Gatsby Link API

For internal navigation, Gatsby includes a built-in `<Link>` component for creating links between internal pages as well as a `navigate` function for programmatic navigation.

The component is a wrapper around @reach/router's Link component that adds useful enhancements specific to Gatsby. All props are passed through to @reach/router's `Link` component.

`<Link>` drives Gatsby's fast page navigation

The `<Link>` component drives a powerful performance feature called preloading. Preloading is used to prefetch page resources so that the resources are available by the time the user navigates to the page. We use the browser's `Intersection Observer API` to observe when a `<Link>` component enters the user viewport and then start a low-priority request for the linked page's resources. Then when a user moves their mouse over a link and the `onMouseOver` event is triggered, we upgrade the fetches to high-priority.

This two stage preloading helps ensure the page is ready to be rendered as soon as the user clicks to navigate.

Intelligent preloading like this eliminates the latency users experience when clicking on links in sites built in most other frameworks.

How to use Gatsby Link

Replace `a` tags with the `Link` tag for local links

In any situation where you want to link between pages on the same site, use the `Link` component instead of an `a` tag. The two elements work much the same except `href` is now `to`.

```
-<a href="/blog">Blog</a>
+<Link to="/blog">Blog</Link>
```

A full example:

```
import React from "react"
// highlight-next-line
```

```
import { Link } from "gatsby"

const Page = () => (
  <div>
    <p>
      {/* highlight-next-line */}
      Check out my <Link to="/blog">blog</Link>!
    </p>
    <p>
      {/* Note that external links still use `a` tags. */}
      Follow me on <a href="https://twitter.com/gatsbyjs">Twitter</a>!
    </p>
  </div>
)
```

Add custom styles for the currently active link

It's often a good idea to show which page is currently being viewed by visually changing the link matching the current page.

`Link` provides two options for adding styles to the active link:

- `activeStyle` — a style object that will only be applied when the current item is active
- `activeClassName` — a class name that will only be added to the `Link` when the current item is active

For example, to turn the active link red, either of the following approaches is valid:

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

const SiteNavigation = () => (
  <nav>
    <Link
      to="/"
      {/* highlight-start */}
      {/* This assumes the `active` class is defined in your CSS */}
      activeClassName="active"
      {/* highlight-end */}
    >
      Home
    </Link>
    <Link
      to="/about/"
      {/* highlight-next-line */}
      activeStyle={{ color: "red" }}
```

```
    >
      About
    </Link>
  </nav>
)
```

Use `getProps` for advanced link styling

Gatsby's `<Link>` component comes with a `getProps` prop, which can be useful for advanced styling. It passes you an object with the following properties:

- `isCurrent` — true if the `location.pathname` is exactly the same as the `<Link>` component's `to` prop
- `isPartiallyCurrent` — true if the `location.pathname` starts with the `<Link>` component's `to` prop
- `href` — the value of the `to` prop
- `location` — the page's `location` object

You can read more about it on `@reach/router`'s documentation.

Show active styles for partially matched and parent links

By default the `activeStyle` and `activeClassName` props will only be set on a `<Link>` component if the current URL matches its `to` prop *exactly*. Sometimes, you may want to style a `<Link>` as active even if it partially matches the current URL. For example:

- You may want `/blog/hello-world` to match `<Link to="/blog">`
- Or `/gatsby-link/#passing-state-through-link-and-navigate` to match `<Link to="/gatsby-link">`

In instances like these, just add the `partiallyActive` prop to your `<Link>` component and the style will also be applied even if the `to` prop only is a partial match:

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

const Header = <>
  <Link
    to="/articles/"
    activeStyle={{ color: "red" }}
    {/* highlight-next-line */}
    partiallyActive={true}
  >
    Articles
  </Link>
</>;
```

Note: *Available from Gatsby V2.1.31, if you are experiencing issues please check your version and/or update.*

Pass state as props to the linked page

Sometimes you'll want to pass data from the source page to the linked page. You can do this by passing a `state` prop to the `Link` component or on a call to the `navigate` function. The linked page will have a `location` prop containing a nested `state` object structure containing the passed data.

```
const PhotoFeedItem = ({ id }) => (
  <div>
    {/* (skip the feed item markup for brevity) */}
    <Link
      to={`/photos/${id}`}
      {/* highlight-next-line */}
      state={{ fromFeed: true }}
    >
      View Photo
    </Link>
  </div>
)

// highlight-start
const Photo = ({ location, photoId }) => {
  if (location.state.fromFeed) {
    // highlight-end
    return <FromFeedPhoto id={photoId} />
  } else {
    return <Photo id={photoId} />
  }
}
```

Replace history to change "back" button behavior

There are a few cases where it might make sense to modify the "back" button's behavior. For example, if you build a page where you choose something, then see an "are you sure?" page to make sure it's what you really wanted, and finally see a confirmation page, it may be desirable to skip the "are you sure?" page if the "back" button is clicked.

In those cases, use the `replace` prop to replace the current URL in history with the target of the `Link`.

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

const AreYouSureLink = () => (
```

```
  <Link
    to="/confirmation/"
    {/* highlight-next-line */}
    replace
  >
    Yes, I'm sure
  </Link>
)
```

How to use the `navigate` helper function

Sometimes you need to navigate to pages programmatically, such as during form submissions. In these cases, `Link` won't work.

Note: *`navigate` was previously named `navigateTo`. `navigateTo` is deprecated in Gatsby v2 and will be removed in the next major release.*

Instead, Gatsby exports a `navigate` helper function that accepts `to` and `options` arguments.

Argument	Required	Description
`to`	yes	The page to navigate to (e.g. `/blog/`). Note: it needs to be a pathname, not a full URL.
`options.state`	no	An object. Values passed here will be available in `location.state` in the target page's props.
`options.replace`	no	A boolean value. If true, replaces the current URL in history.

By default, `navigate` operates the same way as a clicked `Link` component.

```
import React from "react"
import { navigate } from "gatsby" // highlight-line

const Form = () => (
  <form
    onSubmit={event => {
      event.preventDefault()

      // TODO: do something with form values
      // highlight-next-line
      navigate("/form-submitted/")
    }}
  >
    {/* (skip form inputs for brevity) */}
  </form>
)
```

Add state to programmatic navigation

To include state information, add an `options` object and include a `state` prop with the desired state.

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

const Form = () => (
  <form
    onSubmit={event => {
      event.preventDefault()

      // Implementation of this function is an exercise for the reader.
      const formValues = getFormValues()

      navigate(
        "/form-submitted/",
        // highlight-start
        {
          state: { formValues },
        }
        // highlight-end
      )
    }}
  >
    {/* (skip form inputs for brevity) */}
  </form>
)
```

Then from the receiving page you can access the `location` state as demonstrated in Pass state as props to the linked page.

Replace history during programmatic navigation

If the navigation should replace history instead of pushing a new entry into the navigation history, add the `replace` prop with a value of `true` to the `options` argument of `navigate`.

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

const Form = () => (
  <form
    onSubmit={event => {
      event.preventDefault()

      // TODO: do something with form values
```

```
      navigate(
        "/form-submitted/",
        // highlight-next-line
        { replace: true }
      )
    }}
  >
    {/* (skip form inputs for brevity) */}
  </form>
)
```

Add the path prefix to paths using `withPrefix`

It is common to host sites in a sub-directory of a site. Gatsby lets you set the path prefix for your site. After doing so, Gatsby's `<Link>` component will automatically handle constructing the correct URL in development and production.

For pathnames you construct manually, there's a helper function, `withPrefix` that prepends your path prefix in production (but doesn't during development where paths don't need to be prefixed).

```
import { withPrefix } from "gatsby"

const IndexLayout = ({ children, location }) => {
  const isHomepage = location.pathname === withPrefix("/")

  return (
    <div>
      <h1>Welcome {isHomepage ? "home" : "aboard"}!</h1>
      {children}
    </div>
  )
}
```

Reminder: use `<Link>` only for internal links!

This component is intended *only* for links to pages handled by Gatsby. For links to pages on other domains or pages on the same domain not handled by the current Gatsby site, use the normal `<a>` element.

Sometimes you won't know ahead of time whether a link will be internal or not, such as when the data is coming from a CMS. In these cases you may find it useful to make a component which inspects the link and renders either with Gatsby's `<Link>` or with a regular `<a>` tag accordingly.

Since deciding whether a link is internal or not depends on the site in question, you may need to customize the heuristic to your environment, but the following

may be a good starting point:

```
import { Link as GatsbyLink } from "gatsby"

// Since DOM elements <a> cannot receive activeClassName
// and partiallyActive, destructure the prop here and
// pass it only to GatsbyLink
const Link = ({ children, to, activeClassName, partiallyActive, ...other }) => {
  // Tailor the following test to your environment.
  // This example assumes that any internal link (intended for Gatsby)
  // will start with exactly one slash, and that anything else is external.
  const internal = /^\/(?!\/)/.test(to)

  // Use Gatsby Link for internal links, and <a> for others
  if (internal) {
    return (
      <GatsbyLink
        to={to}
        activeClassName={activeClassName}
        partiallyActive={partiallyActive}
        {...other}
      >
        {children}
      </GatsbyLink>
    )
  }
  return (
    <a href={to} {...other}>
      {children}
    </a>
  )
}

export default Link
```

Relative links

The `<Link />` component follows the behavior of @reach/router by ignoring trailing slashes and treating each page as if it were a directory when resolving relative links. For example if you are on either `/blog/my-great-page` or `/blog/my-great-page/` (note the trailing slash), a link to `../second-page` will take you to `/blog/second-page`.

File Downloads

You can similarly check for file downloads:

```
  const file = /\.[0-9a-z]+$/i.test(to)

  ...

  if (internal) {
    if (file) {
        return (
          <a href={to} {...other}>
            {children}
          </a>
      )
    }
    return (
      <GatsbyLink to={to} {...other}>
        {children}
      </GatsbyLink>
    )
  }
```

Recommendations for programmatic, in-app navigation

If you need this behavior, you should either use an anchor tag or import the `navigate` helper from `gatsby`, like so:

```
import { navigate } from 'gatsby';

...

onClick = () => {
  navigate('#some-link');
  // OR
  navigate('?foo=bar');
}
```

Handling stale client-side pages

Gatsby's `<Link>` component will only fetch each page's resources once. Updates to pages on the site are not reflected in the browser as they are effectively "locked in time". This can have the undesirable impact of different users having different views of the content.

In order to prevent this staleness, Gatsby requests an additional resource on each new page load: `app-data.json`. This contains a hash generated when the site is built; if anything in the `src` directory changes, the hash will change. During page loads, if Gatsby sees a different hash in the `app-data.json` than the hash it initially retrieved when the site first loaded, the browser will navigate using

`window.location`. The browser fetches the new page and starts over again, so any cached resources are lost.

However, if the page has previously loaded, it will not re-request `app-data.json`. In that case, the hash comparison will not occur and the previously loaded content will be used.

> **Note:** Any state will be lost during the `window.location` transition. This can have an impact if there is a reliance on state management, e.g. tracking state in wrapPageElement or via a library like Redux.

Additional resources

- Authentication tutorial for client-only routes
- Routing: Getting Location Data from Props
- `gatsby-plugin-catch-links` to automatically intercept local links in Markdown files for `gatsby-link` like behavior