## Development setup

We welcome any quality bugfixes or contributions!

To avoid a conflicts with your existing installation, delete the installed extension at `~/.vscode/extensions/msjsdiag.vscode-react-native`.

### Windows

\* In `C:/Users/<username>/.vscode/extensions/`, `git clone` this repository

### macOS/Linux

\* `git clone` this repository

\* Run `ln -s <path to repo> ~/.vscode/extensions/vscode-react-native`

\* You could clone it to the extensions directory if you want, but working with hidden folders in macOS can be a pain.

### Then...

\* `cd` to the folder you just cloned

\* Run `npm install -g gulp` and `npm install`

\* You may see an error if `bufferutil` or `utf-8-validate` fail to build. These native modules required by `ws` are optional and the debug adapter should work fine without them.

\* Run `gulp build`

## Debugging

There are currently 3 components to our extension: The extension running in the VS Code process, the debug adapter, and some code wrapping the user react-native code which is launched by the debug adapter. These are all debugged in different ways:

\* To debug the extension process itself, in VS Code run the `Launch Extension` debug target which will spawn a new instance of VS Code with the extension installed. You can set breakpoints in the TypeScript and debug things such as extension activation and the command palette.

\* To debug the code running in the same process as the React Native code, open up an instance of VS Code running the extension on a React Native project. From this instance, open up the TypesSript file in the extension codebase that you wish to debug and add breakpoints. Now, when you launch the react-native project, you should hit breakpoints in the extension code wrapper.

\* To debug the debug adapter, open the extension in VS Code and run the `Debug debugger` launch target. This will start the debug adapter as a server rather than standalone process waiting for debug sessions to connect. Then inside a React Native project add `"debugServer": 4712` to the configuration in `launch.json` you will use to start debugging to get it to use the standalone instance. Notice that before debugging the adapter, you must ensure that `debugger/nodeDebugLocation.json` contains a `nodeDebugPath` entry which points to the location of the node debug adapter extension - to do so you could just run the extension normally and open a React Native project to generate this file.

## Testing

There is a set of mocha tests for the debug adapter which can be run with `npm test`, and a set of mocha tests for the other functionality run as part of the `test` launch config. Also run `gulp tslint` to check your code against our tslint rules.

See the project under test/testProjcet/ for a sample project that should build and compile, allow debugging of plugins and merges, and enable IntelliSense for plugins.

## Legal

You must complete a Contributor License Agreement (CLA). Briefly, this agreement testifies that you are granting us permission to use the submitted change according to the terms of the project's license, and that the work being submitted is under appropriate copyright.

Please submit a Contributor License Agreement (CLA) before submitting a pull request. You may visit [https://cla.microsoft.com](https://cla.microsoft.com) to sign the agreement digitally. Alternatively, download the agreement from ([Microsoft Contribution License Agreement.docx](https://www.codeplex.com/Download?ProjectName=typescript&DownloadId=822190) or [Microsoft Contribution License Agreement.pdf](https://www.codeplex.com/Download?ProjectName=typescript&DownloadId=921298)), sign, scan, and email it back to <cla@microsoft.com>. Be sure to include your github user name along with a copy of the signed agreement. Once we have received the signed CLA, we'll review the request.

## Sending a PR

Pull requests should:

\* Include a clear description of the proposed change

\* Be a child commit of a reasonably recent commit in the \*\*master\*\* branch

\* Requests need not be a single commit, but should be a linear sequence of commits (i.e. no merge commits in your PR)

\* Have clear commit messages

\* e.g. "Refactor feature", "Fix issue", "Add tests for issue"

\* Include adequate tests

\* At least one test should fail in the absence of your non-test code changes. If your PR does not match this criteria, please specify why

\* Tests should include reasonable permutations of the target fix/change

\* Include baseline changes with your change

\* Have linting issues (`gulp tslint`)

It is desirable, but not necessary, for the tests to pass at each commit.

To avoid line ending issues, set `autocrlf = input` and `whitespace = cr-at-eol` in your system's git configuration.

## Troubleshooting

\* VS Code complains about Q promises and mark it as problems almost everywhere

This problems caused by incorrect Intellisense Typescript version choosed for the workspace.

Change the Typescript version to workspace version in the lower right corner of the VS Code window.

![Typescript Intellisense version](images/typescript-version.png)