# resolve [![Version Badge][2]][1]

implements the [node require.resolve() algorithm](https://nodejs.org/api/modules.html#modules_all_together) such that you can require.resolve() on behalf of a file asynchronously and synchronously

[github actions](https://github.com/browserify/resolve/actions) [coverage](https://app.codecov.io/gh/browserify/resolve/) [dependency status](https://david-dm.org/browserify/resolve) [dev dependency status](https://david-dm.org/browserify/resolve#info=devDependencies) [License](http://license) [Downloads](https://npm-stat.com/charts.html?package=resolve)

[npm badge](https://npmjs.org/package/resolve)

# example

asynchronously resolve:

var resolve = require('resolve/async'); // or, require('resolve')

resolve('tap', { basedir: \_\_dirname }, function (err, res) {

if (err) console.error(err);

else console.log(res);

});

$ node example/async.js

/home/substack/projects/node-resolve/node\_modules/tap/lib/main.js

synchronously resolve:

var resolve = require('resolve/sync'); // or, `require('resolve').sync

var res = resolve('tap', { basedir: \_\_dirname });

console.log(res);

$ node example/sync.js

/home/substack/projects/node-resolve/node\_modules/tap/lib/main.js

# methods

var resolve = require('resolve');

var async = require('resolve/async');

var sync = require('resolve/sync');

For both the synchronous and asynchronous methods, errors may have any of the following err.code values:

* MODULE\_NOT\_FOUND: the given path string (id) could not be resolved to a module
* INVALID\_BASEDIR: the specified opts.basedir doesn't exist, or is not a directory
* INVALID\_PACKAGE\_MAIN: a package.json was encountered with an invalid main property (eg. not a string)

## resolve(id, opts={}, cb)

Asynchronously resolve the module path string id into cb(err, res [, pkg]), where pkg (if defined) is the data from package.json.

options are:

* opts.basedir - directory to begin resolving from
* opts.package - package.json data applicable to the module being loaded
* opts.extensions - array of file extensions to search in order
* opts.includeCoreModules - set to false to exclude node core modules (e.g. fs) from the search
* opts.readFile - how to read files asynchronously
* opts.isFile - function to asynchronously test whether a file exists
* opts.isDirectory - function to asynchronously test whether a file exists and is a directory
* opts.realpath - function to asynchronously resolve a potential symlink to its real path
* opts.readPackage(readFile, pkgfile, cb) - function to asynchronously read and parse a package.json file
  + readFile - the passed opts.readFile or fs.readFile if not specified
  + pkgfile - path to package.json
  + cb - callback
* opts.packageFilter(pkg, pkgfile, dir) - transform the parsed package.json contents before looking at the "main" field
  + pkg - package data
  + pkgfile - path to package.json
  + dir - directory that contains package.json
* opts.pathFilter(pkg, path, relativePath) - transform a path within a package
  + pkg - package data
  + path - the path being resolved
  + relativePath - the path relative from the package.json location
  + returns - a relative path that will be joined from the package.json location
* opts.paths - require.paths array to use if nothing is found on the normal node\_modules recursive walk (probably don't use this)

For advanced users, paths can also be a opts.paths(request, start, opts) function

* + request - the import specifier being resolved
  + start - lookup path
  + getNodeModulesDirs - a thunk (no-argument function) that returns the paths using standard node\_modules resolution
  + opts - the resolution options
* opts.packageIterator(request, start, opts) - return the list of candidate paths where the packages sources may be found (probably don't use this)
  + request - the import specifier being resolved
  + start - lookup path
  + getPackageCandidates - a thunk (no-argument function) that returns the paths using standard node\_modules resolution
  + opts - the resolution options
* opts.moduleDirectory - directory (or directories) in which to recursively look for modules. default: "node\_modules"
* opts.preserveSymlinks - if true, doesn't resolve basedir to real path before resolving. This is the way Node resolves dependencies when executed with the [--preserve-symlinks](https://nodejs.org/api/all.html#cli_preserve_symlinks) flag. **Note:** this property is currently true by default but it will be changed to false in the next major version because *Node's resolution algorithm does not preserve symlinks by default*.

default opts values:

{

paths: [],

basedir: \_\_dirname,

extensions: ['.js'],

includeCoreModules: true,

readFile: fs.readFile,

isFile: function isFile(file, cb) {

fs.stat(file, function (err, stat) {

if (!err) {

return cb(null, stat.isFile() || stat.isFIFO());

}

if (err.code === 'ENOENT' || err.code === 'ENOTDIR') return cb(null, false);

return cb(err);

});

},

isDirectory: function isDirectory(dir, cb) {

fs.stat(dir, function (err, stat) {

if (!err) {

return cb(null, stat.isDirectory());

}

if (err.code === 'ENOENT' || err.code === 'ENOTDIR') return cb(null, false);

return cb(err);

});

},

realpath: function realpath(file, cb) {

var realpath = typeof fs.realpath.native === 'function' ? fs.realpath.native : fs.realpath;

realpath(file, function (realPathErr, realPath) {

if (realPathErr && realPathErr.code !== 'ENOENT') cb(realPathErr);

else cb(null, realPathErr ? file : realPath);

});

},

readPackage: function defaultReadPackage(readFile, pkgfile, cb) {

readFile(pkgfile, function (readFileErr, body) {

if (readFileErr) cb(readFileErr);

else {

try {

var pkg = JSON.parse(body);

cb(null, pkg);

} catch (jsonErr) {

cb(null);

}

}

});

},

moduleDirectory: 'node\_modules',

preserveSymlinks: true

}

## resolve.sync(id, opts)

Synchronously resolve the module path string id, returning the result and throwing an error when id can't be resolved.

options are:

* opts.basedir - directory to begin resolving from
* opts.extensions - array of file extensions to search in order
* opts.includeCoreModules - set to false to exclude node core modules (e.g. fs) from the search
* opts.readFileSync - how to read files synchronously
* opts.isFile - function to synchronously test whether a file exists
* opts.isDirectory - function to synchronously test whether a file exists and is a directory
* opts.realpathSync - function to synchronously resolve a potential symlink to its real path
* opts.readPackageSync(readFileSync, pkgfile) - function to synchronously read and parse a package.json file
  + readFileSync - the passed opts.readFileSync or fs.readFileSync if not specified
  + pkgfile - path to package.json
* opts.packageFilter(pkg, dir) - transform the parsed package.json contents before looking at the "main" field
  + pkg - package data
  + dir - directory that contains package.json (Note: the second argument will change to "pkgfile" in v2)
* opts.pathFilter(pkg, path, relativePath) - transform a path within a package
  + pkg - package data
  + path - the path being resolved
  + relativePath - the path relative from the package.json location
  + returns - a relative path that will be joined from the package.json location
* opts.paths - require.paths array to use if nothing is found on the normal node\_modules recursive walk (probably don't use this)

For advanced users, paths can also be a opts.paths(request, start, opts) function

* + request - the import specifier being resolved
  + start - lookup path
  + getNodeModulesDirs - a thunk (no-argument function) that returns the paths using standard node\_modules resolution
  + opts - the resolution options
* opts.packageIterator(request, start, opts) - return the list of candidate paths where the packages sources may be found (probably don't use this)
  + request - the import specifier being resolved
  + start - lookup path
  + getPackageCandidates - a thunk (no-argument function) that returns the paths using standard node\_modules resolution
  + opts - the resolution options
* opts.moduleDirectory - directory (or directories) in which to recursively look for modules. default: "node\_modules"
* opts.preserveSymlinks - if true, doesn't resolve basedir to real path before resolving. This is the way Node resolves dependencies when executed with the [--preserve-symlinks](https://nodejs.org/api/all.html#cli_preserve_symlinks) flag. **Note:** this property is currently true by default but it will be changed to false in the next major version because *Node's resolution algorithm does not preserve symlinks by default*.

default opts values:

{

paths: [],

basedir: \_\_dirname,

extensions: ['.js'],

includeCoreModules: true,

readFileSync: fs.readFileSync,

isFile: function isFile(file) {

try {

var stat = fs.statSync(file);

} catch (e) {

if (e && (e.code === 'ENOENT' || e.code === 'ENOTDIR')) return false;

throw e;

}

return stat.isFile() || stat.isFIFO();

},

isDirectory: function isDirectory(dir) {

try {

var stat = fs.statSync(dir);

} catch (e) {

if (e && (e.code === 'ENOENT' || e.code === 'ENOTDIR')) return false;

throw e;

}

return stat.isDirectory();

},

realpathSync: function realpathSync(file) {

try {

var realpath = typeof fs.realpathSync.native === 'function' ? fs.realpathSync.native : fs.realpathSync;

return realpath(file);

} catch (realPathErr) {

if (realPathErr.code !== 'ENOENT') {

throw realPathErr;

}

}

return file;

},

readPackageSync: function defaultReadPackageSync(readFileSync, pkgfile) {

var body = readFileSync(pkgfile);

try {

var pkg = JSON.parse(body);

return pkg;

} catch (jsonErr) {}

},

moduleDirectory: 'node\_modules',

preserveSymlinks: true

}

# install

With [npm](https://npmjs.org) do:

npm install resolve

# license

MIT