
r.js

A command line tool for running JavaScript scripts that use the Asynchronous Module Definition API (AMD) for declaring and using JavaScript modules and regular JavaScript script files.

It is part of the RequireJS project, and works with the RequireJS implementation of AMD.

r.js is a single script that has two major functions:

- Run AMD-based projects in Node and Nashorn, Rhino and xpcshell.
- Includes the RequireJS Optimizer that combines scripts for optimal browser delivery.

Installation

Node

```
1 npm install -g requirejs
```

From then on, you can use `r.js` on the command line to run the optimizer.

Nashorn/Rhino/Browser

Download the latest release from the RequireJS download page.

xpcshell

xpcshell support was added in r.js version 2.1.5, so use that r.js version or later.

Download the latest release of r.js from the RequireJS download page.

From this repo

r.js is made up of a series of modules that are built into one file for distribution. The **dist** directory contains the built version of the code. In the master branch, it should match the current state of the master code.

If you are doing local modifications from a clone of this repo, you can run the following command to generate an r.js at the root of this repo:

```
1 node dist.js
```

To generate an `r.js` that also gets copied to **dist** with a time stamp, run:

```
1 ./copydist.js
```

Running AMD-based projects

If your JS project's main application file is called `main.js`, then do the following:

Node

```
1 r.js main.js
```

Requires Node 0.4 or later.

`r.js` allows using Node modules installed via `npm`. For more info see the [Use with Node docs](#).

Java

Nashorn

As of `r.js` 2.1.16, `r.js` can run in Nashorn, Java 8+'s JavaScript engine, via the `jjs` command line tool that is installed with Java.

Then general format of the command:

```
1 jjs -scripting path/to/r.js -- [r.js command line arguments here]
```

Examples:

```
1 # Calling r.js to optimize a project using the build config in build.js
2 jjs -scripting path/to/r.js -- -o build.js
3
4 # Calling r.js to run AMD modules, where the main app program is main.
   js
5 jjs -scripting path/to/r.js -- main.js
```

All further examples will use the Node notation, but substitute the **`r.js`** references below with the command line structure mentioned above (`jjs -scripting path/to/r.js --`).

Rhino

Using Rhino requires some JAR files in the CLASSPATH for it to work:

-
- rhino.jar from the Rhino project.
 - compiler.jar if you are using the optimizer and want to use Closure Compiler.

Download those files to your machine. To run r.js, you can use this type of command:

OS X/Linux/Unix:

```
1 java -classpath path/to/rhino/js.jar:path/to/closure/compiler.jar org.mozilla.javascript.tools.shell.Main -opt -1 r.js main.js
```

Windows

```
1 java -classpath path/to/rhino/js.jar;path/to/closure/compiler.jar org.mozilla.javascript.tools.shell.Main -opt -1 r.js main.js
```

s If you want to run it in the debugger, replace `org.mozilla.javascript.tools.shell.Main` with **`org.mozilla.javascript.tools.debugger.Main`**.

All further examples will use the Node notation, but substitute the **r.js** references below with the appropriate java command.

xpcshell

To run the optimizer using a build config file or command line build options:

```
1 path/to/xpcshell path/to/r.js -o buildconfig.js
```

r.js can also be used as a library in another .js file run via xpcshell.

- tests/xpcshell/run.js: shows how to load AMD modules by using r.js as a library.
- tests/xpcshell/build.js: shows how to trigger the optimizer from within another .js file.

Optimizer

The optimizer can be run by passing the **-o** command to r.js:

```
1 r.js -o path/to/buildconfig.js
```

See the Optimization doc for more information on the optimizer.

If running in **Java**, be sure to grab the Rhino and Closure Compiler jar files in the lib/ directory, then run this command:

OS X/Linux/Unix:

```
1 java -classpath path/to/rhino/js.jar:path/to/closure/compiler.jar org.mozilla.javascript.tools.shell.Main r.js -o path/to/buildconfig.js
```

Windows

```
1 java -classpath path/to/rhino/js.jar;path/to/closure/compiler.jar org.mozilla.javascript.tools.shell.Main r.js -o path/to/buildconfig.js
```

What makes it special

The optimizer is better than using a plain concatenation script because it runs `require.js` as part of the optimization, so it knows how to:

- Use Loader Plugins to load non-script dependencies and inline them in built files.
- Name anonymous modules. If your optimization step does not do this, and you use anonymous modules, you will get errors running the built code.

Other r.js commands

Get Version

To get the version of `r.js` and the version of `require.js` used by `r.js`:

```
1 r.js -v
```

Convert CommonJS modules

To convert a directory of CommonJS modules to ones that have `define()` wrappers:

```
1 r.js -convert path/to/commonjs/dir output/dir
```

Most, but not all, CommonJS modules can be converted to `define()`-wrapped modules and still work.

However, there are some modules that may fail if:

- They use code branches like `if/else` or `try/catch` to call `require()`. There are problems supporting this kind of dynamic module calls in an async environment.

-
- Some kinds of circular dependencies will not work right. The kinds that fail are normally very brittle and depend on the execution order of the dependent modules.

License

MIT

Code of Conduct

jQuery Foundation Code of Conduct.

Directory layout

Directory prerequisites

r.js assumes that there are some other projects checked out as sibling directories to it, and named certain names, in order for the tests to pass.

So it is best to create the following directory structure with the following git clone commands:

```
1 mkdir requirejs
2 cd requirejs
3 git clone git://github.com/requirejs/r.js.git
4 git clone git://github.com/requirejs/requirejs.git
5 git clone git://github.com/requirejs/text.git
```

So there should be a sibling `requirejs` and `text` directories to the `r.js` directory containing your clone of the r.js project.

Directory details

The r.js project has the following directory layout:

- **dist.js**: the script that builds r.js
- **require.js**: the version of require.js to include in r.js
- **dist** the directory containing releases of r.js
- **build**: The files that make up the optimizer. dist.js includes a list of the files from this directory to build into r.js.

-
- **lib:** The Java libraries for Rhino and Closure Compiler. Only needed if using Java/Rhino to run r.js
 - **tests:** command line tests. Run it under Node and Rhino by doing `../r.js all.js`

dist.js takes the build/jslib/x.js file and injects the require.js files and other files from the build/jslib directory into it.

If you make changes to any of those files, you will need to run **node dist.js** to generate a new r.js. Be sure to run it through the tests , using both Node and Java/Rhino:

```
1 * node dist.js
2 * cd tests
3 * node ../r.js all.js
4 * java -classpath ../lib/rhino/js.jar:../lib/closure/compiler.jar org.
   mozilla.javascript.tools.shell.Main ../r.js all.js
5 * cd ../build/tests
6 * node ../../r.js all.js
7 * java -classpath ../../lib/rhino/js.jar:../../lib/closure/compiler.jar
   org.mozilla.javascript.tools.shell.Main ../../r.js all.js
```

For running tests, put xpcshell in env/xpcshell/ as a directory, that contains all the files needed for it to run, including the xpcshell binary. Downloading a xulrunner nightly might work.

Contributing code changes

See the RequireJS Contributing page for info on how to contribute code/bug fixes to this project.

Use GitHub pull requests to point to code changes, although for larger changes, contact the requirejs mailing list to discuss them first.

Included libraries

r.js includes modules from these projects:

- Esprima
- UglifyJS

Doing a release

To do a release of version 0.0.0:

-
- Make sure the right version of require.js is in the project.
 - Modify build/jslib/x.js to update the r.js version number in two places.
 - node dist.js
 - Run the tests (see above). They should pass. :)
 - mv r.js dist/r.js
 - git commit -am "Release 0.0.0"
 - git tag -am "Release 0.0.0" 0.0.0
 - git push origin master
 - git push -tags

Update the RequireJS download site to point to the latest release, then update the requirejs/requirejs-npm repo to have the latest changes and publish the result to npm.

Make sure to keep `#!/usr/bin/env node` as the first line in bin/r.js in the requirejs-npm repo.