# Contributing

There are many ways to contribute to the Bot Framework Emulator project: reporting issues, submitting pull requests, and creating suggestions.

## Submitting Issues

The Bot Framework Emulator project tracks issues and feature requests using [GitHub issue tracker](https://github.com/Microsoft/BotFramework-Emulator/issues).

### Before Submitting an Issue

First, please do a search in open issues to see if the issue or feature request has already been filed. If there is an existing issue, add your comments to that issue.

If your issue is a question, consider [asking it on Stack Overflow](https://stackoverflow.com/questions/ask?tags=botframework) using the tag `botframework`.

### Writing Great Issues and Suggestions

\* Provide reproducible steps, what the result of the steps was, and what you would have expected to happen.

\* Always file a single bug or feature request per issue. Do not list multiple bugs or requests in the same issue.

\* Do not add your issue as a comment to an existing issue unless it's for the identical input. Many issues look similar, but have different causes.

\* Include a screenshot or animated GIF.

Don't feel bad if we can't reproduce the issue and ask you for more information!

\*\*\*

## How to build from source

### Clone the repo

```

git clone https://github.com/Microsoft/BotFramework-Emulator.git

```

### Navigate to the project

```

cd BotFramework-Emulator

```

### Install global dependencies

> \*\*NOTE:\*\* Due to the version of Electron that the Emulator uses, it's recommended to use \*\*Node v10.14.2\*\* when building the project from source. Later versions might result in build or runtime errors.

>

> \*\*npm version 5.6.0\*\* or greater is also required. (Node v10.14.x includes 6.4.1)

```

npm i -g lerna@3.4.0

```

> \*\*NOTE:\*\* If you are using Linux, building the Emulator might result in an error due to a missing package: \*\*libXScrnSaver\*\*. If you run into this error, install the package using your OS's package manager and retry:

>

>`yum install libXScrnSaver`

>

>The Emulator (on Linux) also leverages a library that uses `libsecret` so you may need to install it before running `npm install`.

>

> Depending on your distribution, you will need to run the following command:

>

> Debian/Ubuntu: `sudo apt-get install libsecret-1-dev`

>

> Red Hat-based: `sudo yum install libsecret-devel`

>

> Arch Linux: `sudo pacman -S libsecret`

### Install local dependencies

```

npm run bootstrap

```

> \*\*IMPORTANT:\*\* Do \*\*not\*\* run `npm install` in any of the directories; lerna will take care of that for you with the

`bootstrap` command.

> If you plan on running the end-to-end tests, please use `npm run bootstrap:dev` instead to bootstrap the `@bfemulator/tools` package.

### Build all packages

```

npm run build

```

## How to run & develop locally

Open 2 terminals:

- One in `packages/app/client` (will be responsible for the renderer process)

- run `npm run start`

- that's all you have to do; you shouldn't have to worry about the client side again unless you modify code in

`packages/app/shared` and rebuild the `shared` package

- One in `packages/app/main` (will be responsible for the node process)

- run `npm run start:watch`

- this starts a new instance of the electron app with the most recently compiled `packages/app/main` files, and will continue to watch for any changes before recompiling and restarting the electron instance

### Debugging

#### The Main Process

Running `npm run start:watch` opens up port 7777 for debugging the main node process. Startup is non-blocking

by default which means code could be executed before you have time to attach your debugger and set breakpoints. To prevent this,

change `--inspect=7777` to `--inspect-brk=7777` in the `start:electron` script in the `package.json` located in `packages\app\main`.

This will prevent code from running until after a debug process has been attached and will require you to start

the debug process before the emulator is displayed.

Setting up a node debugger depends on your tooling. Please refer to the documentation on setting up a node debugger

for your flavor of tools. For more information on debugging NodeJS in general, refer to [this guide](https://nodejs.org/en/docs/guides/debugging-getting-started/)

#### The Client

Debugging the client is done remotely and can be done via your browser. Instructions will be different depending on your browser. Follow these instructions to debug the client side within \*\*Google Chrome\*\*:

1. Open Google Chrome

2. Navigate to the Inspect page by typing `chrome://inspect` in the address bar and pressing `ENTER`

3. Tell Chrome to listen to `localhost` ports `7777` & `7778` by clicking the `Configure` button in the \*\*Devices\*\* section, and then typing `localhost:7777` into the input and pressing enter. Do the same for `localhost:7778`. Now click `Done`.

4. In the \*\*Remote Target\*\* section, you should now see an entry for `localhost:3000` which is the webpack dev server serving the client side of the Emulator. Click `Inspect` to bring up the Chrome DevTools for the client side. Now you can debug the client as you would any other web app. (If you are unfamiliar with the Chrome DevTools, please take a look at [their documentation.](https://developers.google.com/web/tools/chrome-devtools/javascript/))

>\*\*NOTE:\*\* If you are using Chrome, you might want to turn off Device Emulation / Screencast mode if you want to be able to mouse over UI elements and highlight them in the DevTools inspector. You can disable Screencasting by clicking the `Toggle screencast` button in the top left corner of the DevTools window.

\*\*\*

## Pull Requests

Before we can accept a pull request from you, you must agree to the

[Contributor License Agreement (CLA)](https://cla.opensource.microsoft.com/).

It is an automated process and you only need to do this once.

To enable us to quickly review and accept your pull requests, always create one pull request per issue and link the

issue in the pull request. Never merge multiple requests in one unless they have the same root cause. Keep code changes

as small as possible. Avoid pure formatting changes to code that has not been modified otherwise.