Sometimes you want to be able to programmatically access data from files in `src/pages/` or create pages using MDX content that lives at arbitrary locations outside of `src/pages/` or in remote CMSes.

For instance, let's say you have a Gatsby website, and you want to add support for MDX so you can start your blog. The posts will live in `content/posts/`. You can do this with the help of `gatsby-source-filesystem` and `createPages` in `gatsby-node.js`.

Source MDX pages from the filesystem

To let Gatsby know that you'll be working with MDX content you need to add `gatsby-plugin-mdx` to the plugins array in your `gatsby-config.js` file.

You'll need to use `gatsby-source-filesystem` and tell it to source "posts" from a folder called `content/posts` located in the project's root.

> ***Note**: `gatsby-plugin-mdx` uses `.mdx` by default as a file extension to recognize which files to use. You can also use `.md` as a file extension if you want.*

```
module.exports = {
  plugins: [
    // Add support for *.mdx files in gatsby
    "gatsby-plugin-mdx",

    // Add a collection called "posts" that looks
    // for files in content/posts/
    {
      resolve: "gatsby-source-filesystem",
      options: {
        name: "posts",
        path: `${__dirname}/content/posts/`,
      },
    },
  ],
}
```

You can read about `gatsby-source-filesystem` if you'd like to learn more.

Add MDX files

Before you can write any GraphQL queries and programmatically create pages, you need to add some content.

Make a folder called `content/posts` and create two files in it called `blog-1.mdx` and `blog-2.mdx`. You can do this on the command line in a terminal by using the following commands from the root of your project.

```
mkdir -p content/posts
touch content/posts/blog-{1,2}.mdx
```

> ***Note**: `mkdir -p path/to/a/directory` will create every folder in the path if it does not exist.*
>
> *`touch <filename>` will create an empty file named `<filename>`. The brackets (`{}`) are an expansion which means you can create multiple files in one command.*

Open up each of the files you just created and add some content.

```
---
title: Blog Post 1
---

Trying out MDX
```

```
---
title: Blog Post 2
---

Gatsby is the best
```

Create pages from sourced MDX files

`gatsby-plugin-mdx` automatically adds a `slug` field to each MDX node derived from the corresponding filename. You can use special characters, whitespaces, or uppercase in the filenames, and `gatsby-plugin-mdx` will "slugify" it, making the slug safe for serving over the internet and human readable.

In order to create pages from the sourced MDX files, you need to construct a query that finds all MDX nodes and pulls out the `slug` field.

> ***Note**: You can open up a GraphiQL console for query testing in your browser at* `http://localhost:8000/___graphql`

```
query {
  allMdx {
    edges {
      node {
        id
        # Slug field added by gatsby-plugin-mdx
        slug
      }
    }
  }
}
```

```
const path = require("path")

exports.createPages = async ({ graphql, actions, reporter }) => {
  // Destructure the createPage function from the actions object
  const { createPage } = actions

  const result = await graphql(`
    query {
      allMdx {
        edges {
          node {
            id
            slug
          }
        }
      }
    }
  `)

  if (result.errors) {
    reporter.panicOnBuild('🚨  ERROR: Loading "createPages" query')
  }

  // Create blog post pages.
  const posts = result.data.allMdx.edges

  // you'll call `createPage` for each result
  posts.forEach(({ node }, index) => {
    createPage({
      // The slug generated by gatsby-plugin-mdx doesn't contain a slash at the
beginning
      // You can prepend it with any prefix you want
      path: `/blog/${node.slug}`,
      // This component will wrap our MDX content
      component: path.resolve(`./src/components/posts-page-layout.js`),
      // You can use the values in this context in
      // our page layout component
      context: { id: node.id },
    })
  })
}
```

For further reading, check out the createPages API.

Make a template for your posts

You can create a file called `posts-page-layout.js` in `src/components` - this component will be rendered as the template for all posts. There's a component, `MDXRenderer` which is used by `gatsby-plugin-mdx` that will be used to render any programmatically accessed MDX content.

For now, to update imports within `.mdx` files, you should rerun your Gatsby development environment. Otherwise, it will raise a `ReferenceError`. To import things dynamically, you can use the `MDXProvider` component and provide it all the common components you'll be using, such as `Link`.

First, create a component that accepts the queried MDX data (which will be added in the next step).

```
import React from "react"
import { graphql } from "gatsby"
import { MDXProvider } from "@mdx-js/react"
import { MDXRenderer } from "gatsby-plugin-mdx"
import { Link } from "gatsby"
```

```
const shortcodes = { Link } // Provide common components here

export default function PageTemplate({ data: { mdx } }) {
  return (
    <div>
      <h1>{mdx.frontmatter.title}</h1>
      <MDXProvider components={shortcodes}>
        <MDXRenderer frontmatter={mdx.frontmatter}>{mdx.body}</MDXRenderer>
      </MDXProvider>
    </div>
  )
}
```

Then, write a query that uses `id` which is passed through the `context` object in `createPage`. GraphQL requires you to declare the type of arguments at the top of the query before they're used.

```
export const pageQuery = graphql`
  query BlogPostQuery($id: String) {
    mdx(id: { eq: $id }) {
      id
      body
      frontmatter {
        title
      }
    }
  }
`
```

When you put the component and page query all together, the component should look like:

```
import React from "react"
import { graphql } from "gatsby"
import { MDXProvider } from "@mdx-js/react"
import { MDXRenderer } from "gatsby-plugin-mdx"
import { Link } from "gatsby"

const shortcodes = { Link } // Provide common components here

export default function PageTemplate({ data: { mdx } }) {
  return (
    <div>
      <h1>{mdx.frontmatter.title}</h1>
      <MDXProvider components={shortcodes}>
        <MDXRenderer frontmatter={mdx.frontmatter}>{mdx.body}</MDXRenderer>
      </MDXProvider>
    </div>
  )
}

export const pageQuery = graphql`
  query BlogPostQuery($id: String) {
    mdx(id: { eq: $id }) {
      id
      body
      frontmatter {
        title
      }
    }
  }
`
```

That's it, you're done. Run `gatsby develop` and enjoy your new MDX powers.

Now you have all the pieces you need to programmatically create pages with Gatsby and `gatsby-plugin-mdx`. You can check out our other guides to find out more about all of the cool stuff you can do with `gatsby-plugin-mdx`.

Bonus: Make a blog index

```
import React from "react"
import { Link, graphql } from "gatsby"

const BlogIndex = ({ data }) => {
  const { edges: posts } = data.allMdx

  return (
    <div>
      <h1>Awesome MDX Blog</h1>

      <ul>
        {posts.map(({ node: post }) => (
          <li key={post.id}>
            <Link to={post.slug}>
              <h2>{post.frontmatter.title}</h2>
            </Link>
            <p>{post.excerpt}</p>
          </li>
        ))}
      </ul>
    </div>
  )
}

export const pageQuery = graphql`
  query blogIndex {
    allMdx {
      edges {
        node {
          id
          excerpt
          frontmatter {
            title
          }
          slug
        }
      }
    }
  }
`

export default BlogIndex
```