Custom `Document`

A custom `Document` is commonly used to augment your application's `<html>` and `<body>` tags. This is necessary because Next.js pages skip the definition of the surrounding document's markup.

To override the default `Document`, create the file `./pages/_document.js` and extend the `Document` class as shown below:

```
import Document, { Html, Head, Main, NextScript } from 'next/document'

class MyDocument extends Document {
  static async getInitialProps(ctx) {
    const initialProps = await Document.getInitialProps(ctx)
    return { ...initialProps }
  }

  render() {
    return (
      <Html>
        <Head />
        <body>
          <Main />
          <NextScript />
        </body>
      </Html>
    )
  }
}
```

```
export default MyDocument
```

The code above is the default `Document` added by Next.js. Feel free to remove the `getInitialProps` or `render` function from `MyDocument` if you don't need to change them.

`<Html>`, `<Head />`, `<Main />` and `<NextScript />` are required for the page to be properly rendered.

Custom attributes are allowed as props, like `lang`:

```
<Html lang="en">
```

The `<Head />` component used here is not the same one from `next/head`. The `<Head />` component used here should only be used for any `<head>` code that is common for all pages. For all other cases, such as `<title>` tags, we recommend using `next/head` in your pages or components.

The `ctx` object is equivalent to the one received in `getInitialProps`, with one addition:

- `renderPage`: `Function` - a callback that runs the actual React rendering logic (synchronously). It's useful to decorate this function in order to support server-rendering wrappers like Aphrodite's `renderStatic`

Caveats

- `Document` is only rendered in the server, event handlers like `onClick` won't work.
- React components outside of `<Main />` will not be initialized by the browser. Do not add application logic here or custom CSS (like `styled-jsx`). If you need shared

/

components in all your pages (like a menu or a toolbar), take a look at the `App` component instead.

- `Document` currently does not support Next.js Data Fetching methods like `getStaticProps` or `getServerSideProps`.

Customizing `renderPage`

It should be noted that the only reason you should be customizing `renderPage` is for usage with **css-in-js** libraries that need to wrap the application to properly work with server-side rendering.

It takes as argument an options object for further customization:

```
import Document from 'next/document'

class MyDocument extends Document {
  static async getInitialProps(ctx) {
    const originalRenderPage = ctx.renderPage

    ctx.renderPage = () =>
      originalRenderPage({
        // useful for wrapping the whole react tree
        enhanceApp: (App) => App,
        // useful for wrapping in a per-page basis
        enhanceComponent: (Component) => Component,
      })

    // Run the parent `getInitialProps`, it now includes the custom `renderPag
    const initialProps = await Document.getInitialProps(ctx)

    return initialProps
  }
}

export default MyDocument
```

TypeScript

You can use the built-in `DocumentContext` type and change the file name to `./pages/_document.tsx` like so:

```
import Document, { DocumentContext } from 'next/document'

class MyDocument extends Document {
  static async getInitialProps(ctx: DocumentContext) {
    const initialProps = await Document.getInitialProps(ctx)

    return initialProps
  }
}

export default MyDocument
```

‹ Custom `App`   Custom Error Page ›

Was this helpful?

😭 😕 😀 🤩