

## Introduction

[MDX](#) is Markdown for the component era. It lets you write JSX embedded inside Markdown. This combination allows you to use Markdown's terse syntax (such as `# Heading`) for your content and JSX for more advanced or reusable components.

This is useful in content-driven sites where you want the ability to introduce components like charts or alerts without having to configure a plugin. MDX also shines in interactive blog posts, documenting design systems, or long form articles with immersive or dynamic interactions.

## Part 1: Getting Started with MDX

If you already have a Gatsby site that you'd like to add MDX to, you can follow these steps for configuring the [gatsby-plugin-mdx](#) plugin.

**Starting a new project?** Skip the setup and create a new project using `npm init gatsby`

Choose the option "Add Markdown and MDX support" to add the necessary MDX dependencies.

**Already using Remark?** Check out the How-To Guide on [Migrating from Remark to MDX](#).

1. Add `gatsby-plugin-mdx` and MDX as dependencies

```
npm install gatsby-plugin-mdx @mdx-js/mdx@v1 @mdx-js/react@v1
```

2. Update your `gatsby-config.js` to use `gatsby-plugin-mdx`

```
module.exports = {
  plugins: [
    // ...
    `gatsby-plugin-mdx`,
  ],
}
```

3. Restart your local development server by running `gatsby develop`.

## Part 2: Writing Pages in MDX

After installing `gatsby-plugin-mdx`, MDX files located in the `src/pages` directory will automatically be turned into pages.

Pages are rendered at a URL that is constructed from the filesystem path inside `src/pages`. For example, an MDX file at `src/pages/awesome.mdx` will result in a page being rendered at `yoursite.com/awesome`.

Create a new `.mdx` file in the `src/pages` directory. You can use [Markdown syntax](#) to add different HTML elements.

### Using frontmatter in MDX

By default, `gatsby-plugin-mdx` supports [frontmatter](#) so you can define things like titles and paths to use in your GraphQL queries. You can declare frontmatter at the beginning of your MDX document:

```
---
title: Hello, world!
path: /hello-world
date: 2019-01-29
---

# Hello, world!
```

You can then query for this frontmatter data with [GraphQL](#):

```
query {
  allMdx {
    edges {
      node {
        frontmatter {
          title
          path
          date(formatString: "MMMM DD, YYYY")
        }
      }
    }
  }
}
```

**Note:** To query `MDX` content, it must be included in the node system using a source like the `gatsby-source-filesystem` plugin first.

Check out the How-To Guide: [How to Source Data from the Filesystem](#).

Frontmatter is also available in `props.pageContext.frontmatter` and can be accessed in blocks of JSX in your MDX document:

```
---
title: Building with Gatsby
author: Jay Gatsby
---

<h1>{props.pageContext.frontmatter.title}</h1>
<span>{props.pageContext.frontmatter.author}</span>

(Blog post content, components, etc.)
```

## Part 3: Importing JSX components and MDX documents

MDX allows you to use React components alongside Markdown. You can import components from third-party libraries (like [theme-ui](#)) to take advantage of pre-built functionality like data visualizations, email signup forms, or call-to-action buttons. You can also import and reuse *your own* React components and even other MDX documents.

To import a component, add a JavaScript `import` statement to your MDX file. Once you've imported a component, you can use it in the body of your MDX file the same way you'd normally use a React component:

```
---
title: Importing Components Example
---

import { Message } from "theme-ui" // highlight-line

You can import your own components.

<Message>MDX gives you JSX in Markdown!</Message> // highlight-line
```

**Note:** If you would like to include frontmatter metadata and import components, the frontmatter needs to appear at the top of the file and then imports can follow.

## Make components available globally as shortcodes

To avoid having to import the same component inside of every MDX document you author, you can add components to an `MDXProvider` to make them globally available in MDX pages. This pattern is sometimes referred to as shortcodes.

```
import React from "react"
// highlight-start
import { MDXProvider } from "@mdx-js/react"
import { Chart, Pullquote } from "./ui"
import { Message } from "theme-ui"
// highlight-end

const shortcodes = { Chart, Pullquote, Message } // highlight-line

export default function Layout({ children }) {
  return (
    <MDXProvider components={shortcodes}>{children}</MDXProvider> // highlight-line
  )
}
```

All MDX components passed into the `components` prop of the `MDXProvider` will be made available to MDX documents that are nested under the provider. The `MDXProvider` in this example is in a layout component that wraps all MDX pages, you can read about this pattern in [the layout section of the gatsby-plugin-mdx README](#).

Now, you can include components in your MDX without importing them:

```
---
title: Shortcode Components Example
---

Now, if you want to include the Message component, it's available in all MDX
documents!

<Message>MDX gives you JSX in Markdown!</Message> // highlight-line

The Chart is also available since it was passed into the MDXProvider:
```

```
<Chart /> // highlight-line
```

Because the `<Message />` and `<Chart />` components were passed into the provider, they are available for use in all MDX documents.

## Part 4: Making GraphQL queries in an MDX File

You can fetch data to use in your MDX file by exporting a `pageQuery` in the same way you would for a `.js` page. The queried data is passed as a prop, and can be accessed inside any JSX block when writing in MDX:

```
import { graphql } from "gatsby"

# My Awesome Page

Here's a paragraph, followed by a paragraph with data!

<p>{props.data.site.siteMetadata.description}</p>

export const pageQuery = graphql`
  query {
    site {
      siteMetadata {
        description
        title
      }
    }
  }
`
```

*Note: For now, this only works if the `.mdx` file exporting the query is placed in `src/pages`. Exporting GraphQL queries from `.mdx` files that are used for programmatic page creation in `gatsby-node.js` via `actions.createPage` is not currently supported.*