# Instructions for Logging Issues

## 1. Read the FAQ

Please [read the FAQ](https://github.com/Microsoft/TypeScript/wiki/FAQ) before logging new issues, even if you think you have found a bug.

Issues that ask questions answered in the FAQ will be closed without elaboration.

## 2. Search for Duplicates

[Search the existing issues](https://github.com/Microsoft/TypeScript/search?type=Issues) before logging a new one.

Some search tips:

\* \*Don't\* restrict your search to only open issues. An issue with a title similar to yours may have been closed as a duplicate of one with a less-findable title.

\* Check for synonyms. For example, if your bug involves an interface, it likely also occurs with type aliases or classes.

\* Search for the title of the issue you're about to log. This sounds obvious but 80% of the time this is sufficient to find a duplicate when one exists.

\* Read more than the first page of results. Many bugs here use the same words so relevancy sorting is not particularly strong.

\* If you have a crash, search for the first few topmost function names shown in the call stack.

## 3. Do you have a question?

The issue tracker is for \*\*issues\*\*, in other words, bugs and suggestions.

If you have a \*question\*, please use [Stack Overflow](http://stackoverflow.com/questions/tagged/typescript), [Gitter](https://gitter.im/Microsoft/TypeScript), your favorite search engine, or other resources.

Due to increased traffic, we can no longer answer questions in the issue tracker.

## 4. Did you find a bug?

When logging a bug, please be sure to include the following:

\* What version of TypeScript you're using (run `tsc --v`)

\* If at all possible, an \*isolated\* way to reproduce the behavior

\* The behavior you expect to see, and the actual behavior

You can try out the nightly build of TypeScript (`npm install typescript@next`) to see if the bug has already been fixed.

## 5. Do you have a suggestion?

We also accept suggestions in the issue tracker.

Be sure to [check the FAQ](https://github.com/Microsoft/TypeScript/wiki/FAQ) and [search](https://github.com/Microsoft/TypeScript/issues?utf8=%E2%9C%93&q=is%3Aissue) first.

In general, things we find useful when reviewing suggestions are:

\* A description of the problem you're trying to solve

\* An overview of the suggested solution

\* Examples of how the suggestion would work in various places

\* Code examples showing e.g. "this would be an error, this wouldn't"

\* Code examples showing the generated JavaScript (if applicable)

\* If relevant, precedent in other languages can be useful for establishing context and expected behavior

# Instructions for Contributing Code

## Tips

### Faster clones

The TypeScript repository is relatively large. To save some time, you might want to clone it without the repo's full history using `git clone --depth=1`.

### Using local builds

Run `gulp` to build a version of the compiler/language service that reflects changes you've made. You can then run `node <repo-root>/built/local/tsc.js` in place of `tsc` in your project. For example, to run `tsc --watch` from within the root of the repository on a file called `test.ts`, you can run `node ./built/local/tsc.js --watch test.ts`.

## Contributing bug fixes

TypeScript is currently accepting contributions in the form of bug fixes. A bug must have an issue tracking it in the issue tracker that has been approved (labelled ["help wanted"](https://github.com/Microsoft/TypeScript/issues?q=is%3Aopen+is%3Aissue+label%3A%22help+wanted%22) or in the "Backlog milestone") by the TypeScript team. Your pull request should include a link to the bug that you are fixing. If you've submitted a PR for a bug, please post a comment in the bug to avoid duplication of effort.

## Contributing features

Features (things that add new or improved functionality to TypeScript) may be accepted, but will need to first be approved (labelled ["help wanted"](https://github.com/Microsoft/TypeScript/issues?q=is%3Aopen+is%3Aissue+label%3A%22help+wanted%22) or in the "Backlog" milestone) by a TypeScript project maintainer in the suggestion issue. Features with language design impact, or that are adequately satisfied with external tools, will not be accepted.

Design changes will not be accepted at this time. If you have a design change proposal, please log a suggestion issue.

## Legal

You will need to 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. Upon submitting a pull request, you will automatically be given instructions on how to sign the CLA.

## Housekeeping

Your pull request should:

\* Include a description of what your change intends to do

\* 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)

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

\* Have clear commit messages

\* e.g. "Minor refactor in goToTypeDefinition", "Fix iterated type in for-await-of", "Add test for preserveWatchOutput on command line"

\* 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

\* All changed code must have 100% code coverage

\* Follow the code conventions described in [Coding guidelines](https://github.com/Microsoft/TypeScript/wiki/Coding-guidelines)

\* To avoid line ending issues, set `autocrlf = input` and `whitespace = cr-at-eol` in your git configuration

## Contributing `lib.d.ts` fixes

There are three relevant locations to be aware of when it comes to TypeScript's library declaration files:

\* `src/lib`: the location of the sources themselves.

\* `lib`: the location of the last-known-good (LKG) versions of the files which are updated periodically.

\* `built/local`: the build output location, including where `src/lib` files will be copied to.

Any changes should be made to [src/lib](https://github.com/Microsoft/TypeScript/tree/master/src/lib). \*\*Most\*\* of these files can be updated by hand, with the exception of any generated files (see below).

Library files in `built/local/` are updated automatically by running the standard build task:

```sh

gulp

```

The files in `lib/` are used to bootstrap compilation and usually \*\*should not\*\* be updated unless publishing a new version or updating the LKG.

### Modifying generated library files

The files `src/lib/dom.generated.d.ts` and `src/lib/webworker.generated.d.ts` both represent type declarations for the DOM and are auto-generated. To make any modifications to them, you will have to direct changes to https://github.com/Microsoft/TSJS-lib-generator

## Running the Tests

To run all tests, invoke the `runtests-parallel` target using gulp:

```Shell

gulp runtests-parallel

```

This will run all tests; to run only a specific subset of tests, use:

```Shell

gulp runtests --tests=<regex>

```

e.g. to run all compiler baseline tests:

```Shell

gulp runtests --tests=compiler

```

or to run a specific test: `tests\cases\compiler\2dArrays.ts`

```Shell

gulp runtests --tests=2dArrays

```

## Debugging the tests

You can debug with VS Code or Node instead with `gulp runtests --inspect`:

```Shell

gulp runtests --tests=2dArrays --inspect

```

You can also use the [provided VS Code launch configuration](./.vscode/launch.template.json) to launch a debug session for an open test file. Rename the file 'launch.json', open the test file of interest, and launch the debugger from the debug panel (or press F5).

## Adding a Test

To add a new test case, simply place a `.ts` file in `tests\cases\compiler` containing code that exemplifies the bugfix or change you are making.

These files support metadata tags in the format `// @metaDataName: value`.

The supported names and values are the same as those supported in the compiler itself, with the addition of the `fileName` flag.

`fileName` tags delimit sections of a file to be used as separate compilation units.

They are useful for tests relating to modules.

See below for examples.

\*\*Note\*\* that if you have a test corresponding to a specific spec compliance item, you can place it in `tests\cases\conformance` in an appropriately-named subfolder.

\*\*Note\*\* that filenames here must be distinct from all other compiler testcase names, so you may have to work a bit to find a unique name if it's something common.

### Tests for multiple files

When one needs to test for scenarios which require multiple files, it is useful to use the `fileName` metadata tag as such:

```TypeScript

// @fileName: file1.ts

export function f() {

}

// @fileName: file2.ts

import { f as g } from "file1";

var x = g();

```

One can also write a project test, but it is slightly more involved.

## Managing the Baselines

Compiler testcases generate baselines that track the emitted `.js`, the errors produced by the compiler, and the type of each expression in the file. Additionally, some testcases opt in to baselining the source map output.

When a change in the baselines is detected, the test will fail. To inspect changes vs the expected baselines, use

```Shell

gulp diff

```

After verifying that the changes in the baselines are correct, run

```Shell

gulp baseline-accept

```

to establish the new baselines as the desired behavior. This will change the files in `tests\baselines\reference`, which should be included as part of your commit. It's important to carefully validate changes in the baselines.

## Localization

All strings the user may see are stored in [`diagnosticMessages.json`](./src/compiler/diagnosticMessages.json).

If you make changes to it, run `gulp generate-diagnostics` to push them to the `Diagnostic` interface in `diagnosticInformationMap.generated.ts`.

See [coding guidelines on diagnostic messages](https://github.com/Microsoft/TypeScript/wiki/Coding-guidelines#diagnostic-messages).