Juliacon Proceedings in Quarto

Patrick Altmeyer[1]

[1]Delft University of Technology

ABSTRACT

This document is a template demonstrating the `juliacon-proceedings` format. It is a standard Quarto document composed of Markdown, LaTeX and code chunks. It is rendered to PDF and HTML. The PDF is rendered using the `juliacon-proceedings-pdf` format, which is based on the `juliacon-proceedings` LaTeX class. The HTML is rendered using the `juliacon-proceedings-html` format, which is based on the `juliacon-proceedings` HTML template.

Keywords

Template, Demo, Quarto, JuliaCon

1. Introduction

This is a template for writing a JuliaCon Proceedings article in Quarto. This is a proof-of-concept for how we could use Quarto for JuliaCon proceedings. For current submissions, please ignore this repo and follow the official instructions here.

1.1 What is Quarto?

Quarto makes it easy to write reproducible documents that can be rendered to PDF, HTML, Word and more. It is based on Markdown, which is easy to learn and write. It also supports LaTeX, which is useful for more advanced formatting. As this extension demonstrates, Quarto is also very flexible and can be extended with custom templates and styles.

1.2 Why Quarto?

By embracing Quarto, JuliaCon Proceedings can set an example for how to write reproducible documents. We would not only make it easier for authors to write their submissions but also open the door for more advanced features such as interactive figures and executable code blocks in HTML documents.

1.3 About this template

This template is based on the existing JuliaCon Proceedings LaTeX template.

1.4 How to use this template

> The remainder of this document repeats relevant elements from the [JuliaCon Proceedings LaTeX template].

2. The JuliaCon Article Class

The `juliacon` class file preserves the standard LATEX{} interface such that any document that can be produced using the standard LATEX{} article class can also be produced with the class file.

It is likely that the make up will change after file submission. For this reason, we ask you to ignore details such as slightly long lines, page stretching, or figures falling out of synchronization, as these details can be dealt with at a later stage.

Use should be made of symbolic references (`\ref`) in order to protect against late changes of order, etc.

3. Executable Code

Ideally, we would be able to tab into Quarto's existing support for executable, cross-referenceable code chunks.

> To create cross-referenceable code listings from executable code blocks, use `lst-label` and `lst-cap`. — Quarto Docs

For example, the following code block will be labeled as `lst-1` and calling `@lst-1` will render as Listing 1. This ensures that things stay up-to-date and potential errors in the code are identified early: as you render your document, the code will be executed and the output will be inserted into the document.

Listing 1 A listing caption

```
using Plots

x = -3.0:0.01:3.0
y = rand(length(x));
```

Unfortunately, I don't know how to use the special environment that is already defined for Julia in this context.

4. Code Listings

The special environment that is already defined for Julia code can still be used as before.

```
\begin{lstlisting}[
    language = Julia,
    numbers=left,
    label={lst:exmplg},
    caption={Example Code Block.}
]
using Plots

x = -3.0:0.01:3.0
y = rand(length(x))
plot(x, y)
\end{lstlisting}
```

Listing 1: Example Code Block.

```
1 using Plots
2
3 x = -3.0:0.01:3.0
4 y = rand(length(x))
5 plot(x, y)
```