

DaSL Contributor Guide

Scott Chamberlain

Table of contents

Welcome	4
Inspiration	4
1 Help	5
1.1 House Calls	5
1.2 Slack	5
2 Contributing	6
2.1 Code of Conduct	6
2.2 Communication Channels	6
2.3 GitHub Repositories	6
3 Style	7
3.1 Package styler	7
3.2 IDE and Text editor support	7
4 Code review	8
4.1 DaSL interal guidelines	8
4.2 Community guidelines	8
5 Documentation	9
5.1 R	9
5.1.1 DaSL pkgdown template	9
5.2 Python	10
5.3 Guidelines	10
6 Package maintenance	11
6.1 Package versioning	11
6.1.1 Package version numbers	11
6.1.2 Ignore these	12
6.2 Package releases	12
7 Security	13
7.1 Package security	13
7.2 Data	13
7.2.1 Package Data	13

7.2.2	Examples	13
	References	14

Welcome

This book is a resource for anyone in the [Fred Hutch Cancer Center Data Science Lab](#) community contributing to or using software.

The book attempts to cover all important aspects of software development, including how to get involved in software as a user or contributor, code style, code review, package documentation, and more. It includes both internal facing guidelines as well as for any contributions from folks other than DaSL staff.

- Getting help: [Chapter 1](#)
- Contributing guidelines: [Chapter 2](#)
- Style guidelines: [Chapter 3](#)
- Code review guidelines: [Chapter 4](#)
- Package documentation guidelines: [Chapter 5](#)
- Package maintenance: [Chapter 6](#)
- Security: [Chapter 7](#)

Inspiration

Inspiration for this guide is taken in part from:

- [Tidyverse Style Guide](#)
- [rOpenSci Dev Guide](#)

1 Help

Getting help what to put here ... ?

1.1 House Calls

Data House Calls are quick consultations with DaSL staff to get help on data related questions. Fred Hutch staff (and collaborators) can connect with DaSL staff who have expertise on specific topics, or schedule a General Data House Call.

Schedule a house call here <https://calendly.com/data-house-calls>

1.2 Slack

Fred Hutch folks can chat with other community members and DaSL and Fred Hutch SciComp staff in the Fred Hutch Data Slack.

Join our Slack here <https://hutchdatascience.org/joinslack/>

2 Contributing

Contributing guidelines

2.1 Code of Conduct

xxxx

2.2 Communication Channels

- Slack

2.3 GitHub Repositories

xxxx

3 Style

What is code style? It has to do with how you organize your code. The end goal of styling your code is that your code is more consistent. Styling code by definition has to be opinionated. You may not agree with every decision made in a style guide, but having style be done automatically for you gives you more time to think about the important decisions in your code.

If you always write code by yourself and you're the only one that will ever look at it, that's one thing. But that's rarely the case, especially if you consider that yourself 6 months or 5 years from now is a user that is keenly interested in readable, consistent code. Readable, consistent code will be especially appreciated by other users and contributors.

3.1 Package styler

The [styler](#) package allows you to interactively format your code according to the [tidyverse style guide](#).

The [lintr](#) package does automated checks of your code according to the style guide.

3.2 IDE and Text editor support

RStudio supports styler via the Addins drop down; see the [RStudio User Guide](#).

Support for `styler` in other editors is provided via the [R languageserver](#):

- VSCode: [vscode-R](#)
- Atom: [atom-ide-r](#)
- Sublime Text: [R-IDE](#)
- Vim/NeoVim: [LanguageClient-neovim](#)

See the [R languageserver](#) GitHub repository for more information on using the R language-server.

4 Code review

Code review

4.1 DaSL interal guidelines

XXXX

4.2 Community guidelines

XXXX

5 Documentation

All formally supported DaSL R and Python packages should have package documentation.

5.1 R

We use the [pkgdown](#) package to create documentation for R packages, and host it on [GitHub Pages](#). To get started with `pkgdown`, in R within the root of your package run `usethis::use_pkgdown_github_pages()` - it will set up a `_pkgdown.yml` file in the root of your repo used to configure `pkgdown`, and add a `.github/workflows/publish.yml` file to build the package documentation on each push, pull request, or release. See [pkgdown documentation](#) for details on configuring documentation.

After pushing the above changes up to your repository, go to the Actions tab and you should see the new `publish` action running. It will build and then deploy the rendered package docs. The URL for your docs will vary depending on the GitHub organization your repository lives within. For WILDS, the base URL for now is <https://getwilds.github.io>. If your package was in the WILDS org (at <https://github.com/getwilds>) your package (named `mypkg`) docs would live at <https://getwilds.github.io/mypkg>.

You can also build `pkgdown` docs locally - after running `usethis::use_pkgdown()` or `usethis::use_pkgdown_github_pages()` - by running `pkgdown::build_site()`. If you run `build_site()` within RStudio it should open up your site in your default browser, but may not do so if you run in a terminal. You can open the site in your browser by navigating to and opening the file `docs/index.html` within your repo.

5.1.1 DaSL pkgdown template

We are planning to have a DaSL specific `pkgdown` “package template” (see [pkgdown docs](#) for what this means) - but it’s not ready to use yet. When it is ready, you will be able to specify our template like:

```
template:  
  package: dasltemplate
```

For now just use the default theme that `pkgdown` provides.

5.2 Python

Just like there's a variety of ways to do packing in Python there's a variety of documentation options. Two of the well known options are:

- [Sphinx](#) - been around longer, uses [reStructuredText](#)
- [MkDocs](#) - newcomer, used [Markdown](#)

For either of the options above, they can be hosted in many places, including [GitHub Pages](#) and [ReadTheDocs](#).

`Sphinx` and `MkDocs` are less automatic relative to `pkgdown`, so just be prepared for a bit more manual work.

5.3 Guidelines

- README: This is most likely the first place potential users will interact with your package. Make sure the README clearly states what the package does, and how to get started.
- Examples: All user facing functions should have examples. Make sure to be careful about how examples are run if there's any sensitive data or connections to remote services.
- (anything else?)

6 Package maintenance

6.1 Package versioning

There is a detailed discussion of versioning R packages in the [lifecycle](#) chapter of the [R Packages book](#) by Hadley Wickham and Jenny Bryan. Please follow that chapter in general for versioning of R and Python packages within DaSL. To make it easier to grok, below are some of the highlights, and some exceptions to that chapter.

6.1.1 Package version numbers

There's quite a bit of nuance - and surprises - to package version numbers - see the [Package version number section](#) for details. For example, using the `utils::package_version()` function, which parses package version strings into S3 classes, we get a surprising result:

```
"2.0" > "10.0"
#> [1] TRUE
package_version("2.0") > package_version("10.0")
#> [1] FALSE
```

With that example, please do think about your package versions before setting them.

6.1.1.1 DaSL Conventions

Following the [Tidyverse package version conventions](#), DaSL packages will use the following conventions (see the link for more details):

- Always use `.` as the separator, never use `-`.
- A released version number consists of three components, `<major>.<minor>.<patch>`
- While a package is in between releases, there is a fourth component: the development version, starting at 9000 (e.g., `0.2.2.9000`), and incrementing from there until the package has another release at which point return to three components.

6.1.2 Ignore these

We are not following or enforcing any rules about changes at the function/class/etc level below the package level. For example, the R Packages book [talks about](#) using the `lifecycle` package to deal with function changes.

6.2 Package releases

In general follow the [Releasing to CRAN chapter](#) in the book [R Packages](#) for R, and the [Releasing and versioning chapter](#) in the book [Python Packages](#) for Python. Those chapters don't have to be followed to the letter, but in general they provide really good guidance.

There are a few aspects of releases we are opinionated about and would like DaSL R and Python packages to follow:

- Follow our versioning guidelines above
- Git tag released versions, and push the tag to GitHub
- Add the associated NEWS/Changelog items to a release associated with the tag on GitHub

7 Security

7.1 Package security

Keeping secrets a secret is an important task and should not be an afterthought - but instead considered from the beginning of a package.

In general, the rOpenSci Development Guide has good advice on [package security](#).

As time goes on we'll document here if we have any strong opinions about security related aspects of packages.

7.2 Data

Software created in the WILDS may touch PII/PHI data.

7.2.1 Package Data

Do not include any PII/PHI data within a package. If data needs to be included in a package for tests, examples, etc., use fake data that is similar enough to be able to run tests/examples, etc.

There's many packages in both R (e.g. [charlatan](#), [wakefield](#), [fakir](#)) and Python (e.g. [faker](#)) for creating fake data.

7.2.2 Examples

We strongly encourage all exported, user-facing functions, to have examples. However, when PII/PHI data is a crucial part of an example for a function/class/etc. we need to take extra precautions.

In package documentation created with pkgdown (see Chapter 5) examples should use fake data included with the package.

If it makes sense to have examples that run with real data that might be PII/PHI data, those examples should xxxx

References