Building Data Packages

A guide to engineering and developing research data as a FAIR and tidy data package

Luke W. Johnston

Contents

Welcome!

> **Caution**
>
> These documents are very much a work in progress and very much incomplete. We work on it slowly and when we can.

There are little to no resources available online on how to develop data as a package similar to developing software as a package, e.g. R packages, Python packages, or Rust packages. There are many resources on how to build data warehouses or data lakes/lakehouses, along with many enterprise-level tools and platforms to build them. However, these are often overengineered and excessively complicated for the needs within science and research. This guide is an attempt at providing a practical how-to guide on treating data with a "package" mindset. We walk through how to build up data following practices from the software package development cycle so that the final "data package" (or "data product") is findable, accessible, interoperable, and reusable (FAIR) and tidy.

> **Note**
>
> This guide mostly follows the diátaxis "how-to guide" style, though not strictly. It is a living and constantly evolving guide that is regularly updated as we learn and refine how we work and develop data packages. We intend to continually update and release it with every update to Zenodo and as GitHub releases. We don't expect this guide to ever be considered "done".

Who you are

We've written these documents with a few types of people in mind:

- **New contributors/team members**: This is our primary audience for this guide. If you are new to working with us on the Seedcase Project, these documents are designed with you in mind.
- **Research software/data engineers**: If you work in another organization or group, we write these documents to share our practices with you. You can use this either as a guide or as a reference to help you build your own data packages and learn how we work.

Contributing

Check out our contributing document for information on how to contribute to this guide.

Changes

Check out our changelog for information on what has changed in each version of the guide. Previous versions can be found in the releases page of the GitHub repository.

How the website is made

The website and PDF (download link on the top of the website's sidebar) are created using Quarto to write the material and create the book format. GitHub hosts the Git repository of the material, while GitHub Actions builds and releases the book with every change. Netlify hosts the website and manages the domain. The source material is in the `data-pkg-guide` GitHub repository.

I Overview

1. Introduction

2. Package layout

3. Overall workflow

II Structure

4. Source code

5. Raw data

6. Staging data

7. Data resources

8. Metadata

III Lifecycle

9. Continuous integration

10. Build process

11. Release process

IV Development

12. Tools

13. Setup

14. Configurations

15. Testing

16. Developer experience

V Management

17. License

18. Requests for data

19. Subsets of data

Appendices

A.

B. Contributing

B.1 Issues and bugs

The easiest way to contribute is to report issues or bugs that you might find while reading through the website. You can do this by creating a new issue on our GitHub repository.

B.2 Adding or modifying content

If you would like to contribute content, please check out our guidebook for more specific details on how we work and develop. It is a regularly evolving document, so is at various states of completion.

To contribute to `data-pkg-guide`, you first need to install uv and justfile. We use uv and justfile to manage our project, such as to run checks and test the template. Both the uv and justfile websites have a more detailed guide on using uv, but below are some simple instructions to get you started.

It's easiest to first install uv and then install justfile with uv. Once you've installed uv, install justfile by running:

```
uv tool install rust-just
```

We keep all our development workflows in the `justfile`, so you can explore it to see what commands are available. To see a list of commands available, run:

```
just
```

As you contribute, make sure your changes will pass our tests by opening a terminal so that the working directory is the root of this project's repository and running:

```
just run-all
```

When committing changes, please try to follow Conventional Commits as Git messages. Using this convention allows us to be able to automatically create a release based on the commit message by using Cocogitto. If you don't use Conventional Commits when making a commit, we will revise the pull request title to follow that format. That's because we squash merge when merging pull requests, so all other commits in the pull request will be squashed into one commit.

B.3 Explanation of files and folders

This is a description of some of the files in this repository.

- `.copier-answers.yml`: Contains the answers you gave when copying the project from the template. **You should not modify this file directly.**
- `.github/`: Contains GitHub-specific files, such as issue and pull request templates, workflows, dependabot configuration, pull request templates, and a CODEOWNERS file.
- `_quarto.yml`: Quarto configuration file for the website, including settings for the website, such as the theme, navigation, and other options.
- `_metadata.yml`: Quarto metadata file for the website, including information about the project, such as the titles and GitHub names.
- `.gitignore`: This ignore file tells Git which files to not track. Unless you know what you are doing, it's best to not touch this file.
- `.pre-commit-config.yaml`: Pre-commit configuration file for managing and running checks before each commit.
- `.config/`: Contains configuration files for various tools used in the project, such as:
  - `typos.toml`: typos spell checker configuration file.
  - `rumdl.toml` and `panache.toml`: rumdl and Panache configuration file for formatting Markdown files in the project.

  - `cog.toml`: Cocogitto configuration file for managing versions.
  - `cliff.toml`: git-cliff configuration file for creating the changelog.
- `.editorconfig`: Editor configuration file for EditorConfig to maintain consistent coding styles across different editors and IDEs.
- `.zenodo.json`: Structured citation metadata for your project when archived on Zenodo. This is used to add the metadata to Zenodo when a GitHub release has been uploaded to Zenodo.
- `justfile`: `just` configuration file for scripting project tasks.
- `CHANGELOG.md`: Changelog file for tracking changes in the project.
- `CONTRIBUTING.md`: Guidelines for contributing to the project.

C. Contributor Code of Conduct

As contributors and maintainers of this project, we pledge to respect all people who contribute through reporting issues, posting suggestions, updating any material, submitting pull requests, and other activities.

We are committed to making participation in this project a harassment-free experience for everyone, regardless of level of experience, gender, gender identity and expression, sexual orientation, disability, personal appearance, body size, race, ethnicity, age, or religion.

Examples of unacceptable behavior by participants include the use of sexual language or imagery, derogatory comments or personal attacks, trolling, public or private harassment, insults, or other unprofessional conduct.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct. Project maintainers who do not follow the Code of Conduct may be removed from the project team.

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by opening an issue or contacting one or more of the project maintainers.

This Code of Conduct is adapted from the Contributor Covenant (https://www.contributor-covenant.org), version 1.0.0, available at https://www.contributor-covenant.org/version/1/0/0/