Location Data from Props

What is location data

Sometimes it can be helpful to know exactly what your app's browser URL is at any given stage. Because Gatsby uses @reach/router for client-side routing, the `location` prop is passed to any page component and represents where the app is currently, where you'd like it to go, and other helpful information. The `location` object is never mutated but `reach@router` makes it helpful to determine when navigation happens. Here is a sample `props.location`:

```
{
  key: 'ac3df4', // does not populate with a HashHistory!
  pathname: '/somepage',
  search: '?someurlparam=valuestring1&anotherurlparam=valuestring2',
  hash: '#about',
  state: {
    [userDefined]: true
  }
}
```

Note that you have to parse the `search` field (the query string) into individual keys and values yourself.

HashHistory

Using `hash` in JavaScript is one way to update the browser URL and the DOM without having the browser do a full HTML page reload. HashHistory in `@reach/router` is used to track browser history with JavaScript when using hashrouter instead of browserrouter which uses the newer HTML5 `history` API.

Getting the absolute URL of a page

The `location` object's properties generally do not include the domain of your site, since Gatsby doesn't know where you will deploy it.

Running client side is the exception to this rule. In this case, all the information your browser exposes as `window.location` is available. This includes `href` for the absolute URL of the page, including the domain.

Sometimes you need the absolute URL of the current page (including the host name) while using server-side rendering. For example, you may want to add a canonical URL to the page header.

In this case, you would first need to add configuration that describes where your site is deployed. You can add this as a `siteURL` property on `siteMetadata` in `gatsby-config.js`.

Once you have added `siteURL`, you can form the absolute URL of the current page by retrieving `siteURL` and concatenating it with the current path from `location`. Note that the path starts with a slash; `siteURL` must therefore not end in one.

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

const Page = ({ location, data }) => {
  const canonicalUrl = data.site.siteMetadata.siteURL + location.pathname

  return <div>The URL of this page is {canonicalUrl}</div>
}

export default Page

export const query = graphql`
  query PageQuery {
    site {
      siteMetadata {
        siteURL
      }
    }
  }
`
```

Use cases

Through client-side routing in Gatsby you can provide a location object instead of strings, which are helpful in a number of situations:

- Providing state to linked components
- Client-only routes
- Fetching data
- Animation transitions

Example of providing state to a link component

```
// usually you'd do this
<Link to="/somepagecomponent"/>
```

```
// but if you want to add some additional state
<Link
  to={'/somepagecomponent'}
  state={{modal: true}}
/>
```

Then from the receiving component you can conditionally render markup based on the `location` state.

```
const SomePageComponent = ({ location }) => {
  const { state = {} } = location
  const { modal } = state
  return modal ? (
    <dialog className="modal">I'm a modal of Some Page Component!</dialog>
  ) : (
    <div>Welcome to the Some Page Component!</div>
  )
}
```

Other resources

- Gatsby Link API
- @reach/router docs
- react-router location docs
- Hash Router
- Gatsby Breadcrumb Plugin
- Create Modal w/ Navigation State using React Router