Navigator

- Usage
  - Hierarchical paths
  - Height and animations
  - Individual components
    - Navigator
      - Props
        - initialPath: string
        - children: string
    - Navigator.Screen
      - Props
        - path: string
        - children: string
    - Navigator.Button
      - Props
        - path: string
        - attributeName: string
        - children: string
        - Inherited props
    - Navigator.BackButton
      - Props
        - children: string
        - Inherited props
    - useNavigator
      - Props
        - goTo: (path: string, options: NavigateOptions) => void
        - goBack: (path: string, options: NavigateOptions) => void
        - location: NavigatorLocation
        - params: Record< string, string | string[] >

`Navigator` is a collection components that allow rendering nested views/panels/menus (via the `Navigator.Screen` component) and navigate between them (via the `Navigator.Button` and `Navigator.BackButton` components).

Usage

```
import { Navigator } from '@wordpress/components';

const MyNavigation = () => (
    <Navigator initialPath="/">
        <Navigator.Screen path="/">
            <p>This is the home screen.</p>
            <Navigator.Button path="/child">
                Navigate to child screen.
            </Navigator.Button>
        </Navigator.Screen>
        <Navigator.Screen path="/child">
            <p>This is the child screen.</p>
            <Navigator.BackButton>Go back</Navigator.BackButton>
        </Navigator.Screen>
    </Navigator>
);
```

Hierarchical `path`s

`Navigator` assumes that screens are organized hierarchically according to their `path`, which should follow a URL-like scheme where each path segment starts with and is separated by the `/` character.

`Navigator` will treat “back” navigations as going to the parent screen — it is, therefore, the responsibility of the consumer of the component to create the correct screen hierarchy.

For example:

- `/` is the root of all paths. There should always be a screen with `path="/"`;
- `/parent/child` is a child of `/parent`;
- `/parent/child/grand-child` is a child of `/parent/child`;
- `/parent/:param` is a child of `/parent` as well;
- if the current screen has a `path="/parent/child/grand-child"`, when going “back” `Navigator` will try to recursively navigate the path hierarchy until a matching screen (or

the root `/`) is found.

Height and animations

Due to how `Navigator.Screen` animations work, it is recommended that the `Navigator` component is assigned a `height` to prevent some potential UI jumps while moving across screens.

Individual components

`Navigator` is comprised of four individual components:

- `Navigator`: a wrapper component and context provider. It holds the main logic for hiding and showing screens.
- `Navigator.Screen`: represents a single view/screen/panel;
- `Navigator.Button`: renders a button that allows navigating to a different `Navigator.Screen`;
- `Navigator.BackButton`: renders a button that allows navigating to the parent `Navigator.Screen` (see the section above about hierarchical paths).

For advanced usages, consumers can use the `useNavigator` hook.

`Navigator`

Props

`initialPath`: `string`

The initial active path.

- Required: Yes

`children`: `string`

The children elements.

- Required: Yes

`Navigator.Screen`

Props

`path`: `string`

The screen's path, matched against the current path stored in the navigator.

`Navigator` assumes that screens are organized hierarchically according to their `path`, which should follow a URL-like scheme where each path segment starts with and is separated by the `/` character.

`Navigator` will treat “back” navigations as going to the parent screen — it is, therefore, the responsibility of the consumer of the component to create the correct screen hierarchy.

For example:

- `/` is the root of all paths. There should always be a screen with `path="/"`.
- `/parent/child` is a child of `/parent`.
- `/parent/child/grand-child` is a child of `/parent/child`.
- `/parent/:param` is a child of `/parent` as well.
- if the current screen has a `path` with value `/parent/child/grand-child`, when going “back” `Navigator` will try to recursively navigate the path hierarchy until a matching screen (or the root `/`) is found.
- Required: Yes

`children`: `string`

The children elements.

- Required: Yes

`Navigator.Button`

Props

`path`: `string`

The path of the screen to navigate to. The value of this prop needs to be a valid value for an HTML attribute.

- Required: Yes

`attributeName`: `string`

The HTML attribute used to identify the `Navigator.Button`, which is used by `Navigator` to restore focus.

- Required: No
- Default: `id`

`children`: `string`

The children elements.

- Required: No

Inherited props

`Navigator.Button` also inherits all of the `Button` props, except for `href` and `target`.

`Navigator.BackButton`

Props

`children`: `string`

The children elements.

- Required: No

Inherited props

`Navigator.BackButton` also inherits all of the `Button` props, except for `href` and `target`.

`useNavigator`

You can retrieve a `navigator` instance by using the `useNavigator` hook.

Props

The `navigator` instance has a few properties:

`goTo`: `( path: string, options: NavigateOptions ) => void`

The `goTo` function allows navigating to a given path. The second argument can augment the navigation operations with different options.

The available options are:

- `focusTargetSelector`: `string`. An optional property used to specify the CSS selector used to restore focus on the matching element when navigating back;
- `isBack`: `boolean`. An optional property used to specify whether the navigation should be considered as backwards (thus enabling focus restoration when possible, and causing the animation to be backwards too);
- `skipFocus`: `boolean`. An optional property used to opt out of `Navigator`'s focus management, useful when the consumer of the component wants to manage focus themselves;

`goBack`: `( path: string, options: NavigateOptions ) => void`

The `goBack` function allows navigating to the parent screen. Parent/child navigation only works if the paths you define are hierarchical (see note above).

When a match is not found, the function will try to recursively navigate the path hierarchy until a matching screen (or the root `/`) is found.

The available options are the same as for the `goTo` method, except for the `isBack` property, which is not available for the `goBack` method.

`location`: `NavigatorLocation`

The `location` object represents the current location, and has a few properties:

- `path`: `string`. The path associated to the location.

- `isBack`: `boolean`. A flag that is `true` when the current location was reached by navigating backwards.
- `isInitial`: `boolean`. A flag that is `true` only for the initial location.

`params`: `Record< string, string | string[] >`

The parsed record of parameters from the current location. For example if the current screen path is `/product/:productId` and the location is `/product/123`, then `params` will be `{ productId: '123' }`.

□□□ #2
□ □□□ □□□ 2026-10-31 18:16:33 UTC
□ □□□ □□□ 2026-10-31 18:16:34 UTC