**prop-types**

Runtime type checking for React props and similar objects.

You can use prop-types to document the intended types of properties passed to components. React (and potentially other libraries—see the checkPropTypes() reference below) will check props passed to your components against those definitions, and warn in development if they don’t match.

**Installation**

npm install --save prop-types

**Importing**

**import** PropTypes **from** 'prop-types'; *// ES6*

var PropTypes **=** require('prop-types'); *// ES5 with npm*

**CDN**

If you prefer to exclude prop-types from your application and use it globally via window.PropTypes, the prop-types package provides single-file distributions, which are hosted on the following CDNs:

* [**unpkg**](https://unpkg.com/prop-types/)

*<!-- development version -->*

<script **src**="https://unpkg.com/prop-types@15.6/prop-types.js"></script>

*<!-- production version -->*

<script **src**="https://unpkg.com/prop-types@15.6/prop-types.min.js"></script>

* [**cdnjs**](https://cdnjs.com/libraries/prop-types)

*<!-- development version -->*

<script **src**="https://cdnjs.cloudflare.com/ajax/libs/prop-types/15.6.0/prop-types.js"></script>

*<!-- production version -->*

<script **src**="https://cdnjs.cloudflare.com/ajax/libs/prop-types/15.6.0/prop-types.min.js"></script>

To load a specific version of prop-types replace 15.6.0 with the version number.

**Usage**

PropTypes was originally exposed as part of the React core module, and is commonly used with React components. Here is an example of using PropTypes with a React component, which also documents the different validators provided:

**import** React **from** 'react';

**import** PropTypes **from** 'prop-types';

class MyComponent extends React.Component {

  render() {

*// ... do things with the props*

  }

}

MyComponent.propTypes **=** {

*// You can declare that a prop is a specific JS primitive. By default, these*

*// are all optional.*

  optionalArray**:** PropTypes.array,

  optionalBool**:** PropTypes.bool,

  optionalFunc**:** PropTypes.func,

  optionalNumber**:** PropTypes.number,

  optionalObject**:** PropTypes.object,

  optionalString**:** PropTypes.string,

  optionalSymbol**:** PropTypes.symbol,

*// Anything that can be rendered: numbers, strings, elements or an array*

*// (or fragment) containing these types.*

  optionalNode**:** PropTypes.node,

*// A React element (ie. <MyComponent />).*

  optionalElement**:** PropTypes.element,

*// A React element type (ie. MyComponent).*

  optionalElementType**:** PropTypes.elementType,

*// You can also declare that a prop is an instance of a class. This uses*

*// JS's instanceof operator.*

  optionalMessage**:** PropTypes.instanceOf(Message),

*// You can ensure that your prop is limited to specific values by treating*

*// it as an enum.*

  optionalEnum**:** PropTypes.oneOf(['News', 'Photos']),

*// An object that could be one of many types*

  optionalUnion**:** PropTypes.oneOfType([

    PropTypes.string,

    PropTypes.number,

    PropTypes.instanceOf(Message)

  ]),

*// An array of a certain type*

  optionalArrayOf**:** PropTypes.arrayOf(PropTypes.number),

*// An object with property values of a certain type*

  optionalObjectOf**:** PropTypes.objectOf(PropTypes.number),

*// You can chain any of the above with `isRequired` to make sure a warning*

*// is shown if the prop isn't provided.*

*// An object taking on a particular shape*

  optionalObjectWithShape**:** PropTypes.shape({

    optionalProperty**:** PropTypes.string,

    requiredProperty**:** PropTypes.number.isRequired

  }),

*// An object with warnings on extra properties*

  optionalObjectWithStrictShape**:** PropTypes.exact({

    optionalProperty**:** PropTypes.string,

    requiredProperty**:** PropTypes.number.isRequired

  }),

  requiredFunc**:** PropTypes.func.isRequired,

*// A value of any data type*

  requiredAny**:** PropTypes.any.isRequired,

*// You can also specify a custom validator. It should return an Error*

*// object if the validation fails. Don't `console.warn` or throw, as this*

*// won't work inside `oneOfType`.*

  customProp**:** function(props, propName, componentName) {

**if** (**!**/matchme/.test(props[propName])) {

**return** **new** Error(

        'Invalid prop `' **+** propName **+** '` supplied to' **+**

        ' `' **+** componentName **+** '`. Validation failed.'

      );

    }

  },

*// You can also supply a custom validator to `arrayOf` and `objectOf`.*

*// It should return an Error object if the validation fails. The validator*

*// will be called for each key in the array or object. The first two*

*// arguments of the validator are the array or object itself, and the*

*// current item's key.*

  customArrayProp**:** PropTypes.arrayOf(function(propValue, key, componentName, location, propFullName) {

**if** (**!**/matchme/.test(propValue[key])) {

**return** **new** Error(

        'Invalid prop `' **+** propFullName **+** '` supplied to' **+**

        ' `' **+** componentName **+** '`. Validation failed.'

      );

    }

  })

};

Refer to the [**React documentation**](https://facebook.github.io/react/docs/typechecking-with-proptypes.html) for more information.

**Migrating from React.PropTypes**

Check out [**Migrating from React.PropTypes**](https://facebook.github.io/react/blog/2017/04/07/react-v15.5.0.html#migrating-from-react.proptypes) for details on how to migrate to prop-types from React.PropTypes.

Note that this blog posts **mentions a codemod script that performs the conversion automatically**.

There are also important notes below.

**How to Depend on This Package?**

For apps, we recommend putting it in dependencies with a caret range. For example:

  "dependencies"**:** {

    "prop-types"**:** "^15.5.7"

  }

For libraries, we *also* recommend leaving it in dependencies:

  "dependencies"**:** {

    "prop-types"**:** "^15.5.7"

  },

  "peerDependencies"**:** {

    "react"**:** "^15.5.0"

  }

**Note:** there are known issues in versions before 15.5.7 so we recommend using it as the minimal version.

Make sure that the version range uses a caret (^) and thus is broad enough for npm to efficiently deduplicate packages.

For UMD bundles of your components, make sure you **don’t** include PropTypes in the build. Usually this is done by marking it as an external (the specifics depend on your bundler), just like you do with React.

**Compatibility**

**React 0.14**

This package is compatible with **React 0.14.9**. Compared to 0.14.8 (which was released in March of 2016), there are no other changes in 0.14.9, so it should be a painless upgrade.

*# ATTENTION: Only run this if you still use React 0.14!*

npm install --save react@^0.14.9 react-dom@^0.14.9

**React 15+**

This package is compatible with **React 15.3.0** and higher.

npm install --save react@^15.3.0 react-dom@^15.3.0

**What happens on other React versions?**

It outputs warnings with the message below even though the developer doesn’t do anything wrong. Unfortunately there is no solution for this other than updating React to either 15.3.0 or higher, or 0.14.9 if you’re using React 0.14.

**Difference from React.PropTypes: Don’t Call Validator Functions**

First of all, **which version of React are you using**? You might be seeing this message because a component library has updated to use prop-types package, but your version of React is incompatible with it. See the [**above section**](https://www.npmjs.com/package/prop-types#compatibility) for more details.

Are you using either React 0.14.9 or a version higher than React 15.3.0? Read on.

When you migrate components to use the standalone prop-types, **all validator functions will start throwing an error if you call them directly**. This makes sure that nobody relies on them in production code, and it is safe to strip their implementations to optimize the bundle size.

Code like this is still fine:

MyComponent.propTypes **=** {

  myProp**:** PropTypes.bool

};

However, code like this will not work with the prop-types package:

*// Will not work with `prop-types` package!*

var errorOrNull **=** PropTypes.bool(42, 'myProp', 'MyComponent', 'prop');

It will throw an error:

Calling PropTypes validators directly is not supported by the `prop-types` package.

Use PropTypes.checkPropTypes() to call them.

(If you see **a warning** rather than an error with this message, please check the [**above section about compatibility**](https://www.npmjs.com/package/prop-types#compatibility).)

This is new behavior, and you will only encounter it when you migrate from React.PropTypes to the prop-types package. For the vast majority of components, this doesn’t matter, and if you didn’t see [**this warning**](https://facebook.github.io/react/warnings/dont-call-proptypes.html) in your components, your code is safe to migrate. This is not a breaking change in React because you are only opting into this change for a component by explicitly changing your imports to use prop-types. If you temporarily need the old behavior, you can keep using React.PropTypes until React 16.

**If you absolutely need to trigger the validation manually**, call PropTypes.checkPropTypes(). Unlike the validators themselves, this function is safe to call in production, as it will be replaced by an empty function:

*// Works with standalone PropTypes*

PropTypes.checkPropTypes(MyComponent.propTypes, props, 'prop', 'MyComponent');

See below for more info.

**You might also see this error** if you’re calling a PropTypes validator from your own custom PropTypes validator. In this case, the fix is to make sure that you are passing *all* of the arguments to the inner function. There is a more in-depth explanation of how to fix it [**on this page**](https://facebook.github.io/react/warnings/dont-call-proptypes.html#fixing-the-false-positive-in-third-party-proptypes). Alternatively, you can temporarily keep using React.PropTypes until React 16, as it would still only warn in this case.

If you use a bundler like Browserify or Webpack, don’t forget to [**follow these instructions**](https://reactjs.org/docs/optimizing-performance.html#use-the-production-build) to correctly bundle your application in development or production mode. Otherwise you’ll ship unnecessary code to your users.

**PropTypes.checkPropTypes**

React will automatically check the propTypes you set on the component, but if you are using PropTypes without React then you may want to manually call PropTypes.checkPropTypes, like so:

const myPropTypes **=** {

  name**:** PropTypes.string,

  age**:** PropTypes.number,

*// ... define your prop validations*

};

const props **=** {

  name**:** 'hello', *// is valid*

  age**:** 'world', *// not valid*

};

*// Let's say your component is called 'MyComponent'*

*// Works with standalone PropTypes*

PropTypes.checkPropTypes(myPropTypes, props, 'age', 'MyComponent');

*// This will warn as follows:*

*// Warning: Failed prop type: Invalid prop `age` of type `string` supplied to*

*// `MyComponent`, expected `number`.*

**PropTypes.resetWarningCache()**

PropTypes.checkPropTypes(...) only console.error.log(...)s a given message once. To reset the cache while testing call PropTypes.resetWarningCache()

**License**