# Contribution Guide

## Introduction

First off, thank you for considering contributing to this project. It's people like you that make it such a great tool.

Following these guidelines helps to communicate that you respect the time of the developers managing and developing this open source project. In return, they should reciprocate that respect in addressing your issue, assessing changes, and helping you finalize your pull requests.

This is an open source project and we love to receive contributions from our community — you! There are many ways to contribute, from writing tutorials or blog posts, improving the documentation, submitting bug reports and feature requests or writing code which can be incorporated into the main project itself.

If you haven't already, come find us in [Discord](https://discord.gg/N9QVxbM). We want you working on things you're excited about, and we can give you instant feedback.

### I don't want to read this whole thing I just have a question!!

We currently allow our users to use the issue tracker for support questions. But please be wary that maintaining an open source project can take a lot of time from the maintainers. If asking for a support question, state it clearly and take the time to explain your problem properly. Also, if your problem is not strictly related to this project we recommend you to use [Stack Overlow](https://stackoverflow.com) instead.

## How Can I Contribute?

### Testing

We have a handful of unit tests, but most of our testbed consists of running it with existing projects.

Try our builds and pull requests in your own projects and let us know how it goes.

### Reporting Bugs

Before creating bug reports, please check the existing bug reports as you might find out that you don't need to create one. When you are creating a bug report, please include as many details as possible.

#### How Do I Submit A (Good) Bug Report?

[Create an issue](https://github.com/vis2k/Mirror/issues/new?template=bug\_report.md) on the project's repository and provide the following information.

Explain the problem and include additional details to help maintainers reproduce the problem:

\* \*\*Use a clear and descriptive title\*\* for the issue to identify the problem.

\* \*\*Provide a simplified project that reproduces the issue whenever possible.\*\*

\* \*\*Describe the exact steps which reproduce the problem\*\* in as many details as possible. For example, start by explaining how you used the project. When listing steps, \*\*don't just say what you did, but explain how you did it\*\*.

\* \*\*Provide specific examples to demonstrate the steps\*\*. It's always better to get more information. You can include links to files or GitHub projects, copy/pasteable snippets or even print screens or animated GIFS. If you're providing snippets in the issue, use [Markdown code blocks](https://help.github.com/articles/markdown-basics/#multiple-lines).

\* \*\*Describe the behavior you observed after following the steps\*\* and point out what exactly is the problem with that behavior.

\* \*\*Explain which behavior you expected to see instead and why.\*\*

\* \*\*If the problem wasn't triggered by a specific action\*\*, describe what you were doing before the problem happened and share more information using the guidelines below.

Provide more context by answering these questions:

\* \*\*Did the problem start happening recently\*\* (e.g. after updating to a new version) or was this always a problem?

\* If the problem started happening recently, \*\*can you reproduce the problem in an older version?\*\* What's the most recent version in which the problem doesn't happen?

\* \*\*Can you reliably reproduce the issue?\*\* If not, provide details about how often the problem happens and under which conditions it normally happens.

Include details about your configuration and environment:

\* \*\*Which version of the project are you using?\*\*

\* \*\*What's the name and version of the OS you're using\*\*?

\* \*\*Any other information that could be useful about you environment\*\*

### Suggesting Enhancements

This section guides you through submitting an enhancement suggestion for this project, including completely new features and minor improvements to existing functionality. Following these guidelines helps maintainers and the community understand your suggestion and find related suggestions.

Before creating enhancement suggestions, please check the list of enhancements suggestions in the issue tracker as you might find out that you don't need to create one. When you are creating an enhancement suggestion, please include as many details as possible.

#### How Do I Submit A (Good) Enhancement Suggestion?

[Create an issue](https://github.com/vis2k/Mirror/issues/new?template=feature\_request.md) on the project's repository and provide the following information:

\* \*\*Use a clear and descriptive title\*\* for the issue to identify the suggestion.

\* \*\*Provide a step-by-step description of the suggested enhancement\*\* in as many details as possible.

\* \*\*Provide specific examples to demonstrate the steps\*\*. It's always better to get more information. You can include links to files or GitHub projects, copy/pasteable snippets or even print screens or animated GIFS. If you're providing snippets in the issue, use [Markdown code blocks](https://help.github.com/articles/markdown-basics/#multiple-lines).

\* \*\*Describe the current behavior\*\* and \*\*explain which behavior you expected to see instead\*\* and why.

\* \*\*List some other similar projects where this enhancement exists.\*\*

\* \*\*Specify which version of the project you're using.\*\*

\* \*\*Specify the current environment you're using.\*\* if this is a useful information.

\* \*\*Provide a specific use case\*\* - Often we get requests for a feature not realizing there is already a way to fulfill their use case. In other words, don't just give us a solution, give us a problem.

### Creating Pull Requests

#### How Do I Submit A (Good) Pull Request?

Please send a [GitHub Pull Request](https://github.com/vis2k/Mirror/compare) with a clear list of what you've done (read more about [pull requests](http://help.github.com/pull-requests/)).

When you send a pull request, we will love you forever if you include unit tests.

We can always use more test coverage.

\* \*\*Keep pull requests small\*\* Never combine multiple things in the same pull request. It is an order of magnitude easier to review 10 small pull requests than 1 large pull request that combines all changes. A pull request with 300+ changed lines has almost 0% chance of getting merged even if it is the best code in the world.

\* \*\*Use a clear and descriptive title\*\* for the pull request to state the improvement you made to the code or the bug you solved.

\* \*\*Provide a link to the related issue\*\* if the pull request is a follow up of an existing bug report or enhancement suggestion.

\* \*\*Comment why this pull request represents an enhancement\*\* and give a rationale explaining why you did it that way and not another way.

\* \*\*Use the same coding style as the one used in this project\*\*.

\* \*\*Documentation:\*\* If your PR adds or changes any public properties or methods, you must retain the old versions preceded with `[Obsolete("Describe what to do / use instead")` attribute wherever possbile, and you must update any relevant pages in the /docs folder. It's not done until it's documented!

\* \*\*Welcome suggestions from the maintainers to improve your pull request\*\*.

Please follow our coding conventions (below) and make sure all of your commits are atomic (one feature per commit). Rebase your pull requests if necessary.

Always write a clear log message for your commits. One-line messages are fine for small changes, but bigger changes should look like this:

```sh

$ git commit -m "A brief summary of the commit""

>

> A paragraph describing what changed and its impact.

```

If your pull request breaks any test, it has no hope of being merged.

Here are some more [good practices](https://blog.ploeh.dk/2015/01/15/10-tips-for-better-pull-requests/) to follow when submitting pull requests to any project.

#### Optimizations

There are generally 2 types of optimizations, micro-optimizations and macro-optimizations. The distinction has nothing to do with how much they improve the program.

Micro-optimizations try to improve the performance of an application by replacing instructions with equivalent but more efficient instructions. Some example micro-optimizations include:

\* replace `i / 4` with `i >> 2`

\* eliminate an allocation.

\* replace `Vector3.Distance(a,b) < K` with `Vector3.SqrMagnitude(b - a) < K \* K`

\* convert a class to struct

Macro-optimizations try to improve the performance of an application by changing the algorithm. Some examples include:

\* Serialize a message once O(1), instead of for every single client O(n)

\* Change interest management algorithm, as of this writing every object checks every other object O(n^2), it could be replaced by a sweep and prune algorithm O(n log n)

\* When synchronizing movement, instead of synchronizing every position change, you could synchronize the velocity and let the other side predict the position.

Macro-optimizations tend to change the \*\*scalability\*\* of mirror, by changing an algorithm, you may now support 10x more customers on the same hardware, it is even possible for a macro optimization to make performance worse for small numbers. Macro optimization usually make a really big difference, but are much harder to make.

Micro-optimizations tend to change the performance of mirror in a linear way. There are some micro optimizations that make a huge impact on performance such as eliminating allocations in the hot path.

We prefer readable code over optimal code. We do not like any kind of optimization if it makes the code less readable (they generally do). For that reason, we require that both micro and macro optimization pull requests come with screenshots profiling a real game or at least a synthetic \*\*representative\*\* test. It is not enough to show that one operation is faster than the other, you must prove that this makes a significant difference in Mirror or in a real game using Mirror.

If your optimization pull request does not come with profiling data showing real gains in a meaningful test is has no hope of getting merged.

## Coding conventions

Start reading our code and you'll get the hang of it. We optimize for readability:

\* We indent using 4 spaces (soft tabs)

\* We value simplicity. The code should be easy to read and avoid magic

\* \*\*KISS / Occam's Razor\*\* - always use the most simple solution.

\* \*\*No Premature Optimizations\*\*

MMOs need to run for weeks without issues or exploits.

Only do GC optimizations and caching in hot path. Avoid it everywhere else to keep the code simple.

\* \*\*Curly Braces { }\*\*

Always use braces even for one line if's. Unity did this everywhere, and there is value in not accidentally missing a line in an if statement because there were no braces.

\* \*\*Naming\*\*

Follow [C# standard naming conventions](https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md). Also, be descriptive. \`NetworkIdentity identity\`, not \`NetworkIdentity uv\` or similar. If you need a comment to explain it, the name needs to be changed. For example, don't do `msg = ... // the message`, use `message = ...` without a comment instead. Avoid prefixes like `m\_`, `s\_`, or similar.

\* \*\*private\*\*

Fields and methods in a class are private by default, no need to use the private keyword there.

\* This is open source software. Consider the people who will read your code, and make it look nice for them. It's sort of like driving a car: Perhaps you love doing donuts when you're alone, but with passengers the goal is to make the ride as smooth as possible.

\*\*The Python Way\*\*

Unlike Python, C# has different ways to do the same thing, which causes endless discussions and pull requests to change from one way to the other. Let's always use the most simple, most obvious way:

\* \*\*type\*\* vs. \*\*var\*\*: always use 'int x' instead of 'var x'. Less guess work. Less magic. If we \*\*always\*\* use the proper type then we have to waste no brain cycles on unnecessary decision making.

\* \*\*if\*\* vs. \*\*switch\*\*: any if statement could be converted to switch and back. Again, let's not have endless discussions and use if unless \_switch\_ makes overwhelmingly much sense. Python doesn't have switch either, they don't have those discussions and pull requests over there.

\* \*\*int\*\* vs. \*\*Int32\*\*: use int instead of Int32, double instead of Double, string instead of String and so on. We won't convert all ints to Int32, so it makes most sense to never use Int32 anywhere and avoid time wasting discussions.

\* \*\*Empty Class Bodies ({} vs. { })\*\*: please use 'class MyMessage : EmptyMessage {}' instead of 'class MyMessage : EmptyMessage { }'. For the same reason that we use no white space inbetween parameterless function defintions like void Start() vs. void Start( ).

Thanks.