# Contributing

:heart::+1: \*\*Thank you so much for taking the time to contribute!\*\* :+1::heart:

## Code standards

We have a worldwide community and respect everyones backgrounds and languages, however Forge Networking is primarily developed by English (US) speakers. Because of this the variables, names, functions, comments, and everything else for this code base should be exclusively in English (US).

### .editorconfig

We use .editorconfig to help with conforming to our coding conventions. Please make sure your chosen editor has this enabled. Check [editorconfig.org](https://editorconfig.org/#download) to see if you need to install a plugin or your IDE has this built in.

\*\*Anything not listed here should follow the [C# standard](https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines)\*\*

### Tabs vs spaces?

Sorry space fans, we are tabs. Though I know that this is an age old argument in computer science, we had to go with one or the other.

### Accessors

Yes, we are aware that if you do not provide an accessor that it will default to `private`. However, we like knowing that others are doing what they do on purpose. For this reason you are required to put `private` in front of your private members.

Member fields are to be camel case starting with an alpha character. For the most part people don't write C# in notepad so do not worry about any prefixes with underscores, or special things like `b\_` or any of that

```csharp

public int myNumber;

protected bool canHas;

private string catName;

```

Member properties are to be pascal case starting with an alpha character. Also their brackets are to be on 1 line if the behavior is simple. Otherwise you are to use the hanging brackets coding standards

```csharp

public int MyNumber { get; private set; }

public bool CanHas { get { return MyNumber > 0; } }

private string CatName

{

get

{

if (MyNumber > 5)

return "Pickles";

else if (CanHas)

return "Sure";

else

return "George?";

}

}

```

### Brackets

For single line conditional statements we do not have any brackets, just the line. Preferrably we put the code on the line following the if statement but for simple things like a return a single line is acceptable

```csharp

if (1 == 1)

DoSomethingFancy();

```

For multiple line conditional statements we put "hanging brackets" only"

```csharp

if (1 == 1)

{

FirstThing();

LastThing();

}

```

For infinite loops, while loops, and do-while loops, we need these to be exceptionally clear. So in the case of any of these we will use hanging brackets.

```csharp

while (true)

{

DoSomethingFancy();

}

do

{

DoSomethingFancy();

} while(true);

for(;;)

{

DoSomethingFancy();

}

```

For lambda expressions we have hanging brackets

```csharp

Call(() =>

{

InnerCode();

});

```

For initializer lists we have hanging brackets (no parenthisis needed)

```csharp

MyClass thing = new MyClass

{

a = 1,

b = 2,

c = 3

};

```

### Comments

Single line comments should include a space after the slashes

```csharp

// Notice the space after the two slashes

```

Commenting out code should not include a space after the slashes

```csharp

//noSpace.Code();

```

TODO comments should include the space after the slashes and then 2 spaces after the colon following the `TODO`

```csharp

// TODO: Notice the space after the slashes and the 2 spaces after the colon of todo

```

### Line length

We try to keep our line lengths short, try to stick around 90 characters maximum in the horizontal space and do not go any further than 120 characters. Or goals are to prevent any kind of horizontal scrolling or automatic line wrapping.

### Var vs type name

Though we accept code with `var` being used, it is hard to code review things listed as just `var` especially if they are the result of a function. If you are creating a new instance of something you can use `var` since the type name is in the line of declariation, otherwise please try to type the explicit type of the variable when it is the result of a function.

### Function length

We prefer very small, descriptive functions over long monolythic functions. Try to keep your functions to something around 4-9 lines. If your functions are longer than 9 lines, it probably is a good chance for breaking it into smaller local private methods or turning what is being processed into a `class` and handling the logic more inside of there. This is not only for code clarity and readability, but also it makes it much easier to debug knowing the names of functons and produces clear stack traces.

### Testing and TDD

We are currently trying out the Unity unit testing tools and they look promising so far. My hope is that we will be able to fall more into the TDD ([test driven development](https://en.wikipedia.org/wiki/Test-driven\_development)) flow of programming so that we have many tests for the code to make sure it continues to work as expected for all of the functionality. Of course, I know it is very difficult to unit test a multi-threaded, network based application, but it is important for us to try as much as possible to achieve a full test coverage of the system.

Please make sure that you either create a new file in the [ForgeUnity/Assets/BeardedManStudios/Scripts/Editor/Tests](https://github.com/BeardedManStudios/ForgeNetworkingRemastered/tree/develop/ForgeUnity/Assets/BeardedManStudios/Scripts/Editor/Tests) folder. Or if you are adding new tests that are either missing or covering new functionality of a class, feel free to update existing tests.