# How to Contribute Changes

## Contribution Steps

\* [Build and debug the extension](Documentation/Building%20the%20Extension.md).

\* File an [issue](https://github.com/Microsoft/vscode-cpptools/issues) and a [pull request](https://github.com/Microsoft/vscode-cpptools/pulls) with the change and we will review it.

\* If the change affects functionality, add a line describing the change to [\*\*CHANGELOG.md\*\*](Extension/CHANGELOG.md).

\* Try and add a test in [\*\*test/extension.test.ts\*\*](Extension/test/unitTests/extension.test.ts).

\* Run tests via opening the [\*\*Extension\*\*](https://github.com/Microsoft/vscode-cpptools/tree/master/Extension) folder in Visual Studio Code, selecting the "Launch Tests" configuration in the Debug pane, and choosing "Start Debugging".

## About the Code

\* Execution starts in the `activate` method in [\*\*main.ts\*\*](Extension/src/main.ts).

\* `processRuntimeDependencies` handles the downloading and installation of the OS-dependent files. Downloading code exists in [\*\*packageManager.ts\*\*](Extension/src/packageManager.ts).

\* `downloadCpptoolsJsonPkg` handles the \*\*cpptools.json\*\*, which can be used to enable changes to occur mid-update, such as turning the `intelliSenseEngine` to `"Default"` for a certain percentage of users.

\* The debugger code is in the [\*\*Debugger\*\*](https://github.com/Microsoft/vscode-cpptools/tree/master/Extension/src/Debugger) folder.

\* [\*\*LanguageServer/client.ts\*\*](Extension/src/LanguageServer/client.ts) handles various language server functionality.

\* [\*\*LanguageServer/configurations.ts\*\*](Extension/src/LanguageServer/configurations.ts) handles functionality related to \*\*c\_cpp\_properties.json\*\*.

\* [\*\*telemetry.ts\*\*](Extension/src/telemetry.ts): Telemetry data gets sent to either `logLanguageServerEvent` or `logDebuggerEvent`.

\* The Tag Parser (symbol database) doesn't automatically expand macros, so the [\*\*cpp.hint\*\*](Extension/cpp.hint) file contains definitions of macros that should be expanded in order for symbols to be parsed correctly.

## String Localization

\* [vscode-nls](https://github.com/microsoft/vscode-nls) is used to localize strings in TypeScript code. To use [vscode-nls](https://github.com/microsoft/vscode-nls), the source file must contain:

```typescript

import \* as nls from 'vscode-nls';

nls.config({ messageFormat: nls.MessageFormat.bundle, bundleFormat: nls.BundleFormat.standalone })();

const localize: nls.LocalizeFunc = nls.loadMessageBundle();

```

\* For each user-facing string, wrap the string in a call to localize:

```typescript

const readmeMessage: string = localize("refer.read.me", "Please refer to {0} for troubleshooting information. Issues can be created at {1}", readmePath, "https://github.com/Microsoft/vscode-cpptools/issues");

```

\* The first parameter to localize should be a unique key for that string, not used by any other call to localize() in the file unless representing the same string. The second parameter is the string to localize. Both of these parameters must be string literals. Tokens such as {0} and {1} are supported in the localizable string, with replacement values passed as additional parameters to localize().