EIRENE formatting

authors: Emil Løvbak, Xavier Bonnin, Oskar Lappi, Huw Leggate

5 May 2023

A meeting to discuss how to ensure consistent formatting and style in the EIRENE code.

# Different aspects

What do we want to enforce?

## **standard compliance: c**ode should be written in compliance with a given (set of) standard(s) TBD (2003?). EIRENE is compiled by many users across the world with different compilers. Rather than supporting all of these compilers, we should guarantee standard compliance and avoid compiler dependent features.*We can likely enforce this through suitable compiler flags.*

## **formatting: (after changing to free-form)** We wish to ensure a standard format for indentation, spacing of routine arguments (in call and declaration), ... This ensures that the code is more readable and understandable for developers reading each other’s code.*We need a linter for this, but technology exists.*

## **variable names:** Less of a point in Fortran, but capitalization can still be chosen to follow a style guide when using free-format. Even if the compiler cannot differentiate case, humans can. This way it can be immediately clear whether a name refers to a variable, subroutine, function, …*A linter can probably do this, but it may be more complex. Checking variable names conform to some formatting rules can be specified using e.g. REGEXes. Automated checking of how clear and meaningful a name is, is another matter.*

* **Other best practice:** We should see how possible it is to enforce other coding best pracicesIMPLICIT NONE, INTENT, …
*Here the compilers are probably good tools, using certain warning flags. This does tie developers into a given compiler though, as different compilers have different levels of pedantry.*

# Old vs. new files

Concern was raised during the VC that strict enforcing of the above would lead to major changes being needed in a compilation unit following a bug-fix in existing code. The question is then how to ensure that (if we enforce strict compliance) that we only do so for “new code”. We thus require a definition as to what “new code” is. Some options:

* The most likely approach is to work at the file level. This means either having a list of whitelisted files which are not checked for compliance. This list could be explicit (i.e. a configuration file in git), or implicit (i.e. based on a set of rules).
	1. Can we implicitly do this based on the file extension? E.g., we use extension .f for non-modernized files and .fxx for modernized files.
* As an alternative we could try to be more local. For example, we could enforce standard compliance on any lines that are marked as changed in the git-diff. This is likely not an interesting path for two reasons:
	1. A number of the aspects that we wish to standardize only make sense at a more granular level (at a minimum subroutine level, but possibly even at file level).
	2. This alterative would still force people to make changes to modernize an existing line that they wish to change for e.g. a bugfix. These changes may additionally force changes elsewhere in the code.

# Technical options

Strict compliance can be enforced by rejecting commits on server-side if they do not comply to standards. This is only acceptable if we are sure compliance is possible at reasonable effort. Dmitriy asked for a single user tool to automatically comply before pushing. Likely, we can provide a script using some tool and a config file containing the actual specification that we enforce. The same config file can be used server-side and for IDE plugins to make things consistent. Technical feasibility of having such a config is important in tool selection.

For reference we list a number of linters which could be interesting to check out:

* <https://pypi.org/project/fortran-linter/> : linter
* <https://github.com/pseewald/fprettify> : linter
* <https://sourceforge.net/projects/findent/> : linter
* <https://github.com/fortran-lang/fortls> : plugin for most IDEs
* <https://fortranwiki.org/fortran/show/Tools> : list of tools

We need to investigate which tool to use. Fortls seems to be well supported by many IDEs, so is a logical starting point.

# Next steps

Given available tools, formatting is likely the easiest of the above aspects to enforce automatically. Huw will try to run fortls on some EIRENE files to see what results this produces. We further define the following minimal viable product for automated formatting

* Based on file extension, we know if a file is in free format. (\* The feasibility of this distinction is yet to be assessed.)
* All files in free format should (at the moment of conversion) be run through an automatic formatting tool.
* On server side, all free-format files are checked (not formatted) when code is committed. If the format is not correct, the commit is rejected. (Git hook)
* On client side, developers have a set of tools to automatically comply with formatting specification. We provide a minimal tool in the form of a script which can be run manually in the command line, but we select a tool/configuration file which is as generic as possible so most people can automatically integrate with their IDE of choice.

Points for next discussion:

* Is fortls a good approach and is the MVP above attainable?
* Can we distinguish based on extension?
* Roadmap to MVP
* How to get feedback?

Post MVP points

* Assessment of success
* Extension to other aspects (name checking, standard compliance, …)

For reference, the chat messages from the VC on 28/4

00:27:57 Giovanni Samaey: You might even find that the correct operation is NOT ensuring that the code works the same as it was…

00:29:20 Oskar Lappi: Hi, sorry I'm late, the university decided to lock my user account this morning...

00:35:24 Huw Leggate: Sorry, I need to go for 2 mins

00:55:15 Oskar Lappi: I can't unmute! But the CoC usually doesn't include technical stuff like this

00:55:51 Oskar Lappi: The code of conduct is typically the ethics document

00:56:11 Oskar Lappi: That's normally the contribution guideline

00:56:25 Oskar Lappi: (which is a different document)

00:56:32 Oskar Lappi: just a pedantic comment, sorry

00:57:22 Giovanni Samaey: The main question is where people will look for these instructions

00:58:53 Giovanni Samaey: You can even check that the code compiles with the correct compilers as a commit hook.

00:59:10 Giovanni Samaey: Then users don’t need to have read the document and we still have control

00:59:21 Oskar Lappi: And that is a pretty normal CI case

00:59:26 Oskar Lappi: compiling with supported compilers

01:00:39 Giovanni Samaey: Indentation and style is something you can enforce with software

01:00:48 Giovanni Samaey: Variable names of course is something entirely different

01:03:00 Oskar Lappi: The linter/auto-formatter will eventually make the diffs easier to read

01:03:25 Oskar Lappi: You can use git hooks to refuse commits like this

01:03:50 Xavier Bonnin: Do tell! I'd love to know about them.

01:04:09 Oskar Lappi: It's exactly what Derek and Giovanni are talking about

01:04:26 Xavier Bonnin: How is that done technically?

01:04:46 Oskar Lappi: You configure the git repo to reject according to some rule, the rules typically come as plugins

01:04:54 Giovanni Samaey: https://www.prettifyjs.net/

01:05:25 Oskar Lappi: The git hooks can also be on the local repository, and reject commits locally, but that requires some setup from the user

01:05:34 Xavier Bonnin: Thanks. I'll enquire with our GIT repo managers here about this.

01:05:41 Emil Løvbak: Code style is specified in a config file, so you can enforce it in the IDE at the developer end and also before accepting commits in the repository.

01:05:51 Oskar Lappi: I still can't unmute Dmitry

01:06:08 Emil Løvbak: Once developers use this locally, it requires no effort on their end.

01:06:33 Oskar Lappi: "The host is not allowing participants to unmute themselves"

01:08:31 Oskar Lappi: Should be rejected in the upstream, but give tools to developers to do the autoformatting locally, like Emil describes

01:09:51 Giovanni Samaey: The main point I wanted to make is that we can be very strict on many aspects on formatting, because the argument that we can’t enforce this is not a valid argument.

01:10:40 Oskar Lappi: Most IDE's will typically integrate with linters/autoformatters

01:11:02 Giovanni Samaey: So, I would suggest that we list all formatting requirements we want to enforce, and then take two steps:

- enforce when pushing to repository

- facilitate people to adhere

01:11:19 Oskar Lappi: Here are some fortran linters I found:  https://github.com/pseewald/fprettify https://github.com/cphyc/fortran-linter https://gitlab.com/cerfacs/flint

01:11:30 Oskar Lappi: The standard is not the same as the style

01:12:47 Oskar Lappi: There are typically predefined style standards

01:12:54 Oskar Lappi: And you can usually just select one

01:13:23 Oskar Lappi: For C++, there are google style standard, GNU style standards, etc., etc.

01:13:44 Huw Leggate: Thanks Oskar

01:16:55 Huw Leggate: Sorry, I need to go

01:17:29 Oskar Lappi: here's git and gitlab documentation for server hooks: https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks#\_server\_side\_hooks https://docs.gitlab.com/ee/administration/server\_hooks.html

01:17:36 Oskar Lappi: (How to reject commits)

01:22:52 Oskar Lappi: Yes, we pick one tool and one style config

01:23:07 Oskar Lappi: But luckily, most IDEs can integrate against any linter

01:23:35 Oskar Lappi: You don't \_need\_ to use these static analysis features of a linter

01:24:25 Oskar Lappi: I don't know in Fortran, but in C++, clang-format can check variable name style

01:24:35 Oskar Lappi: It's all part of the AST that a compiler spits out

01:27:05 Oskar Lappi: I would qualify that Dmitry, \_most\_ IDEs integrate with tools, but I can't speak for every single one

01:27:38 Oskar Lappi: Checks lenght, and case style

01:27:58 Oskar Lappi: camelCase vs snake\_case

01:28:29 Oskar Lappi: and 'meaningful\_name' vs 'i'

01:29:23 Oskar Lappi: The tools can still recognize the name

01:29:41 Oskar Lappi: Even if the compiler does not see a difference

01:30:59 Giovanni Samaey: If I can assign people that I trust will come up with rules they would want to include, I would suggest a short-cycle between Hew, Xavier, Emil and Oskar...

01:31:15 Giovanni Samaey: (At least)

01:31:31 Oskar Lappi: I can meet next week

01:33:16 Oskar Lappi: See you!