Whither Fortran Forum?

Brad Richardson and Damian Rouson

1 Motivation

As a newsletter dedicated to the world's oldest programming language, ACM SIGPLAN *Fortran Forum* holds a unique place in the literature. For nearly forty years, *Fortran Forum* has chronicled the language that brings the world its weather and climate predictions (Skamarock et al. 2008; Danabasoglu et al. 2020) and supports the design of power plants (Geelhood et al. 2011) and vehicles that traverse land, air, and space (Cifuentes 2012; Biedron et al. 2019). When *Fortran Forum*, n'ee *ForTec Forum*, published its inaugural issue, FORTRAN was still spelled in upper case and was "the most widely used programming language" according to an article by founding Editor Loren Meissner (Meissner 1982).

Fortran Forum has published a mix of research articles, columns, and news with the most highly cited research articles garnering nearly 1,000 citations since publication (Numrich and Reid 1998).

A key factor in the success of a publication is attracting a steady stream of authors to submit articles. One impediment to this can be the difficulty of the process of writing and preparing an article for submission. It is the hope that by using a simpler format for article content, making use of practices and tools familiar to most programmers, and automating a significant portion of the authoring process - namely spell checking and formatting - we can lower the barrier to entry for authors.

Another possible obstacle to authors can be the review process. ACM Fortran Forum has traditionally not been a peer reviewed journal, but the editor is exploring the possibility of having a peer reviewed option. The editor would prefer that the review process be open (public) as a way of encouraging reviewers and authors to be on good behavior. This also allows prospective new authors to see how the process works and not be turned off by fear of the unknown.

Finally, by having much of the process automated and encouraging authors to utilize the template, it can alleviate some of the burden on the editors for producing new issues of the journal. This can help ensure a regular publication schedule, without which new authors might not be attracted. It can also ensure that the publication adheres to a consistent style and has a quality format, also something beneficial for attracting prospective authors.

2 The Template Format

The article template uses markdown as the format. This was chosen as an approachable, yet powerful format. The syntax is very lightweight, making the content easily readable even unprocessed. This makes it easy to learn, and easy to write in.

Additionally, markdown is well supported by a large variety of tools, meaning authors will not have a difficult time using it with their favorite editors. It also means there are a large number of resources available for learning about it, and for helping to troubleshoot any issues that might be encountered.

Also, because markdown is stored as plain text, it is convenient for version control systems. This makes tracking changes easy, and enables a process of collaboration familiar to our target authors: programmers. The authors of this article have had much success collaborating on projects using markdown and version control.

3 The Template Repository

The authors chose to store this template in a GitHub repository for several reasons. First, this allows the template to be publicly available and thus obtaining a copy is easy. In fact, by utilizing GitHub's Template Repository feature, getting started writing a new article with the template is practically as simple as clicking a button.

Next, storing the template as a GitHub repository makes it easy for users to submit feedback. Anyone can open an issue to report a bug, or suggest changes that should be made. Anyone can also contribute directly to improving the template by submitting a pull request.

Finally, GitHub provides convenient facilities for automation in a git repository. As described later in this article, we make significant use of this feature to check a variety of things. This helps to reduce the burden on both authors and the editors, as well as catch simple mistakes.

4 How the Template Works

Click the "Use This Template" button in the article template repository to create your own copy. You write your article in the provided template file (this file), using markdown format. You can also add additional references to the included `bibliography.bib` file in bibtex format, and then cite those references as demonstrated below. All of this editing can be done in a web browser via GitHub, but if you're familiar with git can be done in whatever method is most convenient for you.

For every commit made to your repository, the provided CI script (`.github/workflows/CI.yml`) will convert your article to a pdf that can be downloaded from the "Actions" tab of your repository on GitHub by clicking on the most recent run and looking for the "Artifacts" section. If you are working on your own machine and have `pandoc` and LaTeXinstalled, you can generate the same preview locally using the command found in the "Render Paper" section of the CI script.

4.1 Some Additional Features

In addition to most of the usual features of markdown, such as headings, bulleted and numbered lists, and hyperlinks, this template supports some features also required for producing quality articles.

4.1.1 Citations

Invariably, any article will need to cite prior work. This is done by including an entry for the reference you would like to cite in the `bibliography.bib` file. Then, within the text of the article, citations are made using rMarkdown syntax, as illustrated below for quick reference.

- `@cifuentes2012using` -> Cifuentes (2012)
- `[@berna1997frapcon]` -> (Berna et al. 1997)
- `[@biedron2019fun3d; @danabasoglu2020community]` -> (Biedron et al. 2019; Danabasoglu et al. 2020)

4.1.2 Figures With Captions and References

Quite frequently it is desirable to include a figure within an article, and refer to it within the text. This can be done similarly to regular markdown, but with some additional syntax as shown below. The results of this can then be seen in Figure 1, with the reference to it generated with the syntax `\autoref{fig:example}`.

```
![The Fortran logo from fortran-lang.org\label{fig:example}](Fortran-logo.png){ width=20% }
```

Figure 1: The Fortran logo from fortran-lang.org

4.1.3 Equations

Another aspect quite frequently used in articles is mathematics. One can use single dollar signs (`$`) to delimit inline mathematics. For example, `$e = m c^2$` will be rendered as $e = mc^2$.

Double dollar signs can be used to make self-standing equations, as illustrated by the syntax below being rendered into the equation that follows.

```
$$
\frac{\partial u}{\partial t}
+ \left( u \cdot \nabla \right) u
- \nu \nabla^2 u
= \nabla w + g
$$
```

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)\, u - \nu \nabla^2 u = \nabla w + g$$

You can also use plain LaTeXfor equations, as illustrated by the syntax below, and rendered into Equation 1, referred to by the sytanx `\autoref{eq:boltzmann}`.

```
\begin{equation}\label{eq:boltzmann}
\left( \frac{\partial}{\partial t}
+ \overrightarrow{v_1} \cdot \nabla_{\overrightarrow{r}}
+ \frac{\overrightarrow{K}}{m} \cdot \nabla_{\overrightarrow{v_1}} \right) f_1
= \int d\Omega \int d\overrightarrow{v_2} \sigma\left(\Omega\right)
\left| \overrightarrow{v_1} - \overrightarrow{v_2} \right|
\left( f_1'f_2' - f_1f_2 \right)
\end{equation}
```

$$\left(\frac{\partial}{\partial t} + \overrightarrow{v_1} \cdot \nabla_{\overrightarrow{r}} + \frac{\overrightarrow{K}}{m} \cdot \nabla_{\overrightarrow{v_1}} \right) f_1 = \int d\Omega \int d\overrightarrow{v_2} \sigma\left(\Omega\right) \left| \overrightarrow{v_1} - \overrightarrow{v_2} \right| \left(f_1' f_2' - f_1 f_2 \right) \tag{1}$$

4.1.4 Source Code

In a journal about a programming language, one invariably needs to include some source code. You can write code inline by enclosing it in single back-ticks like ‘code‘ -> `code`. You can also write multi-line snippets of code directly in the document by including a line of three back-ticks before and after. The result looks something like the following.

```
x = y + z
if (thing) then
    call do_something()
else
    call do_other()
end if
```

While the above methods work and are acceptable, we highly recommend writing your source code in separate files. The CI script will then compile and possibly run, any of your code to ensure it works. We use the Fortran Package Manager to do so. You can then include the file by having an empty code block (i.e. two lines of three back-ticks), but on the first line, after the back-ticks include syntax like {`include=src/library_s.f90`}. We use the external pandoc filter `py-pandoc-include-code`, which also has options for including portions of the named file.

```
module communicator
    use iso_fortran_env, only: team_type
```

```
    implicit none
    private
    public :: communicator_t

    type :: communicator_t
        type(team_type), allocatable :: teams(:, :) !! sender, receiver
    contains
        procedure :: init
        procedure :: send_to
        procedure :: receive_from
    end type
contains
    subroutine init(self)
        class(communicator_t), intent(inout) :: self

        type(team_type) :: dummy_team
        integer :: sender, receiver, team_number, new_image_number

        associate(me => this_image(), ni => num_images())
            allocate(self%teams(ni, ni))
            associate(dummy_team_num => ni**2 + 1)
                do sender = 1, ni
                    do receiver = 1, ni
                        if (sender /= receiver) then
                            if (sender == me .or. receiver == me) then
                                team_number = receiver + (sender-1)*ni
                                if (sender == me) then
                                    new_image_number = 1
                                else
                                    new_image_number = 2
                                end if
                            else
                                team_number = dummy_team_num
                                new_image_number = unique_image_num(sender, receiver, me)
                            end if
                            form team (team_number, dummy_team, new_index = new_image_number)
                            if (sender == me .or. receiver == me) then
                                self%teams(sender, receiver) = dummy_team
                            end if
                        end if
                    end do
                end do
            end associate
        end associate
    contains
        pure function unique_image_num(s, r, m) result(num)
            integer, intent(in) :: s, r, m
            integer :: num

            if (s < m) then
                if (r < m) then
                    num = m - 2
                else
```

```
                num = m - 1
            end if
        else
            if (r < m) then
                num = m - 1
            else
                num = m
            end if
        end if
    end function
end subroutine

subroutine send_to(self, to, payload)
    class(communicator_t), intent(in) :: self
    integer, intent(in) :: to
    class(*), intent(in) :: payload

    call send_receive(self%teams(this_image(), to), payload_in = payload)
end subroutine

subroutine receive_from(self, from, payload)
    class(communicator_t), intent(in) :: self
    integer, intent(in) :: from
    class(*), allocatable, intent(out) :: payload

    call send_receive(self%teams(from, this_image()), payload_out = payload)
end subroutine

subroutine send_receive(team, payload_in, payload_out)
    type(team_type), intent(in) :: team
    class(*), intent(in), optional :: payload_in
    class(*), allocatable, intent(out), optional :: payload_out

    type :: payload_t
        class(*), allocatable :: val
    end type
    type(payload_t) :: payload

    if (present(payload_in)) payload%val = payload_in
    change team (team)
        call co_broadcast(payload, 1)
    end team
    if (present(payload_out)) call move_alloc(payload%val, payload_out)
end subroutine
end module
```

5 Submitting an Article

The process of writing and submitting an article can now become much simpler. No longer will you need to battle with word processor template or fight with the difficulties of LaTeX. Formatting is not something the author must worry about. Just write the content.

To submit an article for publication, follow the instructions described in the previous section to write your article. Once your article is ready to review, contact the editors of the Journal at ???, and make sure that they have access to view the repository with your article. If you elect to have your article peer

reviewed prior to publication, you will also need to grant access to the reviewers. Any review comments will be submitted as GitHub Issues. Once all review comments have been satisfactorily addressed, the editors will be able to include your article in the next issue of the journal.

6 Automated Checks

The template repository comes with several automated checks in place. These checks are executed for any change that is pushed to the repository. The script that executes them is stored in the repository at `.github/workflows/CI.yml`. The script uses only open-source and easily available software, so the same commands can be used locally. The script can also be edited, if some reason an article has some different, specific requirements. Note however than any significant changes to the process of rendering the article to a pdf may present a barrier to having it published.

The first check, and of primary importance is that the paper is rendered to pdf and made available to download for preview. As noted earlier, this preview can be downloaded from the "Actions" tab of your repository on GitHub. This should dramatically reduce the chance of having articles submitted which the editors are unable to render for publication.

Next, a spell checker is used to check the contents of the paper for spelling errors. Any misspelled words are reported. This should help find spelling errors earlier in the writing process, and help reduce the chances that any make it to publication.

Finally, any Fortran source code will be compiled and possibly executed. The Fortran Package Manager (fpm) is used to alleviate the burden on authors of putting together their own build system. See its documentation for the specifics of its use. This should help avoid the publication of source code and examples that contain errors.

References

Berna, GA, GA Beyer, KL Davis, and DD Lanning. 1997. "FRAPCON-3: A Computer Code for the Calculation of Steady-State, Thermal-Mechanical Behavior of Oxide Fuel Rods for High Burnup." Nuclear Regulatory Commission, Washington, DC (United States). Div. of ....

Biedron, Robert T, Jan-Reneé Carlson, Joseph M Derlaga, Peter A Gnoffo, Dana P Hammond, William T Jones, Bil Kleb, et al. 2019. *FUN3D Manual: 13.6*. National Aeronautics; Space Administration, Langley Research Center.

Cifuentes, Arturo O. 2012. *Using MSC/NASTRAN: Statics and Dynamics*. Springer Science & Business Media.

Danabasoglu, Gokhan, J-F Lamarque, J Bacmeister, DA Bailey, AK DuVivier, Jim Edwards, LK Emmons, et al. 2020. "The Community Earth System Model Version 2 (CESM2)." *Journal of Advances in Modeling Earth Systems* 12 (2).

Geelhood, KJ, WG Luscher, CE Beyer, and JM Cuta. 2011. "Fraptran 1.4: A Computer Code for the Transient Analysis of Oxide Fuel Rods." *US Nuclear Regulatory Commission, Office of Nuclear Regulatory Research, NUREG/CR-7023* 1.

Meissner, Loren P. 1982. "The Fortran Programming Language: Recent Developments and a View of the Future." In *ACM SIGPLAN Fortran Forum*, 1:3–8. 1. ACM New York, NY, USA.

Numrich, Robert W, and John Reid. 1998. "Co-Array Fortran for Parallel Programming." In *ACM Sigplan Fortran Forum*, 17:1–31. 2. ACM New York, NY, USA.

Skamarock, William C, Joseph B Klemp, Jimy Dudhia, David O Gill, Dale M Barker, Wei Wang, and Jordan G Powers. 2008. "A Description of the Advanced Research WRF Version 3. NCAR Technical Note-475+ STR."