Version: 6.x

CommonActions reference

A navigation action is an object containing at least a `type` property. Internally, the action can be handled by routers with the `getStateForAction` method to return a new state from an existing navigation state.

Each navigation actions can contain at least the following properties:

- `type` (required) - A string which represents the name of the action.
- `payload` (options) - An object containing additional information about the action. For example, it will contain `name` and `params` for `navigate`.
- `source` (optional) - The key of the route which should be considered as the source of the action. This is used for some actions to determine which route to apply the action on. By default, `navigation.dispatch` adds the key of the route that dispatched the action.
- `target` (optional) - The key of the navigation state the action should be applied on.

It's important to highlight that dispatching a navigation action doesn't throw any error when the action is unhandled (similar to when you dispatch an action that isn't handled by a reducer in redux and nothing happens).

Common actions

The library exports several action creators under the `CommonActions` namespace. You should use these action creators instead of writing action objects manually.

navigate

The `navigate` action allows to navigate to a specific route. It takes the following arguments:

- `name` - *string* - A destination name of the route that has been registered somewhere..
- `key` - *string* - The identifier for the route to navigate to. Navigate back to this route if it already exists..

- `params` - *object* - Params to merge into the destination route..

The options object passed should at least contain a `key` or `name` property, and optionally `params`. If both `key` and `name` are passed, stack navigator will create a new route with the specified key if no matches were found.

```
import { CommonActions } from '@react-navigation/native';

navigation.dispatch(
  CommonActions.navigate({
    name: 'Profile',
    params: {
      user: 'jane',
    },
  })
);
```

Try this example on Snack

In a stack navigator, calling `navigate` with a screen name will result in different behavior based on if the screen is already present or not. If the screen is already present in the stack's history, it'll go back to that screen and remove any screens after that. If the screen is not present, it'll push a new screen.

By default, the screen is identified by its name. But you can also customize it to take the params into account by using the `getId` prop.

reset

The `reset` action allows to reset the navigation state to the given state. It takes the following arguments:

- `state` - *object* - The new navigation state object to use.

```
import { CommonActions } from '@react-navigation/native';

navigation.dispatch(
  CommonActions.reset({
    index: 1,
```

```
    routes: [
      { name: 'Home' },
      {
        name: 'Profile',
        params: { user: 'jane' },
      },
    ],
  })
);
```

Try this example on Snack

The state object specified in `reset` replaces the existing navigation state with the new one. This means that if you provide new route objects without a key, or route objects with a different key, it'll remove the existing screens for those routes and add new screens.

If you want to preserve the existing screens but only want to modify the state, you can pass a function to `dispatch` where you can get the existing state. Then you can change it as you like (make sure not to mutate the existing state, but create new state object for your changes). and return a `reset` action with the desired state:

```
import { CommonActions } from '@react-navigation/native';

navigation.dispatch(state => {
  // Remove the home route from the stack
  const routes = state.routes.filter(r => r.name !== 'Home');

  return CommonActions.reset({
    ...state,
    routes,
    index: routes.length - 1,
  });
});
```

> Note: Consider the navigator's state object to be internal and subject to change in a minor release. Avoid using properties from the navigation state state object except `index` and `routes`, unless you really need it. If there is some functionality you cannot achieve without relying on the structure of the state object, please open an issue.

goBack

The `goBack` action creator allows to go back to the previous route in history. It doesn't take any arguments.

```
import { CommonActions } from '@react-navigation/native';

navigation.dispatch(CommonActions.goBack());
```

Try this example on Snack

If you want to go back from a particular route, you can add a `source` property referring to the route key and a `target` property referring to the `key` of the navigator which contains the route:

```
import { CommonActions } from '@react-navigation/native';

navigation.dispatch({
  ...CommonActions.goBack(),
  source: route.key,
  target: state.key,
});
```

Try this example on Snack

By default, the key of the route which dispatched the action is passed as the `source` property and the `target` property is `undefined`.

setParams

The `setParams` action allows to update params for a certain route. It takes the following arguments:

- `params` - *object* - required - New params to be merged into existing route params.

```
import { CommonActions } from '@react-navigation/native';

navigation.dispatch(CommonActions.setParams({ user: 'Wojtek' }));
```

Try this example on Snack

If you want to set params for a particular route, you can add a `source` property referring to the route key:

```
import { CommonActions } from '@react-navigation/native';

navigation.dispatch({
  ...CommonActions.setParams({ user: 'Wojtek' }),
  source: route.key,
});
```

Try this example on Snack

If the `source` property is explicitly set to `undefined`, it'll set the params for the focused route.

Edit this page