gatsby-transformer-excel

Parses Excel files into JSON arrays.

Install

`npm install gatsby-transformer-excel`

Note: You generally will use this plugin together with the `gatsby-source-filesystem` plugin. `gatsby-source-filesystem` reads in the files then this plugin *transforms* the files into data you can query.

How to use

If you put your Excel files in `./src/data`:

```
// In your gatsby-config.js
module.exports = {
  plugins: [
    {
      resolve: `gatsby-source-filesystem`,
      options: {
        name: `data`,
        path: `${__dirname}/src/data/`,
      },
    },
    `gatsby-transformer-excel`,
  ],
}
```

This plugin allows you to pass any available options used by the underlying library's function. Click here to view the full list of options and the default values.

You can see an example project at https://github.com/gatsbyjs/gatsby/tree/master/examples/using-excel.

Parsing algorithm

The parsing is powered by the SheetJS / js-xlsx library. Each row of each worksheet is converted into a node whose keys are determined by the first row and whose type is determined by the name of the worksheet.

So if your project has a `letters.xlsx` with two worksheets:

```
------ Sheet1 ------
/|    A    |   B   |
-+---------+-------+
1| letter  | value |
```

```
-+---------+-------+
2|    a    |   97  |
-+---------+-------+
3|    b    |   98  |

------ Sheet2 ------
/|    A    |   B   |
-+---------+-------+
1| letter  | value |
-+---------+-------+
2|    A    |   65  |
-+---------+-------+
3|    B    |   66  |
```

the following nodes would be created:

```
[
  { "letter": "a", "value": 97, "type": "LettersXlsxSheet1" },
  { "letter": "b", "value": 98, "type": "LettersXlsxSheet1" },
  { "letter": "A", "value": 65, "type": "LettersXlsxSheet2" },
  { "letter": "B", "value": 66, "type": "LettersXlsxSheet2" }
]
```

How to query

Using the same `letters.xlsx` example from above, you'd be able to query your letters like:

```
{
  allLettersXlsxSheet1 {
    edges {
      node {
        letter
        value
      }
    }
  }
}
```

Which would return:

```
{
  allLettersXlsxSheet1: {
    edges: [
      {
        node: {
          letter: 'a'
          value: 97
```

```
        }
      },
      {
        node: {
          letter: 'b'
          value: 98
        }
      },
    ]
  }
}
```

Troubleshooting

Default Values

If your spreadsheet contains column headers with only blank cells, the default behavior is to exclude this column in the GraphQL output as per the documentation

> If `defval` is not specified, `null` and `undefined` values are skipped normally. If specified, all `null` and `undefined` points will be filled with `defval`.

```
// In your gatsby-config.js
module.exports = {
  plugins: [
    {
      resolve: `gatsby-transformer-excel`,
      options: {
        defval: "",
      },
    },
  ],
}
```

This will make sure that any blank cells are assigned the `defval` value.

Field Type Conflicts

If your columns have different data types, e.g. numbers and strings, GraphQL will omit these values and provide you with a field type conflicts warning during build. To solve this, you can set the rawOutput option of the plugin to false. This will convert all values to strings.

```
// In your gatsby-config.js
module.exports = {
  plugins: [
```

```
    {
      resolve: `gatsby-transformer-excel`,
      options: {
        raw: false,
      },
    },
  ],
}
```

This will make sure that all field types are converted to strings.

NOTE 1: A previous version of this library used the attribute name `rawOutput`. This name still works, but is an alias for the correct attribute `raw`. If both attributes are specified, the value for `raw` takes precedence.