

ALLIANCE SOFTWARE MANAGEMENT PLAN (SMP) TEMPLATE

NOTE:

This is the Alliance Software Management Plan (SMP) Template (version 1) consisting of 18 questions, including 13 mandatory questions (marked with *) and 5 optional questions.

Glossary

Software: Software in this SMP template refers to two categories: 1). software tools (e.g., software programs, languages, libraries, scripts, computational code, models, electronic lab notebooks, repository software, workflow management tools, etc.); and 2). software platforms (variously referred to as research infrastructures, virtual science labs, virtual research environments (VREs), or Science Gateways, etc.).

Software Versioning: In software development, software versioning is a type of numbering or naming scheme that helps software development teams keep track of changes they make to software project code by identifying successive versions. The changes may include new functions, features, or bug fixes. Occasionally, entirely new functions and features are released based on developments across multiple versions. As such, it assists in the creation and management of multiple software product releases. Modern computer software is often tracked using two different software versioning schemes: 1). an *internal* version number that may be incremented many times in a single day, such as a revision control number, and 2). a *release* version that typically changes far less often, such as [semantic versioning](#) or a project code name.

Version Control: Version control (also known as source control, revision control) is the practice of tracking and managing changes to software code over time. Version control is also a way to ensure efficient and collaborative code sharing and editing among multiple developers on different versions of the software at any given time within the larger system. Version control systems (VCS), sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System), are software tools that help software teams manage changes to source code over time.

Software Release Life Cycle (SRLC): The Software release life cycle (SRLC) is a set of milestones that describe various stages in a piece of software's sequential release timeline, from its conception to its public distribution. It typically consists of several stages, such as pre-alpha (referring to the early stage of development where the software is still in its design and development phase), alpha (which represents the first formal testing phase using internal resources), beta (during which it's tested by a larger group of users outside of the organization to find and fix potential bugs or issues), release candidate (for further refinement and testing), and production release (also called stable release, marking the stable and complete version of the product ready for use by end-users). Once released, depending on the method of release, there are stages such as, release to manufacturing (RTM, also known as "going gold", when a software product is ready to be

delivered), general availability (GA, is a marketing stage when it becomes available for purchase), and release to the web (RTW, or web release, is a means of software delivery utilizing the internet for distribution).

Purpose

*** Provide a brief description of your software, stating its purpose, intended audience, and the problem it solves.**

This File Tree Generator is a desktop application developed in Python using the PyQt5 framework. It helps users visualize, annotate, and export directory structures using a user-friendly interface. The left side displays an interactive file tree with readable file sizes and the ability to attach descriptive notes to any folder or file. The right side provides a live Markdown preview of that same tree. Users can export their annotated file hierarchy in either Markdown or plain-text formats, making it suitable to incorporate directory documentation directly into reports, wikis, or publications.

This tool is designed for researchers, data curators, software developers, or RDM practitioners who routinely need to describe complex file arrangements in a shareable format. By automating the collection of file paths and sizes and customizable descriptions, this app eliminates error-prone process of manually writing out directory contents.

Documentation and Metadata

Please describe the architecture of the software (platforms). Are there some high-level principles that will be or were used as an inspiration for the design of the software (platforms)?

This app follows a modular architecture built on Python and the Qt framework to ensure cross-platform compatibility. The application separates file-system interaction, data representation, and user interface into distinct components. The “view” layer is constructed entirely in PyQt5, combining widgets such as QTreeView, QTextEdit and QSplitter to present an interactive tree alongside a live Markdown representation. A controller logic ties these layers together, responding to user actions: directory selection, description edits, and export commands and updating, both the visual tree and its markdown in real time.

From a design perspective, the app mimics established desktop paradigms, such as integrated development environments and file managers, that favor split-pane layouts for simultaneous navigation and content preview. By adhering to the Qt model-view architecture, the File Tree Generator achieves a balance between maintainability, performance, and a user-centered interface.

*** Please describe how your documentation supports users and developers of your software (platforms). Provide links to existing documentation if available, including any documentation best practices you follow.**

From the moment a new user visits the File Tree Generator repository, they are guided by a comprehensive README.md (https://github.com/Alliance-RDM-GDR/RDM_FileTree) that showcases the usage information. The document presents an overview of the application's purpose and features, followed by installation instructions and platform requirements. Also, the README file presents the workflow a user can follow: selecting directories, filtering and searching, annotating files, and exporting to Markdown or plain text.

The code itself is documented with docstrings for public classes and methods, and comments clarify non-obvious steps. By hosting all documentation in Markdown alongside the source code, version control via GitHub ensures that every change to functionality is reflected in the documentation history, and GitHub's native rendering makes these resources immediately accessible online.

*** How will you make sure that documentation is created or captured consistently throughout your project?**

To ensure that documentation is created and captured consistently, we use GitHub for version control and posting updates.

Testing and Deployment Strategy

Please describe or outline how you intend to test and deploy your software (platform) to ensure it: (1) meets the requirements that guided its design and development, (2) is usable and performs its intended function/s, and (3) can be installed and run in its intended environment.

During development, every change or feature is accompanied by test that validate core logic such as directory traversal, size computation, annotation persistence, and Markdown generation against clearly defined acceptance criteria. In parallel, GUI behaviors like selecting folders, filtering, editing descriptions, exporting, are exercised through pytest-qt scripts, simulating end-user interactions to verify responsiveness and usability. These tests run automatically on each pull request and merge via GitHub. We also compile the application for both windows and Mac. PyInstaller packages the application into standalone executables for each operating system. Once the usability is verified, the files are released on GitHub and archived on Zenodo alongside pinned requirements.txt and environment.yml files for reproducibility.

Version Control & Release Management

*** How do you plan to manage [versioning](#) for your software? What are specific functionalities provided by the [version control](#) system/platform?**

We manage source code with Git, hosting the repository on GitHub to take full advantage of version control, collaboration workflows, and releases. From day one, we adhere to Semantic Versioning, incrementing the

major version for backwards-incompatible changes, and the minor version for backwards-compatible feature additions. Each time we prepare a public release, a Git tag corresponding to the new version (for example, v1.2.0). Within GitHub, we implement branch protection so all changes must arrive via pull requests. Every pull request is revised before merging. We also use GitHub Issues to track feature requests and bugs that will drive future version increments.

What strategies/high-level principles/mechanisms/practices/tools do you plan to utilize for your [software release life cycle \(SRLC\)](#)?

Our release life cycle is shaped by collaborative planning, where each new feature request or bug report is triaged in GitHub issues. Development occurs on short-lived feature branches, where contributors implement changes, update associated documentation, and write or extend tests. Continuous integration pipelines validate every commit through GUI tests, linting checks, and dependency audits. Once a feature is complete and test pass, the changes are merged into main and a release candidate is built using PyInstaller (for executables). Post-release, we monitor issue activity for patches.

Sharing and Reuse

*** Describe how you will make your software publicly available during and after development/upgrading, including through repositories like GitHub, Zenodo, or Figshare. Provide a rationale if open access is not feasible.**

The File Tree Generator will be made openly accessible via GitHub, where each commit, issue, and pull request is visible to the community; contributors can fork the project, submit changes, and track progress in real time. Upon tagging a new release, GitHub Actions will automatically package the application and publish a release entry containing executables, source archives, and release notes. To provide a citable, every major release on Zenodo, obtaining a DOI that can be referenced in scholarly publications and data management plans.

*** Specify the type of license for your software, preferably an open license. If not open, provide a justification. Consider hybrid licensing for software with multiple components.**

The File Tree Generator is distributed under the MIT License, an OSI-approved open-source license that grants users the freedom to use, copy, modify, merge, publish, distribute, sublicense, and sell the software without restriction, provided that the original copyright notice and license text are included in all copies or substantial portions of the software.

*** What steps will be taken to help the research community know that your software exists?**

To ensure that the research community becomes aware of the File Tree Generator, we will publish detailed announcements on the Digital Research Alliance of Canada's LinkedIn account, highlighting new capabilities and

linking directly to the GitHub release notes. In parallel, we will submit the tool for inclusion in prominent software registries such as bio.tools, making it discoverable to those searching for “file tree” or “directory documentation” utilities. We also plan to send notification via our list-serv mailing list so all the RDM practitioners in Canada become aware of the tool.

*** How will users of your software be able to cite your software? Please provide a link to your software citation file (CFF) if available. (<https://citation-file-format.github.io/>)**

To facilitate proper attribution, the File Tree Generator repository includes a machine-readable CITATION.cff file at the project root (https://github.com/Alliance-RDM-GDR/RDM_FileTree/CITATION.cff). This file follows the Citation File Format standard and contains metadata such as the software title, version number, authors, DOI, and project URL. Users can incorporate the contents of CITATION.cff into their reference lists or use it with citation tools. For each major release archived on Zenodo, the DOI issued (e.g., 10.5281/zenodo.xxxxxxx) is also included in the citation metadata.

Preservation and Long-Term Maintenance

*** How and where will your software be archived, after the software is developed?**

After each major release, the File Tree Generator’s source code and executables will be secured on GitHub, with every commit, tag, and release retained indefinitely under the project’s open-source license.

*** How do you plan to support long-term maintenance of your software?**

Ensuring the File Tree Generator remains reliable, Daniel Manrique-Castano, currently research data curator at the Digital Research Alliance of Canada will take the responsibility of monitoring issue queues, triaging bug reports, and reviewing community pull requests. Second, a clear contribution guidelines and a well-documented codebase lower the barrier for new volunteer contributors; we actively encourage community members to take ownership.

If using an existing software component/platform, how would you deal with upgrades / patches to third-party software packages that you might use?

Because the File Tree Generator relies on third-party libraries (i.e PyQt5 and humanize), we maintain monitor dependency upgrades to balance stability and security. All required packages are listed with exact version pins in requirements.txt (and optionally in an environment.yml for Conda users), ensuring that every developer or user is able to works from the same reproducible environment. When monitoring reveals a major version upgrade, we first review the library’s changelog to understand any behavioral shifts. We then create a dedicated “dependency-upgrade” branch in which we update the version constraint, adjust our code where necessary, and re-run our tests.

User Support

*** After the software/research software platform has been developed, please include plans for any support-related activities (e.g., platform operations, providing user support, extending / adding functionality, facilitating adoption by new research teams, etc.) in terms of training, support staff allocation, communication channels.**

<i>Software webpage</i>	https://github.com/Alliance-RDM-GDR/RDM_FileTree/
<i>Software documentation</i>	https://github.com/Alliance-RDM-GDR/RDM_FileTree/README.md
<i>Software tutorials</i>	Alliance YouTube channel

Responsibilities and Resources

What resources will you require to implement your software management plan? What do you estimate the overall cost for software management to be?

Implementing and sustaining the Software Management Plan for the File Tree Generator requires light human and technical resources. We anticipate building new features, maintain dependencies, correct bugs and issues. Being open-source in GitHub facilitates the task by the curation team or other RDM practitioners.

*** How will responsibilities for managing software activities be handled if substantive changes happen in the personnel overseeing the project's software, including a change of Principal Investigator? Please consider the situation for managing software both during and after the project.**

In the event that personnel change, we established a succession process to ensure uninterrupted stewardship of the File Tree Generator. All roles and responsibilities are recorded in our CONTRIBUTING.md, including primary maintainers and at least two secondary contacts who have full repository and release-management privileges.

Other Concerns

*** Describe the main external factors that should be considered by developers and users of the software. These could include any security-related information or concerns.**

Developers and users of the File Tree Generator must be aware of external factors that can influence the application's behavior and its safe operation. Because the software directly inspects the local file system, differences in operating system semantics, such as path separators, case sensitivity, or symbolic links can

affect how directories are displayed. Performance constraints also play a role. On very large file systems or slow network shares, recursive scanning can become time-consuming, so users should be aware of potential delays and consider excluding massive subtrees or hidden system folders when appropriate. From a security standpoint, the application deliberately avoids executing or modifying files, but it must still guard against malformed directory structures or unexpectedly long path names that could trigger resource exhaustion. Finally, organizational IT policies, like antivirus scanners, endpoint protections, and corporate firewalls can influence whether the application can access certain drives or launch external links.

Date and Sign

The authors of this document will ensure that this Software Management Plan is carried out as specified above.

Name: Daniel Manrique-Castano

Affiliation: Digital Research Alliance of Canada

Date: 2025-07-23

Signature:

A handwritten signature in black ink, appearing to be 'DMC', written over a light gray grid background.