# Contributing

## Project Scope

The AngleSharp project ultimately tries to provide tools to parse, inspect, modify and interact with traditional web resources, such as HTML or CSS, for .NET development. Anything that is related to this goal will be considered. The project aims to be fully standards compliant. If your contribution is not following the standard, the chances of accepting it are limited.

## Code License

This is an open source project falling under the [MIT License](../LICENSE). By using, distributing, or contributing to this project, you accept and agree that all code within the AngleSharp project and its libraries are licensed under MIT license.

## Becoming a Contributor

Until the project has enough contributors a [BDFL](https://en.wikipedia.org/wiki/Benevolent\_dictator\_for\_life) model is followed. As such the sole key maintainer keeps the right to appoint GitHub members as regular project contributors. Nevertheless, usually appointing someone follows this process:

1. An individual contributes actively via discussions (reporting bugs, giving feedback to existing or opening new issues) and / or pull requests

2. The individual is either directly asked, invited or asks for contributor rights on the project

3. The individual uses the contribution rights to sustain or increase the active contributions

Every contributor has to sign the contributor's license agreement (CLA) to establish a legal trust between the project and its contributors.

## Working on AngleSharp

### Issue Discussion

Discussion of issues should be placed transparently in the issue tracker here on GitHub.

\* [AngleSharp.Core](https://github.com/AngleSharp/AngleSharp/issues/)

\* [AngleSharp.Css](https://github.com/AngleSharp/AngleSharp.Css/issues/)

\* [AngleSharp.Io](https://github.com/AngleSharp/AngleSharp.Io/issues/)

\* [AngleSharp.Js](https://github.com/AngleSharp/AngleSharp.Js/issues/)

\* [AngleSharp.Xml](https://github.com/AngleSharp/AngleSharp.Xml/issues/)

### Modifying the code

AngleSharp and its libraries uses features from the latest versions of C# (e.g., C# 7). You will therefore need a C# compiler that is up for the job.

1. Fork and clone the repo.

2. First try to build the AngleSharp.Core libray and see if you get the tests running.

3. You will be required to resolve some dependencies via NuGet.

AngleSharp itself does not have dependencies, however, the tests are dependent on NUnit.

The build system of AngleSharp uses Cake. A bootstrap script (build.ps1 for Windows or build.sh for \*nix systems) is included. Note, that at the moment AngleSharp may require NuGet 3.5, which looks for MSBuild pre-15, i.e., before Visual Studio 2017 on Windows systems. We aim to drop this requirement enitirely soon.

### Code Conventions

Most parts in the AngleSharp project are fairly straight forward. Among these are:

- Always use statement blocks for control statements, e.g., in a for-loop, if-condition, ...

- You may use a simple (throw) statement in case of enforcing contracts on argument

- Be explicit about modifiers (some files follow an older convention of the code base, but we settled on the explicit style)

There are a couple of rules, which are definitely not standard, but highly recommended for consistency and readability:

- AngleSharp uses the RHS convention, where types are always put on the right hand side if possible, i.e., preferring `var` under all circumstances

- A single empty line between two non-simple statements (e.g., for-loop and if-condition) should be inserted

- Types are preferred to keywords (`String` instead of `string` or `Int32` instead of `int`)

- `using` statements must be inside the namespace declaration

### Development Workflow

1. If no issue already exists for the work you'll be doing, create one to document the problem(s) being solved and self-assign.

2. Otherwise please let us know that you are working on the problem. Regular status updates (e.g. "still in progress", "no time anymore", "practically done", "pull request issued") are highly welcome.

2. Create a new branch—please don't work in the `master` branch directly. It is reserved for releases. We recommend naming the branch to match the issue being addressed (`feature/#777` or `issue-777`).

3. Add failing tests for the change you want to make. Tests are crucial and should be taken from W3C (or other specification).

4. Fix stuff. Always go from edge case to edge case.

5. All tests should pass now. Also your new implementation should not break existing tests.

6. Update the documentation to reflect any changes. (or document such changes in the original issue)

7. Push to your fork or push your issue-specific branch to the main repository, then submit a pull request against `devel`.

Just to illustrate the git workflow for AngleSharp a little bit more we've added the following graphs.

Initially, AngleSharp starts at the `master` branch. This branch should contain the latest stable (or released) version.

Here we now created a new branch called `devel`. This is the development branch.

Now active work is supposed to be done. Therefore a new branch should be created. Let's create one:

```

git checkout -b feature/#777

```

There may be many of these feature branches. Most of them are also pushed to the server for discussion or synchronization.

```

git push -u origin feature/#777

```

Now feature branches may be closed when they are done. Here we simply merge with the feature branch(es). For instance the following command takes the `feature/#777` branch from the server and merges it with the `devel` branch.

```

git checkout devel

git pull

git pull origin feature/#777

git push

```

Finally, we may have all the features that are needed to release a new version of AngleSharp. Here we tag the release. For instance for the 1.0 release we use `v1.0`.

```

git checkout master

git merge devel

git tag v1.0

```

(The last part is automatically performed by our CI system.)

### Basic Files

The following files should not be edited directly in the current repository, but rather in the `AngleSharp.GitBase` repository. They are then synced via `git pull` from a different remote.

```

.editorconfig

.gitignore

.gitattributes

.github/\*

appveyor.yml

build.ps1

build.sh

tools/anglesharp.cake

tools/packages.config

LICENSE

```

To sync manually:

```

git remote add gitbase git@github.com:AngleSharp/AngleSharp.GitBase.git

git pull gitbase master

```

### Versioning

The rules of [semver](http://semver.org/) don't necessarily apply here, but we will try to stay quite close to them.

Prior to version 1.0.0 we use the following scheme:

1. MINOR versions for reaching a feature milestone potentially combined with dramatic API changes

2. PATCH versions for refinements (e.g. performance improvements, bug fixes)

After releasing version 1.0.0 the scheme changes to become:

1. MAJOR versions at maintainers' discretion following significant changes to the codebase (e.g., API changes)

2. MINOR versions for backwards-compatible enhancements (e.g., performance improvements)

3. PATCH versions for backwards-compatible bug fixes (e.g., spec compliance bugs, support issues)

#### Code style

Regarding code style like indentation and whitespace, \*\*follow the conventions you see used in the source already.\*\* In general most of the [C# coding guidelines from Microsoft](https://msdn.microsoft.com/en-us/library/ff926074.aspx) are followed. This project prefers type inference with `var` to explicitly stating (redundant) information.

It is also important to keep a certain `async`-flow and to always use `ConfigureAwait(false)` in conjunction with an `await` expression.