

---

## scratch-gui

Scratch GUI is a set of React components that comprise the interface for creating and running Scratch 3.0 projects

To open the current build in your browser on Github Pages:

<https://scratchfoundation.github.io/scratch-gui/>

### Installation

This requires you to have Git and Node.js installed.

In your own node environment/application:

```
1 npm install https://github.com/scratchfoundation/scratch-gui.git
```

If you want to edit/play yourself:

```
1 git clone https://github.com/scratchfoundation/scratch-gui.git
2 cd scratch-gui
3 npm install
```

**You may want to add `--depth=1` to the `git clone` command because there are some large files in the git repository history.**

### Getting started

Running the project requires Node.js to be installed.

### Running

Open a Command Prompt or Terminal in the repository and run:

```
1 npm start
```

Then go to <http://localhost:8601/> - the playground outputs the default GUI component

---

## Developing alongside other Scratch repositories

### Getting another repo to point to this code

If you wish to develop `scratch-gui` alongside other scratch repositories that depend on it, you may wish to have the other repositories use your local `scratch-gui` build instead of fetching the current production version of the scratch-gui that is found by default using `npm install`.

Here's how to link your local `scratch-gui` code to another project's `node_modules/scratch-gui`.

### Configuration

1. In your local `scratch-gui` repository's top level:
  1. Make sure you have run `npm install`
  2. Build the `dist` directory by running `BUILD_MODE=dist npm run build`
  3. Establish a link to this repository by running `npm link`
2. From the top level of each repository (such as `scratch-www`) that depends on `scratch-gui`:
  1. Make sure you have run `npm install`
  2. Run `npm link scratch-gui`
  3. Build or run the repository

**Using `npm run watch`** Instead of `BUILD_MODE=dist npm run build`, you can use `BUILD_MODE=dist npm run watch` instead. This will watch for changes to your `scratch-gui` code, and automatically rebuild when there are changes. Sometimes this has been unreliable; if you are having problems, try going back to `BUILD_MODE=dist npm run build` until you resolve them.

**Oh no! It didn't work!** If you can't get linking to work right, try:

- Follow the recipe above step by step and don't change the order. It is especially important to run `npm install` before `npm link` as installing after the linking will reset the linking.
- Make sure the repositories are siblings on your machine's file tree, like `.../.../MY_SCRATCH_DEV_DIRECTORY/scratch-gui/` and `.../.../MY_SCRATCH_DEV_DIRECTORY/scratch-www/`.
- Consistent node.js version: If you have multiple Terminal tabs or windows open for the different Scratch repositories, make sure to use the same node version in all of them.

- 
- If nothing else works, unlink the repositories by running `npm unlink` in both, and start over.

## Testing

### Documentation

You may want to review the documentation for Jest and Enzyme as you write your tests.

See [jest cli docs](#) for more options.

### Running tests

*NOTE: If you're a Windows user, please run these scripts in Windows `cmd.exe` instead of Git Bash/MINGW64.*

Before running any tests, make sure you have run `npm install` from this (scratch-gui) repository's top level.

**Main testing command** To run linter, unit tests, build, and integration tests, all at once:

```
1 npm test
```

**Running unit tests** To run unit tests in isolation:

```
1 npm run test:unit
```

To run unit tests in watch mode (watches for code changes and continuously runs tests):

```
1 npm run test:unit -- --watch
```

You can run a single file of integration tests (in this example, the `button` tests):

```
1 $(npm bin)/jest --runInBand test/unit/components/button.test.jsx
```

**Running integration tests** Integration tests use a headless browser to manipulate the actual HTML and javascript that the repo produces. You will not see this activity (though you can hear it when sounds are played!).

To run the integration tests, you'll first need to install Chrome, Chromium, or a variant, along with Chromedriver.

Note that integration tests require you to first create a build that can be loaded in a browser:

---

```
1 npm run build
```

Then, you can run all integration tests:

```
1 npm run test:integration
```

Or, you can run a single file of integration tests (in this example, the `backpack` tests):

```
1 $(npm bin)/jest --runInBand test/integration/backpack.test.js
```

If you want to watch the browser as it runs the test, rather than running headless, use:

```
1 USE_HEADLESS=no $(npm bin)/jest --runInBand test/integration/backpack.test.js
```

## Troubleshooting

### Ignoring optional dependencies

When running `npm install`, you can get warnings about optional dependencies:

```
1 npm WARN optional Skipping failed optional dependency /chokidar/
  fsevents:
2 npm WARN notsup Not compatible with your operating system or
  architecture: fsevents@1.2.7
```

You can suppress them by adding the `no-optional` switch:

```
1 npm install --no-optional
```

Further reading: [Stack Overflow](#)

### Resolving dependencies

When installing for the first time, you can get warnings that need to be resolved:

```
1 npm WARN eslint-config-scratch@5.0.0 requires a peer of babel-eslint@
  ^8.0.1 but none was installed.
2 npm WARN eslint-config-scratch@5.0.0 requires a peer of eslint@^4.0 but
  none was installed.
3 npm WARN scratch-paint@0.2.0-prerelease.20190318170811 requires a peer
  of react-intl-redux@^0.7 but none was installed.
4 npm WARN scratch-paint@0.2.0-prerelease.20190318170811 requires a peer
  of react-responsive@^4 but none was installed.
```

You can check which versions are available:

---

```
1 npm view react-intl-redux@0.* version
```

You will need to install the required version:

```
1 npm install --no-optional --save-dev react-intl-redux@^0.7
```

The dependency itself might have more missing dependencies, which will show up like this:

```
1 user@machine:~/sources/scratch/scratch-gui (491-translatable-library-objects)$ npm install --no-optional --save-dev react-intl-redux@^0.7
2 scratch-gui@0.1.0 /media/cuideigin/Linux/sources/scratch/scratch-gui |—
3 react-intl-redux@0.7.0 |—
4 UNMET PEER DEPENDENCY react-responsive@5.0.0
```

You will need to install those as well:

```
1 npm install --no-optional --save-dev react-responsive@^5.0.0
```

Further reading: [Stack Overflow](#)

## Troubleshooting

If you run into npm install errors, try these steps:

1. run `npm cache clean --force`
2. Delete the `node_modules` directory
3. Delete `package-lock.json`
4. run `npm install` again

## Publishing to GitHub Pages

You can publish the GUI to [github.io](#) so that others on the Internet can view it. Read the [wiki](#) for a step-by-step guide.

## Understanding the project state machine

Since so much code throughout `scratch-gui` depends on the state of the project, which goes through many different phases of loading, displaying and saving, we created a “finite state machine” to make it clear which state it is in at any moment. This is contained in the file `src/reducers/project-state.js`.

It can be hard to understand the code in `src/reducers/project-state.js`. There are several types of data and functions used, which relate to each other:

---

## Loading states

These include state constant strings like:

- `NOT_LOADED` (the default state),
- `ERROR`,
- `FETCHING_WITH_ID`,
- `LOADING_VM_WITH_ID`,
- `REMIXING`,
- `SHOWING_WITH_ID`,
- `SHOWING_WITHOUT_ID`,
- etc.

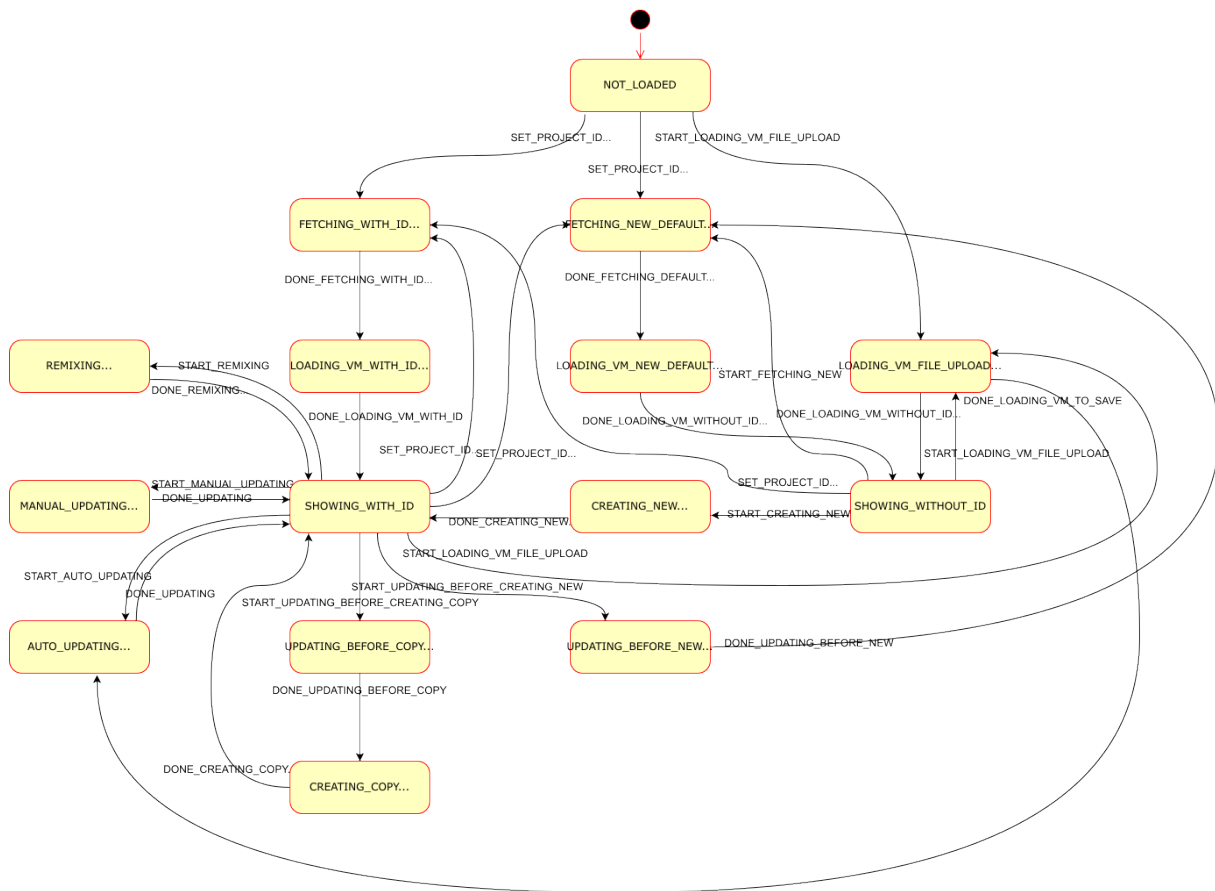
## Transitions

These are names for the action which causes a state change. Some examples are:

- `START_FETCHING_NEW`,
- `DONE_FETCHING_WITH_ID`,
- `DONE_LOADING_VM_WITH_ID`,
- `SET_PROJECT_ID`,
- `START_AUTO_UPDATING`,

## How transitions relate to loading states

Like this diagram of the project state machine shows, various transition actions can move us from one loading state to another:

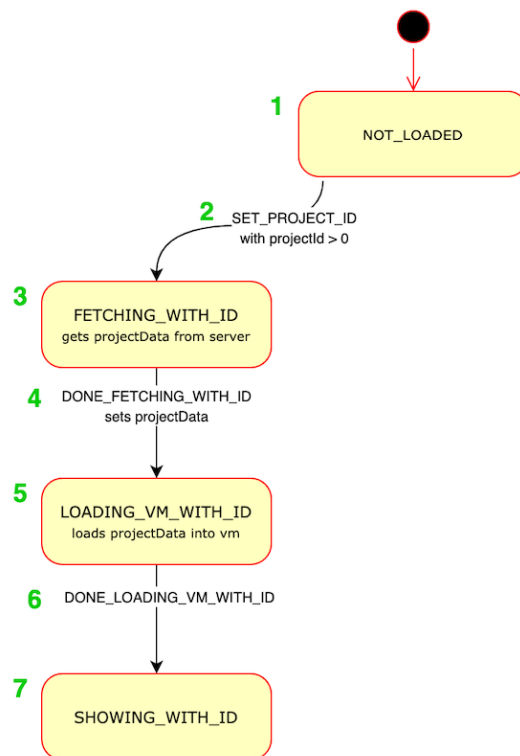


*Note: for clarity, the diagram above excludes states and transitions relating to error handling.*

**Example** Here's an example of how states transition.

Suppose a user clicks on a project, and the page starts to load with URL <https://scratch.mit.edu/projects/123456>.

Here's what will happen in the project state machine:



1. When the app first mounts, the project state is **NOT\_LOADED**.
2. The **SET\_PROJECT\_ID** redux action is dispatched (from `src/lib/project-fetcher-hoc.jsx`), with `projectId` set to 123456. This transitions the state from **NOT\_LOADED** to **FETCHING\_WITH\_ID**.
3. The **FETCHING\_WITH\_ID** state. In `src/lib/project-fetcher-hoc.jsx`, the `projectId` value 123456 is used to request the data for that project from the server.
4. When the server responds with the data, `src/lib/project-fetcher-hoc.jsx` dispatches the **DONE\_FETCHING\_WITH\_ID** action, with `projectData` set. This transitions the state from **FETCHING\_WITH\_ID** to **LOADING\_VM\_WITH\_ID**.
5. The **LOADING\_VM\_WITH\_ID** state. In `src/lib/vm-manager-hoc.jsx`, we load the `projectData` into Scratch's virtual machine ("the vm").
6. When loading is done, `src/lib/vm-manager-hoc.jsx` dispatches the **DONE\_LOADING\_VM\_WITH\_ID** action. This transitions the state from **LOADING\_VM\_WITH\_ID** to **SHOWING\_WITH\_ID**.
7. The **SHOWING\_WITH\_ID** state. Now the project appears normally and is playable and editable.

## Donate

We provide Scratch free of charge, and want to keep it that way! Please consider making a donation to support our continued engineering, design, community, and resource development efforts. Dona-



---

tions of any size are appreciated. Thank you!