# React Redux TypeScript Quick Start

**WHAT YOU'LL LEARN**

- How to set up and use Redux Toolkit and React Redux with TypeScript

**PREREQUISITES**

- Knowledge of React Hooks
- Understanding of Redux terms and concepts
- Understanding of TypeScript syntax and concepts

## Introduction

Welcome to the React Redux TypeScript Quick Start tutorial! **This tutorial will briefly show how to use TypeScript with Redux Toolkit and React-Redux**.

This page focuses on just how to set up the TypeScript aspects. For explanations of what Redux is, how it works, and full examples of how to use Redux, see the Redux core docs tutorials.

React Redux is also written in TypeScript as of version 8, and also includes its own type definitions.

The Redux+TS template for Create-React-App comes with a working example of these patterns already configured.

**INFO**

The recently updated `@types/react@18` major version has changed component definitions to remove having `children` as a prop by default. This causes errors if you have multiple copies of `@types/react` in your project. To fix this, tell your package manager to resolve `@types/react` to a single version. Details:

> https://github.com/facebook/react/issues/24304#issuecomment-1094565891

# Project Setup

## Define Root State and Dispatch Types

Redux Toolkit's `configureStore` API should not need any additional typings. You will, however, want to extract the `RootState` type and the `Dispatch` type so that they can be referenced as needed. Inferring these types from the store itself means that they correctly update as you add more state slices or modify middleware settings.

Since those are types, it's safe to export them directly from your store setup file such as `app/store.ts` and import them directly into other files.

app/store.ts

```
import { configureStore } from '@reduxjs/toolkit'
// ...

const store = configureStore({
  reducer: {
    posts: postsReducer,
    comments: commentsReducer,
    users: usersReducer,
  },
})

// Infer the `RootState` and `AppDispatch` types from the store itself
export type RootState = ReturnType<typeof store.getState>
// Inferred type: {posts: PostsState, comments: CommentsState, users:
UsersState}
export type AppDispatch = typeof store.dispatch
```

## Define Typed Hooks

While it's possible to import the `RootState` and `AppDispatch` types into each component, it's **better to create typed versions of the `useDispatch` and `useSelector` hooks for usage in your application**. This is important for a couple reasons:

- For `useSelector`, it saves you the need to type `(state: RootState)` every time

- For `useDispatch`, the default `Dispatch` type does not know about thunks. In order to correctly dispatch thunks, you need to use the specific customized `AppDispatch` type from the store that includes the thunk middleware types, and use that with `useDispatch`. Adding a pre-typed `useDispatch` hook keeps you from forgetting to import `AppDispatch` where it's needed.

Since these are actual variables, not types, it's important to define them in a separate file such as `app/hooks.ts`, not the store setup file. This allows you to import them into any component file that needs to use the hooks, and avoids potential circular import dependency issues.

app/hooks.ts

```ts
import { useDispatch, useSelector } from 'react-redux'
import type { RootState, AppDispatch } from './store'

// Use throughout your app instead of plain `useDispatch` and `useSelector`
export const useAppDispatch = useDispatch.withTypes<AppDispatch>()
export const useAppSelector = useSelector.withTypes<RootState>()
```

# Application Usage

## Define Slice State and Action Types

Each slice file should define a type for its initial state value, so that `createSlice` can correctly infer the type of `state` in each case reducer.

All generated actions should be defined using the `PayloadAction<T>` type from Redux Toolkit, which takes the type of the `action.payload` field as its generic argument.

You can safely import the `RootState` type from the store file here. It's a circular import, but the TypeScript compiler can correctly handle that for types. This may be needed for use cases like writing selector functions.

features/counter/counterSlice.ts

```ts
import { createSlice, PayloadAction } from '@reduxjs/toolkit'
import type { RootState } from '../../app/store'

// Define a type for the slice state
interface CounterState {
  value: number
```

```
}

// Define the initial state using that type
const initialState: CounterState = {
  value: 0,
}

export const counterSlice = createSlice({
  name: 'counter',
  // `createSlice` will infer the state type from the `initialState` argument
  initialState,
  reducers: {
    increment: (state) => {
      state.value += 1
    },
    decrement: (state) => {
      state.value -= 1
    },
    // Use the PayloadAction type to declare the contents of `action.payload`
    incrementByAmount: (state, action: PayloadAction<number>) => {
      state.value += action.payload
    },
  },
})

export const { increment, decrement, incrementByAmount } =
counterSlice.actions

// Other code such as selectors can use the imported `RootState` type
export const selectCount = (state: RootState) => state.counter.value

export default counterSlice.reducer
```

The generated action creators will be correctly typed to accept a `payload` argument based on the `PayloadAction<T>` type you provided for the reducer. For example, `incrementByAmount` requires a `number` as its argument.

In some cases, TypeScript may unnecessarily tighten the type of the initial state. If that happens, you can work around it by casting the initial state using `as`, instead of declaring the type of the variable:

```
// Workaround: cast state instead of declaring variable type
const initialState = {
```

```
  value: 0,
} as CounterState
```

## Use Typed Hooks in Components

In component files, import the pre-typed hooks instead of the standard hooks from React-Redux.

features/counter/Counter.tsx

```
import React, { useState } from 'react'

import { useAppSelector, useAppDispatch } from 'app/hooks'

import { decrement, increment } from './counterSlice'

export function Counter() {
  // The `state` arg is correctly typed as `RootState` already
  const count = useAppSelector((state) => state.counter.value)
  const dispatch = useAppDispatch()

  // omit rendering logic
}
```

# What's Next?

See the "Usage with TypeScript" page for extended details on how to use Redux Toolkit's APIs with TypeScript.

Edit this page

*Last updated on **Sep 13, 2024***