# Contributing

## Report Issues and Request Changes

If you find a bug, please report it, including environment and examples of current behavior and what you believe to be the correct behavior. The clearer your description and information, the more likely it is someone will be able to make progress on it. The default issue template will help guide you through this.

## How to Make Changes (Implement Fixes and New Features)

Fixes and enhancements are totally welcome. We prefer you to file an issue before filing a PR, as this gives us chance to discuss design details, but feel free to dive right in.

### 0. Prereqisites for development

\* bash

\* make

\* nodejs - v10.x (with npm)

\* python - v2.7.x or v3.x (with pip)

If you encounter issues and cannot build, come chat on gitter.im. We're happy to help.

### 1. Build and Test Locally

This repository holds two mostly identical implementations of the beautifiers: a JavaScript implementation and a Python implementation.

While developing, you may locally build and test the JavaScript or Python (or both). The HTML beautifier is only implemented in JavaScript.

\* Familiarize yourself with the folder structure and code style before you dive in.

\* Make changes to the implementation of your choice.

\* If working in the JavaScript implementation:

\* Run `make js` to build and run unit tests

\* Run `make static` to manually test changes locally at `http://localhost:8080`

\* To load a debug (human readable) version of the beautifier source, open `http://localhost:8080/?debug`

\* If working in the Python implementation:

\* Run `make py` to build and run unit tests

\* Add tests to `/test/data/\*/test.js`.

\* Run `make jstest` or `make pytest` to run style checks, and to generate and run tests.

\* Include all changed files in your commit - The generated test files are checked in along with changes to the test data files.

### 2. Ensure Feature Parity

Any changes made to one implementation must be also made to the other implementation. \*\*This is required\*\*. Every time we make an exception to this requirement the project becomes harder to maintain. This will become painfully clear, should you find yourself unable to port changes from one implementation to the other due to implementations being out of sync. We made this a requirement several years ago and there are \*\*still\*\* open issues for changes that people promised to port "in the next week or two". The entire HTML beautifier is an example of this. :(

The implementations are already very similar and neither Python nor JavaScript are that hard to understand. Take the plunge, it is easier than you think. If you get stuck, go ahead and file a Pull Request and we can discuss how to move forward.

\* Run `make` (with no parameters) to run style checks, and to generate and run tests on both implementations.

\* Include all changed files in your commit - The generated test files are checked in along with changes to the test data files.

### 3. Update Documentation and Tools

Update documentation as needed. This includes files such as the README.md, internal command-line help, and file comments.

Also, check if your change needs any tooling updates. For example, the CDN URLs required additional scripting to update automatically for new releases.

### 4. Submit a Pull Request

\* Run `make ci` locally after commit but before creation of Pull Request. You may start a Pull Request even if this reports errors, but the PR will not be merged until all errors are fixed.

\* Include description of changes. Include examples of input and expected output if possible.

\* Pull requests must pass build checks on all platforms before being merged. We use Travis CI and AppVeyor to run tests on Linux and Windows across multiple versions of Node.js and Python.

# Folders

## Root

Some files related to specific implementations or platforms are found in the root folder, but most are cross-project tools and configuration.

## `js`

Files related to the JavaScript implementations of the beautifiers.

## `python`

Files related to the Python implementations of the beautifiers.

## `web`

Files related to https://beautifier.io/.

## `test`

Test data files and support files used to generate implementation-specific test files.

# Branches and Tags

## Master

We use the `master` branch as the primary development branch.

## Release

We use the `release` branch to hold releases, including built files for bower and the website.

# Tags

Each release is has a tag created for it in the `release` branch when it is published. The latest release is linked from the `README.md`.

## Attic

This project has been around for a while. While some parts have improved significantly over time, others fell

into disrepair and were mothballed. All branches named `attic-\*` are significantly out of date and kept for reference purposes only.

### PHP

`attic-php` contains a PHP implmenetation of the beautifier.

If you're interested in using it feel free.

If you plan to enhance it, please consider joining this project, and updating this version to match current functionality.

### Other Languages

Versions of the beautifier adapted to other languages are available on branch `attic-other`.

Take a look and feel free to resurrect them, but know it's pretty dusty back there.

### Generic Eval Unpacker

The `attic-genericeval` branch includes an unpacker that calls `eval` on whatever source is passed to it.

This file may be useful when working with source that unpacks itself when `eval` is called on it, but is also very unsafe.

We have isolated it on this separate branch to keep it from hurting the other children.

# How to publish a new version

NOTE: Before you publish a release make sure the latest changes have passed the Travis CI build!

## Setup

### Python

1. A PyPI user account from https://pypi.python.org/pypi?%3Aaction=register\_form.

2. Permissions to the jsbeautifier package. File an issue here on GitHub and the appropriate person will help you.

### Node

1. An npmjs.org user account from https://npmjs.org/signup.

2. Permissions to the js-beautify module on npmjs.org. File an issue here on GitHub and the appropriate person will help you.

## Publish to Production

To publish a release:

\* Close the Milestone for this release on github

\* Run `./tools/release-all.sh <version>`.

This is script will:

\* Update README.md with correct cdn links

\* Update CHANGLOG.md with the milestone description and a list of closed issues

\* Commit the built files to the `release` branch

\* Publish the python version to PyPI

\* Publish the javascript version to npm

\* Merge the changes and publish them on the gh-pages branch

NOTE: We keep Python and Node version numbers synchronized,

so if you publish a Python release, you publish a Node release as well.

## Publish to Beta Channel

To publish a Beta or RC (Release Candidate), add `-beta1` or `-rc1` to the end of the version, incrementing the number on the end as needed.