Version: 6.x

Navigation state reference

The navigation state is the state where React Navigation stores the navigation structure and history of the app. It's useful to know about the structure of the navigation state if you need to do advanced operations such as resetting the state, providing a custom initial state etc.

It's a JavaScript object which looks like this:

```
const state = {
  type: 'stack',
  key: 'stack-1',
  routeNames: ['Home', 'Profile', 'Settings'],
  routes: [
    { key: 'home-1', name: 'Home', params: { sortBy: 'latest' } },
    { key: 'settings-1', name: 'Settings' },
  ],
  index: 1,
  stale: false,
};
```

There are few properties present in every navigation state object:

- `type` - Type of the navigator that the state belongs to, e.g. `stack`, `tab`, `drawer`.
- `key` - Unique key to identify the navigator.
- `routeNames` - Name of the screens defined in the navigator. This is an unique array containing strings for each screen.
- `routes` - List of route objects (screens) which are rendered in the navigator. It also represents the history in a stack navigator. There should be at least one item present in this array.
- `index` - Index of the focused route object in the `routes` array.
- `history` - A list of visited items. This is an optional property and not present in all navigators. For example, it's only present in tab and drawer navigators in the core. The shape of the items in the `history` array can vary depending on the navigator. There should be at least one item present in this array.

- `stale` - A navigation state is assumed to be stale unless the `stale` property is explicitly set to `false`. This means that the state object needs to be "rehydrated".

Each route object in a `routes` array may contain the following properties:

- `key` - Unique key of the screen. Created automatically or added while navigating to this screen.
- `name` - Name of the screen. Defined in navigator component hierarchy.
- `params` - An optional object containing params which is defined while navigating e.g. `navigate('Home', { sortBy: 'latest' })`.
- `state` - An optional object containing the navigation state of a child navigator nested inside this screen.

For example, a stack navigator containing a tab navigator nested inside it's home screen may have a navigation state object like this:

```
const state = {
  type: 'stack',
  key: 'stack-1',
  routeNames: ['Home', 'Profile', 'Settings'],
  routes: [
    {
      key: 'home-1',
      name: 'Home',
      state: {
        key: 'tab-1',
        routeNames: ['Feed', 'Library', 'Favorites'],
        routes: [
          { key: 'feed-1', name: 'Feed', params: { sortBy: 'latest' } },
          { key: 'library-1', name: 'Library' },
          { key: 'favorites-1', name: 'Favorites' },
        ],
        index: 0,
      },
    },
    { key: 'settings-1', name: 'Settings' },
  ],
  index: 1,
};
```

It's important to note that even if there's a nested navigator, the `state` property on the `route` object is not added until a navigation happens, hence it's not guaranteed to exist.

Partial state objects

Earlier there was a mention of `stale` property in the navigation state. A stale navigation state means that the state object needs to be rehydrated or fixed or fixed up, such as adding missing keys, removing invalid screens etc. before being used. As a user, you don't need to worry about it, React Navigation will fix up any issues in a state object automatically unless `stale` is set to `false`. If you're writing a custom router, the `getRehydratedState` method let's you write custom rehydration logic to fix up state objects.

This also applies to the `index` property: `index` should be the last route in a stack, and if a different value was specified, React Navigation fixes it. For example, if you wanted to reset your app's navigation state to have it display the `Profile` route, and have the `Home` route displayed upon going back, and did the below,

```
navigation.reset({
  index: 0,
  routes: [
    { name: 'Home' },
    { name: 'Profile' }
  ],
});
```

React Navigation would correct `index` to 1, and display the route and perform navigation as intended.

This feature comes handy when doing operations such as reset, providing a initial state etc., as you can safely omit many properties from the navigation state object and relying on React Navigation to add those properties for you, making your code simpler. For example, you can only provide a `routes` array without any keys and React Navigation will automatically add everything that's needed to make it work:

```
const state = {
  routes: [{ name: 'Home' }, { name: 'Profile' }],
```

```
};
```

After rehydration, it'll look something like this:

```
const state = {
  type: 'stack',
  key: 'stack-1',
  routeNames: ['Home', 'Profile', 'Settings'],
  routes: [
    { key: 'home-1', name: 'Home' },
    { key: 'settings-1', name: 'Settings' },
  ],
  index: 1,
  stale: false,
};
```

Here, React Navigation filled in the missing bits such as keys, route names, index etc.

It's also possible to provide invalid data such as non-existent screens and it'll be fixed automatically. While it's not recommended to write code with invalid state objects, it can be super useful if you do things like state persistence, where the configured screens might have changed after an update, which could cause problems if React Navigation didn't fix the state object automatically.

> If you want React Navigation to fix invalid state, you need to make sure that you don't have `stale: false` in the state object. State objects with `stale: false` are assumed to be valid state objects and React Navigation won't attempt to fix them.

When you're providing a state object in `initialState`, React Navigation will always assume that it's a stale state object, which makes sure that things like state persistence work smoothly without extra manipulation of the state object.

Edit this page