

Redux Cheat Sheet (3.2.1)

```
import React from 'react'
import ReactDOM from 'react-dom'
import { createStore, combineReducers,
  applyMiddleware, bindActionCreators } from 'redux'

const greetingReducer = (state='', action) => {
  switch (action.type) {
    case 'SAY_HELLO': return 'Hello '
    case 'SAY_GOODBYE': return 'Goodbye '
  }
  return state
}

const nameReducer = (state='John', action) => {
  switch (action.type) {
    case 'CHANGE_NAME': return 'Joel'
  }
  return state
}

const actionLogger = ({dispatch, getState}) =>
  (next) => (action) =>
    { console.log(action); return next(action) }

const reducers = combineReducers({
  greeting: greetingReducer,
  name: nameReducer
})

const middleware = applyMiddleware(actionLogger)
const store = createStore(
  reducers,
  { greeting: '(Roll over me) ' },
  middleware
)

const changeName = () => {return { type: 'CHANGE_NAME' }}
const hello = () => {return { type: 'SAY_HELLO' }}
const goodbye = () => {return { type: 'SAY_GOODBYE' }}

const Hello = (props) =>
  <div
    onMouseOver={props.hello}
    onMouseOut={props.goodbye}
    onClick={props.changeName}>
    {props.greeting}{props.name}
  </div>

const render = () => {
  ReactDOM.render(
    <Hello
      greeting={store.getState().greeting}
      name={store.getState().name}
      {...bindActionCreators({changeName, hello, goodbye},
        store.dispatch)}
    />,
    document.getElementById('root')
  )
}

render()
store.subscribe(render)
```

Welcome to the egghead.io Redux cheat sheet! On your left you will find a full-fledged Redux application with a React.js front-end (React is not required).

function reducer(**STATE**, **ACTION**) => **State**

Takes the previous state and an action, and returns the next state.

Splitting your app into multiple reducers (**greetingsReducer**, **nameReducer**) allows for a clean separation of concerns when modifying your application's state.

function middleware(**{DISPATCH, GETSTATE}**) => **next** => **action**

Receives Store's **dispatch** and **getState** functions as named arguments, and returns a function. That function will be given the next middleware's dispatch method, and is expected to return a function of action calling **next(action)** with a potentially different argument, or at a different time, or maybe not calling it at all. The last middleware in the chain will receive the real store's **dispatch** method as the next parameter, thus ending the chain.

combineReducers(**{REDUCERS}**) => **Function**

Combines multiple reducers into a single reducing function with each reducer as a key/value pair. Can then be passed to **createStore()**.

applyMiddleware(**...MIDDLEWARES**) => **Function**

Extends Redux with custom functionality by wrapping the store's dispatch method.

createStore(**REDUCER**, **?INITIALSTATE**, **?ENHANCER**) => **Store**

Creates a Redux store that holds the complete state tree of your app. There should only be a single store in your app.

store = { ... }

Brings together your application's state and has the following responsibilities:

- Allows access to state via **getState()**;
- Allows state to be updated via **dispatch(action)**;
- Registers listeners via **subscribe(listener)**;
- Handles unregistering of listeners via the function returned by **subscribe(listener)**.

action = { type: **String**, ...payload: **any** }

Holds action payloads in plain javascript objects. Must have a type property that indicates the performed action, typically be defined as string constants. All other properties are the action's payload.

function actionCreator(**?ANY**) => **Action|AsyncAction**

Creates an action with optional payload and bound dispatch.

bindActionCreators(**ACTIONCREATORS**, **DISPATCH**) => **Fn | Obj**

Turns an object whose values are action creators, into an object with the same keys, but with every action creator wrapped into a dispatch call so they may be invoked directly.

Redux's Three Principles

- Single source of truth
- State is read-only
- Changes are made with pure functions

Glossary

State

type **State** = **any**

Action

type **Action** = { **TYPE: STRING**, **PAYLOAD: ANY** }

Reducer

type **Reducer**<**State**, **Action**> = (**STATE**, **ACTION**) => **State**

Dispatching Functions

type **BaseDispatch** = (**ACTION**) => **Action**

type **Dispatch** = (**ACTION | ASYNCACTION**) => **any**

Action Creator

type **ActionCreator** = (**ANY**) => **Action | AsyncAction**

Async Action

type **AsyncAction** = **any**

Middleware

type **MiddlewareAPI** = { **DISPATCH: DISPATCH**, **GETSTATE: () => STATE** }

type **Middleware** = (**MIDDLEWAREAPI**) => (**DISPATCH**) => **Dispatch**

Store

type **Store** =

```
{
  dispatch( ACTION | ASYNCACTION ) => any,
  getState() => State,
  subscribe( () => VOID ) => () => void,
  replaceReducer( REDUCER ) => void
}
```

Store Creator

type **StoreCreator** = (**REDUCER**, **?INITIALSTATE**, **?ENHANCER**) => **Store**

Store Enhancer

type **StoreEnhancer** = (**STORECREATOR**) => **StoreCreator**