react-dev-utils

This package includes some utilities used by Create React App. Please refer to its documentation:

- Getting Started – How to create a new app.
- User Guide – How to develop apps bootstrapped with Create React App.

Usage in Create React App Projects

These utilities come by default with Create React App. **You don't need to install it separately in Create React App projects.**

Usage Outside of Create React App

If you don't use Create React App, or if you ejected, you may keep using these utilities. Their development will be aligned with Create React App, so major versions of these utilities may come out relatively often. Feel free to fork or copy and paste them into your projects if you'd like to have more control over them, or feel free to use the old versions. Not all of them are React-specific, but we might make some of them more React-specific in the future.

Entry Points

There is no single entry point. You can only import individual top-level modules.

`new InterpolateHtmlPlugin(htmlWebpackPlugin: HtmlWebpackPlugin, replacements: {[key:string]: string})` This webpack plugin lets us interpolate custom variables into `index.html`. It works in tandem with HtmlWebpackPlugin 2.x via its events.

```
var path = require('path');
var HtmlWebpackPlugin = require('html-webpack-plugin');
var InterpolateHtmlPlugin = require('react-dev-utils/InterpolateHtmlPlugin');

// webpack config
var publicUrl = '/my-custom-url';

module.exports = {
  output: {
    // ...
    publicPath: publicUrl + '/',
  },
  // ...
  plugins: [
    // Generates an `index.html` file with the <script> injected.
    new HtmlWebpackPlugin({
      inject: true,
      template: path.resolve('public/index.html'),
    }),
    // Makes the public URL available as %PUBLIC_URL% in index.html, e.g.:
    // <link rel="icon" href="%PUBLIC_URL%/favicon.ico">
    new InterpolateHtmlPlugin(HtmlWebpackPlugin, {
      PUBLIC_URL: publicUrl,
      // You can pass any key-value pairs, this was just an example.
      // WHATEVER: 42 will replace %WHATEVER% with 42 in index.html.
    }),
    // ...
  ],
  // ...
};
```

`new InlineChunkHtmlPlugin(htmlWebpackPlugin: HtmlWebpackPlugin, tests: Regex[])` This webpack plugin inlines script chunks into `index.html`. It works in tandem with HtmlWebpackPlugin 4.x.

```
var path = require('path');
var HtmlWebpackPlugin = require('html-webpack-plugin');
var InlineChunkHtmlPlugin = require('react-dev-utils/InlineChunkHtmlPlugin');

// webpack config
var publicUrl = '/my-custom-url';

module.exports = {
  output: {
    // ...
    publicPath: publicUrl + '/',
  },
  // ...
  plugins: [
    // Generates an `index.html` file with the <script> injected.
    new HtmlWebpackPlugin({
      inject: true,
      template: path.resolve('public/index.html'),
    }),
    // Inlines chunks with `runtime` in the name
    new InlineChunkHtmlPlugin(HtmlWebpackPlugin, [/runtime/]),
    // ...
  ],
  // ...
};
```

`new ModuleScopePlugin(appSrc: string | string[], allowedFiles?: string[])` This webpack plugin ensures that relative imports from app's source directories don't reach outside of it.

```
var path = require('path');
var ModuleScopePlugin = require('react-dev-utils/ModuleScopePlugin');

module.exports = {
  // ...
  resolve: {
    // ...
    plugins: [
      new ModuleScopePlugin(paths.appSrc, [paths.appPackageJson]),
      // ...
    ],
    // ...
  },
  // ...
};
```

`checkRequiredFiles(files: Array<string>): boolean` Makes sure that all passed files exist. Filenames are expected to be absolute. If a file is not found, prints a warning message and returns `false`.

```
var path = require('path');
var checkRequiredFiles = require('react-dev-utils/checkRequiredFiles');

if (
  !checkRequiredFiles([
    path.resolve('public/index.html'),
    path.resolve('src/index.js'),
  ])
) {
  process.exit(1);
}
```

`clearConsole(): void` Clears the console, hopefully in a cross-platform way.

```
var clearConsole = require('react-dev-utils/clearConsole');

clearConsole();
console.log('Just cleared the screen!');
```

`eslintFormatter(results: Object): string` This is our custom ESLint formatter that integrates well with Create React App console output. You can use the default one instead if you prefer so.

```
const eslintFormatter = require('react-dev-utils/eslintFormatter');

// In your webpack config:
// ...
module: {
  rules: [
    {
      test: /\.(js|jsx)$/,
      include: paths.appSrc,
      enforce: 'pre',
      use: [
        {
          loader: 'eslint-loader',
          options: {
            // Pass the formatter:
            formatter: eslintFormatter,
          },
        },
      ],
    },
  ];
}
```

`FileSizeReporter`

`measureFileSizesBeforeBuild(buildFolder: string): Promise<OpaqueFileSizes>`
Captures JS and CSS asset sizes inside the passed `buildFolder`. Save the result value to compare it after the build.

`printFileSizesAfterBuild(webpackStats: WebpackStats, previousFileSizes: OpaqueFileSizes, buildFolder: string, maxBundleGzipSize?: number, maxChunkGzipSize?: number)` Prints the JS and CSS asset sizes after the build, and includes a size comparison with `previousFileSizes` that were captured earlier using `measureFileSizesBeforeBuild()`. `maxBundleGzipSize` and `maxChunkGzipSizemay` may optionally be specified to display a warning when the main bundle or a chunk exceeds the specified size (in bytes).

```
var {
  measureFileSizesBeforeBuild,
  printFileSizesAfterBuild,
} = require('react-dev-utils/FileSizeReporter');

measureFileSizesBeforeBuild(buildFolder).then(previousFileSizes => {
  return cleanAndRebuild().then(webpackStats => {
    printFileSizesAfterBuild(webpackStats, previousFileSizes, buildFolder);
  });
});
```

`formatWebpackMessages({errors: Array<string>, warnings: Array<string>}): {errors: Array<string>, warnings: Array<string>}` Extracts and prettifies warning and error messages from webpack stats object.

```
var webpack = require('webpack');
var config = require('../config/webpack.config.dev');
var formatWebpackMessages = require('react-dev-utils/formatWebpackMessages');

var compiler = webpack(config);

compiler.hooks.invalid.tap('invalid', function () {
  console.log('Compiling...');
});

compiler.hooks.done.tap('done', function (stats) {
  var rawMessages = stats.toJson({}, true);
  var messages = formatWebpackMessages(rawMessages);
  if (!messages.errors.length && !messages.warnings.length) {
    console.log('Compiled successfully!');
  }
  if (messages.errors.length) {
    console.log('Failed to compile.');
    messages.errors.forEach(e => console.log(e));
    return;
  }
  if (messages.warnings.length) {
    console.log('Compiled with warnings.');
    messages.warnings.forEach(w => console.log(w));
  }
});
```

`printBuildError(error: Object): void` Prettify some known build errors. Pass an Error object to log a prettified error message in the console.

```
  const printBuildError = require('react-dev-utils/printBuildError')
  try {
    build()
  } catch(e) {
    printBuildError(e) // logs prettified message
  }
```

`getProcessForPort(port: number): string` Finds the currently running process on `port`. Returns a string containing the name and directory, e.g.,

```
create-react-app
in /Users/developer/create-react-app
```

```
var getProcessForPort = require('react-dev-utils/getProcessForPort');

getProcessForPort(3000);
```

`launchEditor(fileName: string, lineNumber: number): void` On macOS, tries to find a known running editor process and opens the file in it. It can also be explicitly configured by `REACT_EDITOR`, `VISUAL`, or `EDITOR` environment variables. For example, you can put `REACT_EDITOR=atom` in your `.env.local` file, and Create React App will respect that.

`noopServiceWorkerMiddleware(servedPath: string): ExpressMiddleware` Returns Express middleware that serves a `${servedPath}/service-worker.js` that resets any previously set service worker configuration. Useful for development.

`redirectServedPathMiddleware(servedPath: string): ExpressMiddleware` Returns Express middleware that redirects to `${servedPath}/${req.path}`, if `req.url` does not start with `servedPath`. Useful for development.

`openBrowser(url: string): boolean` Attempts to open the browser with a given URL. On Mac OS X, attempts to reuse an existing Chrome tab via AppleScript. Otherwise, falls back to opn behavior.

```
var path = require('path');
var openBrowser = require('react-dev-utils/openBrowser');

if (openBrowser('http://localhost:3000')) {
  console.log('The browser tab has been opened!');
}
```

`printHostingInstructions(appPackage: Object, publicUrl: string, publicPath: string, buildFolder: string, useYarn: boolean): void` Prints hosting instructions after the project is built.

Pass your parsed `package.json` object as `appPackage`, your URL where you plan to host the app as `publicUrl`, `output.publicPath` from your webpack configuration as `publicPath`, the `buildFolder` name, and whether to `useYarn` in instructions.

```
const appPackage = require(paths.appPackageJson);
const publicUrl = paths.publicUrlOrPath;
const publicPath = config.output.publicPath;
printHostingInstructions(appPackage, publicUrl, publicPath, 'build', true);
```

`WebpackDevServerUtils`

`choosePort(host: string, defaultPort: number): Promise<number | null>` Returns a Promise resolving to either `defaultPort` or next available port if the user confirms it is okay to do. If the port is taken and the user has refused to use another port, or if the terminal is not interactive and can't present user with the choice, resolves to `null`.

`createCompiler(args: Object): WebpackCompiler` Creates a webpack compiler instance for WebpackDevServer with built-in helpful messages.

The `args` object accepts a number of properties:

- **appName** `string`: The name that will be printed to the terminal.
- **config** `Object`: The webpack configuration options to be provided to the webpack constructor.
- **urls** `Object`: To provide the `urls` argument, use `prepareUrls()` described below.
- **useYarn** `boolean`: If `true`, yarn instructions will be emitted in the terminal instead of npm.
- **useTypeScript** `boolean`: If `true`, TypeScript type checking will be enabled. Be sure to provide the `devSocket` argument above if this is set to `true`.
- **webpack** `function`: A reference to the webpack constructor.

`prepareProxy(proxySetting: string, appPublicFolder: string, servedPathname: string): Object` Creates a WebpackDevServer `proxy` configuration object from the `proxy` setting in `package.json`.

`prepareUrls(protocol: string, host: string, port: number, pathname: string = '/'): Object` Returns an object with local and remote URLs for the development server. Pass this object to `createCompiler()` described above.

`webpackHotDevClient` This is an alternative client for WebpackDevServer that shows a syntax error overlay.

It currently supports only webpack 3.x.

```
// webpack development config
module.exports = {
  // ...
  entry: [
    // You can replace the line below with these two lines if you prefer the
    // stock client:
    // require.resolve('webpack-dev-server/client') + '?/',
    // require.resolve('webpack/hot/dev-server'),
    'react-dev-utils/webpackHotDevClient',
    'src/index',
  ],
  // ...
};
```

`getCSSModuleLocalIdent(context: Object, localIdentName: String, localName: String, options: Object): string` Creates a class name for CSS Modules that uses either the filename or folder name if named `index.module.css`.

For `MyFolder/MyComponent.module.css` and class `MyClass` the output will be `MyComponent.module_MyClass__[hash]` For `MyFolder/index.module.css` and class `MyClass` the output will be `MyFolder_MyClass__[hash]`

```
const getCSSModuleLocalIdent = require('react-dev-utils/getCSSModuleLocalIdent');

// In your webpack config:
// ...
module: {
  rules: [
    {
      test: /\.module\.css$/,
      use: [
        require.resolve('style-loader'),
        {
          loader: require.resolve('css-loader'),
          options: {
            importLoaders: 1,
            modules: {
              getLocalIdent: getCSSModuleLocalIdent,
            },
          },
        },
        {
          loader: require.resolve('postcss-loader'),
          options: postCSSLoaderOptions,
        },
      ],
    },
  ];
}
```

`getCacheIdentifier(environment: string, packages: string[]): string` Returns a cache identifier (string) consisting of the specified environment and related package versions, e.g.,

```
var getCacheIdentifier = require('react-dev-utils/getCacheIdentifier');

getCacheIdentifier('prod', ['react-dev-utils', 'chalk']); // # => 'prod:react-dev-utils@5.0
```