libnpmaccess

npm v6.0.4 | license ISC | Node CI failing | coverage 100%

`libnpmaccess` is a Node.js library that provides programmatic access to the guts of the npm CLI's `npm access` command and its various subcommands. This includes managing account 2FA, listing packages and permissions, looking at package collaborators, and defining package permissions for users, orgs, and teams.

Example

```
const access = require('libnpmaccess')

// List all packages @zkat has access to on the npm registry.
console.log(Object.keys(await access.lsPackages('zkat')))
```

Table of Contents

- Installing
- Example
- Contributing
- API
  - access opts
  - `public()`
  - `restricted()`
  - `grant()`
  - `revoke()`
  - `tfaRequired()`
  - `tfaNotRequired()`
  - `lsPackages()`
  - `lsPackages.stream()`
  - `lsCollaborators()`
  - `lsCollaborators.stream()`

Install

`$ npm install libnpmaccess`

API

`opts` for `libnpmaccess` commands

`libnpmaccess` uses `npm-registry-fetch`. All options are passed through directly to that library, so please refer to its own `opts` documentation for options that can be passed in.

A couple of options of note for those in a hurry:

- `opts.token` - can be passed in and will be used as the authentication token for the registry. For other ways to pass in auth details, see the n-r-f docs.

- `opts.otp` - certain operations will require an OTP token to be passed in. If a `libnpmaccess` command fails with `err.code === EOTP`, please retry the request with `{otp: <2fa token>}`

`> access.public(spec, [opts]) -> Promise<Boolean>`

`spec` must be an npm-package-arg-compatible registry spec.

Makes package described by `spec` public.

Example

```
await access.public('@foo/bar', {token: 'myregistrytoken'})
// `@foo/bar` is now public
```

`> access.restricted(spec, [opts]) -> Promise<Boolean>`

`spec` must be an npm-package-arg-compatible registry spec.

Makes package described by `spec` private/restricted.

Example

```
await access.restricted('@foo/bar', {token: 'myregistrytoken'})
// `@foo/bar` is now private
```

`> access.grant(spec, team, permissions, [opts]) -> Promise<Boolean>`

`spec` must be an npm-package-arg-compatible registry spec. `team` must be a fully-qualified team name, in the `scope:team` format, with or without the `@` prefix, and the team must be a valid team within that scope. `permissions` must be one of `'read-only'` or `'read-write'`.

Grants `read-only` or `read-write` permissions for a certain package to a team.

Example

```
await access.grant('@foo/bar', '@foo:myteam', 'read-write', {
  token: 'myregistrytoken'
})
// `@foo/bar` is now read/write enabled for the @foo:myteam team.
```

`> access.revoke(spec, team, [opts]) -> Promise<Boolean>`

`spec` must be an npm-package-arg-compatible registry spec. `team` must be a fully-qualified team name, in the `scope:team` format, with or without the `@` prefix, and the team must be a valid team within that scope. `permissions` must be one of `'read-only'` or `'read-write'`.

Removes access to a package from a certain team.

Example

```
await access.revoke('@foo/bar', '@foo:myteam', {
  token: 'myregistrytoken'
})
// @foo:myteam can no longer access `@foo/bar`
```

`> access.tfaRequired(spec, [opts]) -> Promise<Boolean>`

`spec` must be an [`npm-package-arg`]-compatible registry spec.

Makes it so publishing or managing a package requires using 2FA tokens to complete operations.

Example

```
await access.tfaRequires('lodash', {token: 'myregistrytoken'})
// Publishing or changing dist-tags on `lodash` now require OTP to be enabled.
```

`> access.tfaNotRequired(spec, [opts]) -> Promise<Boolean>`

`spec` must be an [`npm-package-arg`]-compatible registry spec.

Disabled the package-level 2FA requirement for `spec`. Note that you will need to pass in an `otp` token in `opts` in order to complete this operation.

Example

```
await access.tfaNotRequired('lodash', {otp: '123654', token: 'myregistrytoken'})
// Publishing or editing dist-tags on `lodash` no longer requires OTP to be
// enabled.
```

`> access.lsPackages(entity, [opts]) -> Promise<Object | null>`

`entity` must be either a valid org or user name, or a fully-qualified team name in the `scope:team` format, with or without the `@` prefix.

Lists out packages a user, org, or team has access to, with corresponding permissions. Packages that the access token does not have access to won't be listed.

In order to disambiguate between users and orgs, two requests may end up being made when listing orgs or users.

For a streamed version of these results, see [`access.lsPackages.stream()`].

Example

```
await access.lsPackages('zkat', {
  token: 'myregistrytoken'
})
// Lists all packages `@zkat` has access to on the registry, and the
// corresponding permissions.
```

`> access.lsPackages.stream(scope, [team], [opts]) -> Stream`

`entity` must be either a valid org or user name, or a fully-qualified team name in the `scope:team` format, with or without the `@` prefix.

Streams out packages a user, org, or team has access to, with corresponding permissions, with each stream entry being formatted like `[packageName, permissions]`. Packages that the access token does not have access to won't be listed.

In order to disambiguate between users and orgs, two requests may end up being made when listing orgs or users.

The returned stream is a valid `asyncIterator`.

Example

```
for await (let [pkg, perm] of access.lsPackages.stream('zkat')) {
  console.log('zkat has', perm, 'access to', pkg)
}
// zkat has read-write access to eggplant
// zkat has read-only access to @npmcorp/secret
```

`> access.lsCollaborators(spec, [user], [opts]) -> Promise<Object | null>`

`spec` must be an [`npm-package-arg`]-compatible registry spec. `user` must be a valid user name, with or without the `@` prefix.

Lists out access privileges for a certain package. Will only show permissions for packages to which you have at least read access. If `user` is passed in, the list is filtered only to teams *that* user happens to belong to.

For a streamed version of these results, see [`access.lsCollaborators.stream()`].

Example

```
await access.lsCollaborators('@npm/foo', 'zkat', {
  token: 'myregistrytoken'
})
// Lists all teams with access to @npm/foo that @zkat belongs to.
```

`> access.lsCollaborators.stream(spec, [user], [opts]) -> Stream`

`spec` must be an [`npm-package-arg`]-compatible registry spec. `user` must be a valid user name, with or without the `@` prefix.

Stream out access privileges for a certain package, with each entry in `[user, permissions]` format. Will only show permissions for packages to which you have at least read access. If `user` is passed in, the list is filtered only to teams *that* user happens to belong to.

The returned stream is a valid `asyncIterator`.

Example

```
for await (let [usr, perm] of access.lsCollaborators.stream('npm')) {
  console.log(usr, 'has', perm, 'access to npm')
}
// zkat has read-write access to npm
// iarna has read-write access to npm
```