. For example, the `gatsby-transformer-remark` uses the following cache key for the HTML node:

```
const htmlCacheKey = node =>
  `transformer-remark-markdown-
html-${node.internal.contentDigest}-${pluginsCacheStr}-${pathPrefixCacheStr}`
```

Accessing and setting content in the cache is as simple as:

```
const cachedHTML = await cache.get(htmlCacheKey(markdownNode))

cache.set(htmlCacheKey(markdownNode), html)
```

Additional resources

- Tutorial: Creating a Remark Transformer Plugin