Version: 6.x

Type checking with TypeScript

React Navigation is written with TypeScript and exports type definitions for TypeScript projects.

Type checking the navigator

To type check our route name and params, the first thing we need to do is to create an object type with mappings for route name to the params of the route. For example, say we have a route called `Profile` in our root navigator which should have a param `userId`:

```
type RootStackParamList = {
  Profile: { userId: string };
};
```

Similarly, we need to do the same for each route:

```
type RootStackParamList = {
  Home: undefined;
  Profile: { userId: string };
  Feed: { sort: 'latest' | 'top' } | undefined;
};
```

Specifying `undefined` means that the route doesn't have params. A union type with `undefined` (e.g. `SomeType | undefined`) means that params are optional.

After we have defined the mappings, we need to tell our navigator to use it. To do that, we can pass it as a generic to the `createXNavigator` functions:

```
import { createStackNavigator } from '@react-navigation/stack';

const RootStack = createStackNavigator<RootStackParamList>();
```

And then we can use it:

```
<RootStack.Navigator initialRouteName="Home">
  <RootStack.Screen name="Home" component={Home} />
  <RootStack.Screen
    name="Profile"
    component={Profile}
    initialParams={{ userId: user.id }}
  />
  <RootStack.Screen name="Feed" component={Feed} />
</RootStack.Navigator>
```

This will provide type checking and intelliSense for props of the `Navigator` and `Screen` components.

> Note: The type containing the mappings must be a type alias (e.g. `type RootStackParamList = { ... }`). It cannot be an interface (e.g. `interface RootStackParamList { ... }`). It also shouldn't extend `ParamListBase` (e.g. `interface RootStackParamList extends ParamListBase { ... }`). Doing so will result in incorrect type checking where it allows you to pass incorrect route names.

Type checking screens

To type check our screens, we need to annotate the `navigation` prop and the `route` prop received by a screen. The navigator packages in React Navigation export a generic types to define types for both the `navigation` and `route` props from the corresponding navigator.

For example, you can use `NativeStackScreenProps` for the Native Stack Navigator.

```
import type { NativeStackScreenProps } from '@react-navigation/native-stack';

type RootStackParamList = {
  Home: undefined;
  Profile: { userId: string };
  Feed: { sort: 'latest' | 'top' } | undefined;
};

type Props = NativeStackScreenProps<RootStackParamList, 'Profile'>;
```

The type takes 3 generics:

- The param list object we defined earlier
- The name of the route the screen belongs to
- The ID of the navigator (optional)

If you have an `id` prop for your navigator, you can do:

```
type Props = NativeStackScreenProps<RootStackParamList, 'Profile', 'MyStack'>;
```

This allows us to type check route names and params which you're navigating using `navigate`, `push` etc. The name of the current route is necessary to type check the params in `route.params` and when you call `setParams`.

Similarly, you can import `StackScreenProps` for `@react-navigation/stack`, `DrawerScreenProps` from `@react-navigation/drawer`, `BottomTabScreenProps` from `@react-navigation/bottom-tabs` and so on.

Then you can use the `Props` type you defined above to annotate your component.

For function components:

```
function ProfileScreen({ route, navigation }: Props) {
  // ...
}
```

For class components:

```
class ProfileScreen extends React.Component<Props> {
  render() {
    // ...
  }
}
```

You can get the types for `navigation` and `route` from the `Props` type as follows:

```
type ProfileScreenNavigationProp = Props['navigation'];
```

```
type ProfileScreenRouteProp = Props['route'];
```

Alternatively, you can also annotate the `navigation` and `route` props separately.

To get the type for the `navigation` prop, we need to import the corresponding type from the navigator. For example, `NativeStackNavigationProp` for `@react-navigation/native-stack`:

```
import type { NativeStackNavigationProp } from '@react-navigation/native-stack';

type ProfileScreenNavigationProp = NativeStackNavigationProp<
  RootStackParamList,
  'Profile'
>;
```

Similarly, you can import `StackNavigationProp` from `@react-navigation/stack`, `DrawerNavigationProp` from `@react-navigation/drawer`, `BottomTabNavigationProp` from `@react-navigation/bottom-tabs` etc.

To get the type for the `route` prop, we need to use the `RouteProp` type from `@react-navigation/native`:

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

type ProfileScreenRouteProp = RouteProp<RootStackParamList, 'Profile'>;
```

We recommend creating a separate `types.tsx` file where you keep the types and import them in your component files instead of repeating them in each file.

Nesting navigators

Type checking screens and params in nested navigator

You can navigate to a screen in a nested navigator by passing `screen` and `params` properties for the nested screen:

```
navigation.navigate('Home', {
  screen: 'Feed',
```

```
  params: { sort: 'latest' },
});
```

To be able to type check this, we need to extract the params from the screen containing the nested navigator. This can be done using the `NavigatorScreenParams` utility:

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

type TabParamList = {
  Home: NavigatorScreenParams<StackParamList>;
  Profile: { userId: string };
};
```

Combining navigation props

When you nest navigators, the navigation prop of the screen is a combination of multiple navigation props. For example, if we have a tab inside a stack, the `navigation` prop will have both `jumpTo` (from the tab navigator) and `push` (from the stack navigator). To make it easier to combine types from multiple navigators, you can use the `CompositeScreenProps` type:

```
import type { CompositeScreenProps } from '@react-navigation/native';
import type { BottomTabScreenProps } from '@react-navigation/bottom-tabs';
import type { StackScreenProps } from '@react-navigation/stack';

type ProfileScreenProps = CompositeScreenProps<
  BottomTabScreenProps<TabParamList, 'Profile'>,
  StackScreenProps<StackParamList>
>;
```

The `CompositeScreenProps` type takes 2 parameters, first parameter is the type of props for the primary navigation (type for the navigator that owns this screen, in our case the tab navigator which contains the `Profile` screen) and second parameter is the type of props for secondary navigation (type for a parent navigator). The primary type should always have the screen's route name as its second parameter.

For multiple parent navigators, this secondary type should be nested:

```
type ProfileScreenProps = CompositeScreenProps<
  BottomTabScreenProps<TabParamList, 'Profile'>,
  CompositeScreenProps<
    StackScreenProps<StackParamList>,
    DrawerScreenProps<DrawerParamList>
  >
>;
```

If annotating the `navigation` prop separately, you can use `CompositeNavigationProp` instead. The usage is similar to `CompositeScreenProps`:

```
import type { CompositeNavigationProp } from '@react-navigation/native';
import type { BottomTabNavigationProp } from '@react-navigation/bottom-tabs';
import type { StackNavigationProp } from '@react-navigation/stack';

type ProfileScreenNavigationProp = CompositeNavigationProp<
  BottomTabNavigationProp<TabParamList, 'Profile'>,
  StackNavigationProp<StackParamList>
>;
```

Annotating `useNavigation`

To annotate the `navigation` prop that we get from `useNavigation`, we can use a type parameter:

```
const navigation = useNavigation<ProfileScreenNavigationProp>();
```

It's important to note that this isn't completely type-safe because the type parameter you use may not be correct and we cannot statically verify it.

Annotating `useRoute`

To annotate the `route` prop that we get from `useRoute`, we can use a type parameter:

```
const route = useRoute<ProfileScreenRouteProp>();
```

It's important to note that this isn't completely type-safe, similar to `useNavigation`.

Annotating `options` and `screenOptions`

When you pass the `options` to a `Screen` or `screenOptions` prop to a `Navigator` component, they are already type-checked and you don't need to do anything special. However, sometimes you might want to extract the options to a separate object, and you might want to annotate it.

To annotate the options, we need to import the corresponding type from the navigator. For example, `StackNavigationOptions` for `@react-navigation/stack`:

```
import type { StackNavigationOptions } from '@react-navigation/stack';

const options: StackNavigationOptions = {
  headerShown: false,
};
```

Similarly, you can import `DrawerNavigationOptions` from `@react-navigation/drawer`, `BottomTabNavigationOptions` from `@react-navigation/bottom-tabs` etc.

When using the function form of `options` and `screenOptions`, you can annotate the arguments with the same type you used to annotate the `navigation` and `route` props.

Annotating `ref` on `NavigationContainer`

If you use the `createNavigationContainerRef()` method to create the ref, you can annotate it to type-check navigation actions:

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

// ...

const navigationRef = createNavigationContainerRef<RootStackParamList>();
```

Similarly, for `useNavigationContainerRef()`:

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

// ...
```

```
const navigationRef = useNavigationContainerRef<RootStackParamList>();
```

If you're using a regular `ref` object, you can pass a generic to the `NavigationContainerRef` type..

Example when using `React.useRef` hook:

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

// ...

const navigationRef =
  React.useRef<NavigationContainerRef<RootStackParamList>>(null);
```

Example when using `React.createRef`:

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

// ...

const navigationRef =
  React.createRef<NavigationContainerRef<RootStackParamList>>();
```

Specifying default types for `useNavigation`, `Link`, `ref` etc

Instead of manually annotating these APIs, you can specify a global type for your root navigator which will be used as the default type.

To do this, you can add this snippet somewhere in your codebase:

```
declare global {
  namespace ReactNavigation {
    interface RootParamList extends RootStackParamList {}
  }
}
```

The `RootParamList` interface lets React Navigation know about the params accepted by your root navigator. Here we extend the type `RootStackParamList` because that's the type of params for our stack navigator at the root. The name of this type isn't important.

Specifying this type is important if you heavily use `useNavigation`, `Link` etc. in your app since it'll ensure type-safety. It will also make sure that you have correct nesting on the `linking` prop.

Organizing types

When writing types for React Navigation, there are a couple of things we recommend to keep things organized.

1. It's good to create a separate files (e.g. `navigation/types.tsx`) which contains the types related to React Navigation.
2. Instead of using `CompositeNavigationProp` directly in your components, it's better to create a helper type that you can reuse.
3. Specifying a global type for your root navigator would avoid manual annotations in many places.

Considering these recommendations, the file containing the types may look something like this:

```
import type {
  CompositeScreenProps,
  NavigatorScreenParams,
} from '@react-navigation/native';
import type { StackScreenProps } from '@react-navigation/stack';
import type { BottomTabScreenProps } from '@react-navigation/bottom-tabs';

export type RootStackParamList = {
  Home: NavigatorScreenParams<HomeTabParamList>;
  PostDetails: { id: string };
  NotFound: undefined;
};

export type RootStackScreenProps<T extends keyof RootStackParamList> =
  StackScreenProps<RootStackParamList, T>;

export type HomeTabParamList = {
  Popular: undefined;
  Latest: undefined;
```

```
};

export type HomeTabScreenProps<T extends keyof HomeTabParamList> =
  CompositeScreenProps<
    BottomTabScreenProps<HomeTabParamList, T>,
    RootStackScreenProps<keyof RootStackParamList>
  >;

declare global {
  namespace ReactNavigation {
    interface RootParamList extends RootStackParamList {}
  }
}
```

Now, when annotating your components, you can write:

```
import type { HomeTabScreenProps } from './navigation/types';

function PopularScreen({ navigation, route }: HomeTabScreenProps<'Popular'>) {
  // ...
}
```

If you're using hooks such as `useRoute`, you can write:

```
import type { HomeTabScreenProps } from './navigation/types';

function PopularScreen() {
  const route = useRoute<HomeTabScreenProps<'Popular'>['route']>();

  // ...
}
```

Edit this page