# Contributing

Everybody is more than welcome to contribute to Oga, no matter how small the

change. To keep everything running smoothly there are a bunch of guidelines that

one should follow.

## Code of Conduct

The code of conduct ("CoC") can be found in the file "CODE\_OF\_CONDUCT.md".

Everybody participating in this project must adhere to the rules and guidelines

stated in this CoC.

## General

\* When changing code make sure to write RSpec tests for the changes.

\* Document code using YARD. At the very least the method arguments and return

value(s) should be documented.

\* Use `raise` for raising errors instead of `fail`. You're raising errors after

all, not failing them.

## Submitting Changes

Before making any big changes it's best to open a GitLab issue to discuss the

matter, this saves you from potentially spending hours on something that might

ultimately be rejected.

When making changes please stick to the existing style and patterns as this

keeps the codebase consistent. If a certain pattern or style is getting in your

way please open a separate issue about this so it can be discussed.

Every commit and every merge request made is carefully reviewed. Chances are I'll

spend more time reviewing it than the time an author spent on their changes.

This should ensure that Oga's codebase is stable, of high quality and easy to

maintain. As such \_please\_ take my feedback into consideration (or discuss it in

a civilized manner) instead of just dismissing it with comments such as "But I

fixed the problem so your feedback is irrelevant" or "This is my way of doing

things".

Finally, and this will sound harsh: I will \_not\_ merge merge requests if the

author(s) simply disregard the feedback I've given them or if there are other

problems with the merge request. Do not expect me to just blindly accept whatever

changes are submitted.

Some examples of good merge requests:

\* https://gitlab.com/yorickpeterse/oga/-/merge\_requests/96

\* https://gitlab.com/yorickpeterse/oga/-/merge\_requests/67

\* https://gitlab.com/yorickpeterse/ffi-aspell/-/merge\_requests/21

\* https://gitlab.com/yorickpeterse/ffi-aspell/-/merge\_requests/20

\* https://gitlab.com/yorickpeterse/ruby-ll/-/merge\_requests/16

## Git

Git commits should have a <= 50 character summary, optionally followed by a

blank line and a more in depth description of 72 characters per line. For

example:

Use blacklists/whitelists for HTML closing rules

This allows for more fine grained control over when to close certain

elements. For example, an unclosed <tr> element should be closed first

when bumping into any element other than <td> or <th>. Using the old

NodeNameSet this would mean having to list every possible HTML element

out there. Using this new setup one can just create a whitelist of the

<td> and <th> elements.

Please, \_please\_ write meaningful commit messages. Writing a good commit

messages is \_just\_ as important as writing good code. If you're having trouble

writing a commit message you should try to break the commits up into smaller

chunks. You can do so using a `git rebase`.

## Editor Setup

Use spaces for indentation, tabs are not accepted. The usage of spaces ensures

the indentation is identical no matter what program or system is used to view

the source code.

Hard wrap lines at roughly 80 characters per line. Most modern editors can

easily handle this. For example, in Vim you can select text in visual mode

(using `v`) and press `gq` to automatically re-wrap the selected text.

It's OK if a line is a few characters longer than 80 but \_please\_ keep it as

close to 80 characters as possible. Typically I do this when wrapping the line

results in several extra lines without it being much more readable.

To make this process easier Oga comes with an [EditorConfig][editorconfig]

configuration file. If your editor supports this it will automatically apply

various settings for you.

## Hacking on Oga

Before you start hacking on Oga make sure the following libraries/tools are

installed:

\* Ragel 6.x (6.10 recommended), Ragel 7.x is not supported

\* gunzip (to unpack the fixtures)

\* javac (only when using JRuby)

Assuming you have the above tools installed and a local Git clone of Oga, first

you'll need to install the required Gems:

bundle install

Next up, compile the required files and run the tests:

rake

If you just want to generate various files (e.g. the C extension), run the

following instead:

rake generate

For more information about the available tasks, run `rake -T`.

## Running Benchmarks

Benchmarks are located in the `benchmark` directory. Some of these require

fixture files which can be generated by running `rake fixtures`. Running a

benchmark is just a matter of running a Ruby script, for example:

ruby benchmark/xml/parser/parser\_bench.rb

## Tests

Tests are written using RSpec and use the "expect" syntax. Specification blocks

should be written using `it`, grouping should be done using `describe`.

Specification descriptions should be meaningful and human-friendly English. For

example:

describe Oga::XML::Entities do

describe 'decode' do

it 'decodes &lt; into <' do

# ...

end

end

end

Typically the top-level `describe` block is used to describe a method name. In

such a case use `describe 'foo'` for class methods and `describe '#foo'` for

instance methods.

Whenever adding new specifications please keep them in the existing style. If

the style is problematic you can open a separate merge request to address it. If

you expect this to be a lot of work you should open an issue first to discuss

things.

## Continuous Integration

Oga is tested using GitLab CI. Merge requests require that all tests pass before

they can be merged.

## Extension Setup

Oga uses native extensions for the XML lexer. This is due to Ruby sadly not

being fast enough to chew through large amounts of XML (at least when using

Ragel). For example, the benchmark `benchmark/lexer/big\_xml\_time.rb` would take

around 6 seconds to complete on MRI 2.1.1. The native extensions on the other

hand can complete this benchmark in roughly 600 milliseconds.

Oga has two native extensions: one for MRI/Rubinius (written in C) and one for

JRuby (written in Java). Both extensions share the same Ragel grammar, found in

`ext/ragel/base\_lexer.rl`. This grammar is set up in such a way that the syntax

is compatible with both C and Java. Specific details on how the grammar is used

can be found in the documentation of said grammar file.

The native extensions call back in to Ruby to actually perform the task of

creating tokens, validating input and so forth. As a result of this you'll most

likely never have to touch the C and/or Java code when changing the behaviour

of the lexer.

To compile the extensions run `rake generate` using your Ruby implementation of

choice. Note that extensions compiled for MRI can not be used on Rubinius and

vice-versa. To compile the JRuby extension you'll have to switch your active

Ruby version to JRuby first.

## Thread Safety

To ensure Oga remains thread-safe for as much as possible the usage of global

objects and/or state is forbidden. This means that you should \_only\_ use

constants/class methods for static/read-only data (e.g. an Array of static

Strings). In other words, this is fine:

NUMBERS = [10, 20, 30]

NUMBERS.each do |number|

end

But this is not:

TOOL = SomeFindReplaceTool.new

output = TOOL.replace(input, 'foo', 'bar')

The exception here are libraries that are designed to be thread-safe, clearly

state this \_and\_ can prove it (e.g. by simply using a mutex). Even then global

state is highly frowned upon.

## Loading Libraries

All `require` calls should be placed in `lib/oga.rb`. Any `require` calls

specific to a Ruby implementation (e.g. JRuby) should be wrapped in a

conditional. For example:

if RUBY\_PLATFORM == 'java'

org.foo.bar.baz.DoSomething()

end

For loading files in Oga itself `require` should be used. \_Don't\_

modify `$LOAD\_PATH`, instead run any scripts using `ruby -I lib`.

## Contact

In case you have any further questions or would like to receive feedback before

submitting a change, feel free to contact me. You can either open an issue,

send a tweet to [@yorickpeterse][twitter] or send an Email to

<yorick@yorickpeterse.com>.

[editorconfig]:http://editorconfig.org/

[twitter]: https://twitter.com/yorickpeterse