GraphQL API

A great advantage of Gatsby is a built-in data layer that combines all data sources you configure. Data is collected at build time and automatically assembled into a schema that defines how data can be queried throughout your site.

This doc serves as a reference for GraphQL features built into Gatsby, including methods for querying and sourcing data, and customizing GraphQL for your site's needs.

Getting started with GraphQL

GraphQL is available in Gatsby without a special install: a schema is automatically inferred and created when you run `gatsby develop` or `gatsby build`. When the site compiles, the data layer can be explored at: `http://localhost:8000/___graphql`

Sourcing data

Data needs to be sourced — or added to the GraphQL schema — to be queried and pulled into pages using GraphQL. Gatsby uses source plugins to pull in data.

Note: GraphQL isn't required: you can still use Gatsby without GraphQL.

To source data with an existing plugin you have to install all needed packages. Furthermore you have to add the plugin to the plugins array in the `gatsby-config` with any optional configurations. If you want to source data from the filesystem for use with GraphQL, such as Markdown files, images, and more, refer to the filesystem data sourcing docs and recipes.

For instructions on installing plugins from npm, take a look at the instructions in the docs on using a plugin.

You can also create custom plugins to fit your own use cases and pull in data however you want.

Query components and hooks

Data can be queried inside pages, components, or the `gatsby-node.js` file, using one of these options:

- The `pageQuery` component
- The `StaticQuery` component
- The `useStaticQuery` hook

Note: Because of how Gatsby processes GraphQL queries, you can't mix page queries and static queries in the same file. You also can't have multiple page queries or static queries in one file.

For information on page and non-page components as they relate to queries, check out the docs guide on building with components

`pageQuery`

`pageQuery` is a built-in component that retrieves information from the data layer in Gatsby pages. You can have one page query per page. It can take GraphQL arguments for variables in your queries.

A page is made in Gatsby by any React component in the `src/pages` folder, or by calling the `createPage` action and using a component in the `createPage` options – meaning a `pageQuery` won't work in any component, only in components which meet this criteria.

Also, refer to the guide on querying data in pages with page query.

Params A page query isn't a method, but rather an exported variable that's assigned a `graphql` string and a valid query block as its value:

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

Note: the query exported in a `const` doesn't need to be named `pageQuery`. More importantly, Gatsby looks for an exported `graphql` string from the file.

Returns When included in a page component file, a page query returns a data object that is passed automatically to the component as a prop.

```
// highlight-start
const HomePage = ({ data }) => {
  // highlight-end
  return (
    <div>
      Hello!
```

```
      {data.site.siteMetadata.description} // highlight-line
    </div>
  )
}
```

`StaticQuery`

StaticQuery is a built-in component for retrieving data from Gatsby's data layer in non-page components, such as a header, navigation, or any other child component.

You can only have one `StaticQuery` per page: in order to include the data you need from multiple sources, you can use one query with multiple root fields. It cannot take variables as arguments.

Also, refer to the guide on querying data in components with static query.

Params The `StaticQuery` component takes two values as props in JSX:

- `query`: a `graphql` query string
- `render`: a component with access to the data returned

```
<StaticQuery
  query={graphql` //highlight-line
    query HeadingQuery {
      site {
        siteMetadata {
          title
        }
      }
    }
  `}
  render={(
    data //highlight-line
  ) => (
    <header>
      <h1>{data.site.siteMetadata.title}</h1>
    </header>
  )}
/>
```

Returns The StaticQuery component returns `data` in a `render` prop:

```
<StaticQuery
  // ...
  // highlight-start
  render={data => (
    <header>
```

```
      <h1>{data.site.siteMetadata.title}</h1>
    </header>
  )}
  // highlight-end
/>
```

`useStaticQuery`

The `useStaticQuery` hook can be used similar to `StaticQuery` in any component or page, but doesn't require the use of a component and render prop.

Because it is a React hook, the rules of hooks apply and you'll need to use it with React and ReactDOM version 16.8.0 or later. Because of how queries currently work in Gatsby, only one instance of `useStaticQuery` is supported in each file.

Also, refer to the guide on querying data in components with useStaticQuery.

Params The `useStaticQuery` hook takes one argument:

- `query`: a `graphql` query string

```
const data = useStaticQuery(graphql`
  query HeaderQuery {
    site {
      siteMetadata {
        title
      }
    }
  }
`)
```

Returns The `useStaticQuery` hook returns data in an object:

```
const data = useStaticQuery(graphql`
  query HeaderQuery {
    site {
      siteMetadata {
        title
      }
    }
  }
`)
return (
  // highlight-start
  <header>
    <h1>{data.site.siteMetadata.title}</h1>
  </header>
```

```
  // highlight-end
)
```

Query structure

Queries are written in the same shape you want data returned in. How you source data will determine the names of fields that you can query on, based on the nodes they add to the GraphQL schema.

For understanding the parts of a query refer to the conceptual guide.

GraphQL query arguments

GraphQL queries can take arguments to alter how the data is returned. The logic for these arguments is handled internally by Gatsby. Arguments can be passed into fields at any level of the query.

Different nodes can take different arguments based off of the nature of the node.

The arguments you can pass to collections (like arrays or long lists of data - ex. `allFile`, or `allMdx`) are:

- `filter`
- `limit`
- `sort`
- `skip`

The arguments you can pass to a `date` field are:

- `formatString`
- `locale`

The arguments you can pass to an `excerpt` field are:

- `pruneLength`
- `truncate`

GraphQL query operations

Other built-in configurations can be used in queries

- `Alias`
- `Group`

For examples, refer to the query recipes and GraphQL query options reference guide.

Query fragments

Fragments allow you to reuse parts of GraphQL queries. They also allow you to split up complex queries into smaller, easier to understand components.

For more information, check out the docs guide on using fragments in Gatsby.

Gatsby fragments

Some fragments come included in Gatsby plugins, such as fragments for returning optimized image data in various formats with `gatsby-image` and `gatsby-transformer-sharp`, or data fragments with `gatsby-source-contentful`. For more information on what plugins include fragments, see the `gatsby-image` README.

Advanced customizations

You can customize sourced data in the GraphQL layer and create relationships between nodes with the Gatsby Node APIs.

The GraphQL schema can be customized for more advanced use cases: read more about it in the schema customization API docs.