# How to contribute

We would love to have your help. Before you start working however, please read

and follow this short guide.

# Reporting Issues

Provide as much information as possible. Mention the version of Jitsi Meet,

Jicofo and JVB you are using, and explain (as detailed as you can) how the

problem can be reproduced.

# Code contributions

Found a bug and know how to fix it? Great! Please read on.

## Contributor License Agreement

While the Jitsi projects are released under the

[Apache License 2.0](https://github.com/jitsi/jitsi-meet/blob/master/LICENSE), the copyright

holder and principal creator is [8x8](https://www.8x8.com/). To

ensure that we can continue making these projects available under an Open Source license,

we need you to sign our Apache-based contributor

license agreement as either a [corporation](https://jitsi.org/ccla) or an

[individual](https://jitsi.org/icla). If you cannot accept the terms laid out

in the agreement, unfortunately, we cannot accept your contribution.

## Creating Pull Requests

- Make sure your code passes the linter rules beforehand. The linter is executed

automatically when committing code.

- Perform \*\*one\*\* logical change per pull request.

- Maintain a clean list of commits, squash them if necessary.

- Rebase your topic branch on top of the master branch before creating the pull

request.

## Coding style

### Comments

\* Comments documenting the source code are required.

\* Comments from which documentation is automatically generated are \*\*not\*\*

subject to case-by-case decisions. Such comments are used, for example, on

types and their members. Examples of tools which automatically generate

documentation from such comments include JSDoc, Javadoc, Doxygen.

\* Comments which are not automatically processed are strongly encouraged. They

are subject to case-by-case decisions. Such comments are often observed in

function bodies.

\* Comments should be formatted as proper English sentences. Such formatting pays

attention to, for example, capitalization and punctuation.

### Duplication

\* Don't copy-paste source code. Reuse it.

### Formatting

\* Line length is limited to 120 characters.

\* Sort by alphabetical order in order to make the addition of new entities as

easy as looking a word up in a dictionary. Otherwise, one risks duplicate

entries (with conflicting values in the cases of key-value pairs). For

example:

\* Within an `import` of multiple names from a module, sort the names in

alphabetical order. (Of course, the default name stays first as required by

the `import` syntax.)

````javascript

import {

DOMINANT\_SPEAKER\_CHANGED,

JITSI\_CLIENT\_CONNECTED,

JITSI\_CLIENT\_CREATED,

JITSI\_CLIENT\_DISCONNECTED,

JITSI\_CLIENT\_ERROR,

JITSI\_CONFERENCE\_JOINED,

MODERATOR\_CHANGED,

PEER\_JOINED,

PEER\_LEFT,

RTC\_ERROR

} from './actionTypes';

````

\* Within a group of imports (e.g. groups of imports delimited by an empty line

may be: third-party modules, then project modules, and eventually the

private files of a module), sort the module names in alphabetical order.

````javascript

import React, { Component } from 'react';

import { connect } from 'react-redux';

````

### Indentation

\* Align `switch` and `case`/`default`. Don't indent the `case`/`default` more

than its `switch`.

````javascript

switch (i) {

case 0:

...

break;

default:

...

}

````

### Naming

\* An abstraction should have one name within the project and across multiple

projects. For example:

\* The instance of lib-jitsi-meet's `JitsiConnection` type should be named

`connection` or `jitsiConnection` in jitsi-meet, not `client`.

\* The class `ReducerRegistry` should be defined in ReducerRegistry.js and its

imports in other files should use the same name. Don't define the class

`Registry` in ReducerRegistry.js and then import it as `Reducers` in other

files.

\* The names of global constants (including ES6 module-global constants) should

be written in uppercase with underscores to separate words. For example,

`BACKGROUND\_COLOR`.

\* The underscore character at the beginning of a name signals that the

respective variable, function, property is non-public i.e. private, protected,

or internal. In contrast, the lack of an underscore at the beginning of a name

signals public API.

### Feature layout

When adding a new feature, this would be the usual layout.

```

react/features/sample/

??? actionTypes.js

??? actions.js

??? components

?   ??? AnotherComponent.js

?   ??? OneComponent.js

?   ??? index.js

??? middleware.js

??? reducer.js

```

The middleware must be imported in `react/features/app/` specifically

in `middlewares.any`, `middlewares.native.js` or `middlewares.web.js` where appropriate.

Likewise for the reducer.

An `index.js` file must not be provided for exporting actions, action types and

component. Features / files requiring those must import them explicitly.

This has not always been the case and the entire codebase hasn't been migrated to

this model but new features should follow this new layout.

When working on an old feature, adding the necessary changes to migrate to the new

model is encouraged.