PharmaSUG China 2026 - Paper AA-158

A Desktop Application for Automating PDT Management, SAS Execution, and Report Assembly

Mingjie Fu, Jiangsu Hengrui Pharmaceuticals Co., Ltd.

ABSTRACT

Statistical programming teams track the development and validation of every table, listing, figure, and analysis dataset in a Program Development Tracking (PDT) workbook stored on a shared network drive. As studies grow, this single Excel file becomes a bottleneck: programmers scroll through hundreds of rows to find their own assignments, edits collide when several people open the workbook at once, and there is no automatic way to tell whether an output is still current with the raw data it depends on. This paper describes PDT Manager, a Windows desktop application built with Python and PySide6 that wraps the existing PDT workbook in a faster, safer interface. The tool reads the workbook into a local cache, lets programmers filter to their own tasks in one click, and writes edits back to the original Excel file through a controlled save path that detects concurrent modification before overwriting. Beyond tracking, it integrates two operational steps that previously required leaving the workbook: it submits SAS programs to a remote server through SASpy and parses the returned log for errors, warnings, and PROC COMPARE results; and it assembles individual output files into a single combined report with a generated cover page and table of contents. A freshness indicator on each row compares output and source file timestamps and propagates a "stale" state along a user-defined dependency graph, so a change in an upstream SDTM dataset visibly flags every downstream output that needs to be rerun. The application has been used on production studies at the author's organization. This paper focuses on the design decisions that made the tool safe to use against a live shared workbook and the techniques that other teams can reuse for similar tracking-and-automation problems.

Keywords: statistical programming, Python, PySide6, SAS automation, workflow tracking

INTRODUCTION

In a typical clinical statistical programming deliverable, every output a team produces is recorded in a tracking workbook. At the author's organization this is called the Program Development Tracking workbook, or PDT. Each row describes one deliverable: its identifier, type (SDTM, ADaM, table, listing, or figure), title, the development program and its author, the development status, the validation level, the QC program and reviewer, and the QC status. The workbook lives in a fixed location on a shared drive within each study's folder structure, and it is the single source of truth for who is doing what and how far along they are.

This arrangement works, but it scales poorly. A late-phase study can carry several hundred rows. A programmer who wants to see only the outputs assigned to them, in progress, has to apply Excel filters by hand each time they open the file. Because the workbook is shared, two people editing at once risk overwriting each other unless the file is put into Excel's shared-workbook mode, which brings its own problems. And the workbook records intent, not reality: it says a program is "Ready for QC," but it cannot tell the programmer that the ADaM dataset feeding that table was regenerated yesterday and the table is now out of date.

Three tasks that surround the workbook are also disconnected from it. To run a development or QC program, the programmer leaves the tracker, opens a SAS session, locates the program on the server, submits it, and reads the log. To check whether an output is current, they manually compare file dates. To produce a combined report for a submission package, they open each output file and paste it into a master document. Each of these is routine, repetitive, and a source of error.

PDT Manager addresses these problems in one application. It does not replace the PDT workbook; the workbook remains the system of record, and all edits are written back to it. Instead, the tool provides a responsive interface over a cached copy of the workbook, adds one-click filtering for common views such as "my development tasks" and "my pending QC," and folds the three surrounding tasks (running SAS, checking freshness, and assembling reports) into the same window. This paper describes how the tool is

built, with emphasis on the parts that are reusable: a cache-and-write-back strategy that keeps a shared Excel file safe under concurrent use, a timestamp-based freshness indicator that propagates along a dependency graph, and a remote SAS execution layer that returns parsed, navigable logs. The intended audience is statistical programmers and programming leads who maintain similar trackers and want to reduce the manual work around them.

THE PDT WORKBOOK AND THE CACHING MODEL

The PDT workbook contains three sheets the application reads: a Deliverables sheet with one row per output, a Users sheet that maps a person's login ID to their display name, and a List Values sheet of controlled vocabularies. The Deliverables sheet carries two header rows above the data, so the application skips them when loading and keeps only rows that have a non-empty output identifier.

Reading a multi-hundred-row workbook directly from a network drive on every interaction would be slow. PDT Manager loads each workbook once and stores the parsed data as a local cache file under the user's application data folder. A small metadata file records, for each cached study, the source path and the source file's last-modified time. When the user selects a study, the application checks whether a valid cache exists by comparing the stored modification time against the file on disk; if they match within one second, the cache is used and the interface populates immediately. If the source is newer, the cache is treated as expired and the user is asked whether to refresh from the server.

This caching model is what makes the interface feel fast, but it introduces the central design risk of the whole application: the cached copy and the live workbook can drift apart, and an uncontrolled write-back would silently overwrite someone else's work. The next section describes how saving is handled to prevent that.

The application discovers studies by walking the shared-drive folder structure rather than asking the user to browse to files. Study folders follow a fixed convention (product, then study, then a stage directory such as `csr_1` or `dsur_1`), and the PDT file sits at a predictable path within each stage. The left panel of the window presents this as a lazy-loaded tree: products load first, and a product's studies are scanned only when it is expanded, so opening the application does not block on a full-drive walk.

SAVING SAFELY AGAINST A SHARED WORKBOOK

Writing edits back to a workbook that other people may have open is the hardest part of the design. PDT Manager treats every save as a potential conflict and checks for one before writing.

When the user requests a save, the application first re-reads the source file's modification time and compares it against the time recorded when the cache was built. If the file has been modified since, the save is refused and the user is told to refresh first. This is optimistic concurrency control: edits are allowed to proceed in parallel, and the conflict is caught at the moment of writing rather than by locking the file for the duration of an editing session.

Only the cells the user actually changed are written. The application keeps an unmodified copy of the loaded data and, at save time, computes a per-row, per-column difference restricted to the editable columns. The result is a small map of row index to changed values, which keeps the write surface minimal and avoids rewriting formatting or untouched cells.

The write itself is performed through Excel's own COM automation, via the `xlwings` library, rather than by rewriting the file with a Python Excel writer. This preserves the workbook's formatting, formulas, and validation exactly, because Excel is doing the writing. Two cases are handled separately. If the workbook is already open in Excel on the user's machine, the application locates that live instance and writes directly into it. If it is not open, the application launches a hidden Excel instance, opens the file, writes, saves, and then terminates that instance. Cell locations are resolved by reading the header row and mapping column names to column indices, so the save does not depend on column order.

Saving runs on a background thread so the interface stays responsive, and the result is reported back when it completes. Common failure modes are translated into actionable messages: a file locked for exclusive editing, a workbook open in edit mode with an uncommitted cell, or a network interruption each

produce a specific instruction rather than a raw exception. After a successful save, the cache is refreshed so the local copy and the workbook stay aligned.

FRESHNESS INDICATORS AND DEPENDENCY PROPAGATION

A tracking workbook records that an output exists, but not whether it is still valid. An output produced last week is stale if the dataset it reads from was regenerated yesterday. PDT Manager surfaces this directly with a colored indicator on each row.

The first layer of the check is a direct timestamp comparison. For an output whose source files have been mapped (the user points the tool at the raw data files an output depends on), the application compares the most recent source modification time against the output file's modification time. If any source is newer than the output, the indicator is red; if the output is newer than all of its sources, it is green; if no sources have been mapped, it is gray. Clicking the indicator shows the per-file comparison so the programmer can see exactly which source triggered a red state.

The second layer propagates staleness along a dependency graph. Outputs are not independent: an ADaM dataset depends on SDTM datasets, and a table depends on ADaM. The application lets the user draw these dependencies in a workflow editor, with nodes grouped into SDTM, ADaM, and TFL layers, and stores the resulting edges per study. Given the direct freshness of each node and the dependency edges, the tool computes a transitive freshness: a node becomes red not only when its own source is stale, but also when any upstream node is red, or when an upstream node's output is newer than the node's own output. The reason for each propagated red state is retained and shown on click, distinguishing "an upstream node is red" from "an upstream output is newer than this one."

Computing freshness touches the file system for every output and so runs on a background thread, refreshing on a timer and after any change to the source mappings or the dependency graph. The dependency edges can also be inferred automatically: when a SAS program runs, its log names the datasets it read, and the application can parse those references to propose upstream links, reducing the manual drawing the user has to do.

RUNNING SAS PROGRAMS FROM THE TRACKER

Development and QC programs live on a remote SAS server, and the deliverable row already records each program's name and directory. PDT Manager uses this to run programs without leaving the tracker.

The path on the shared Windows drive is translated to the equivalent path on the SAS server, and the development-versus-QC distinction is used to resolve whether the program is in the development or validation directory. A SAS session is opened through SASpy, the program is submitted with the team's macro library on the autocall path, and the returned log is scanned for errors, warnings, and notes. The result is summarized (for example, the count of errors and any PROC COMPARE outcome) and the full log is shown with syntax highlighting and clickable navigation between error, warning, and flagged-note lines.

Log review is more than counting `ERROR:` lines. The application classifies log lines against a list of known SAS pitfalls, including generated missing values, uninitialized variables, and implicit numeric-to-character conversions, with an optional fuller set of note-level checks the user can enable. PROC COMPARE blocks in the HTML results are parsed into a pass or fail verdict per comparison, with the specific reason (unequal values, differing observation counts, missing variables) extracted and shown as a badge, so a programmer can confirm a QC comparison passed without reading the entire output.

Programs can be run one at a time, in parallel across several SAS sessions, or in a fixed sequence ordered by analysis target (formats, then data, then the efficacy, safety, PK/PD, and statistics outputs). The sequential mode is intended for regenerating a study's outputs in dependency order for a final run.

ASSEMBLING COMBINED REPORTS

Submission packages often require a single document that combines many individual outputs behind one cover page and table of contents. Before this tool, the team produced that document with a SAS macro in which the outputs to combine were listed by hand, one call per output:

```
%rtf_combined(A1);
%rtf_combined(A2);
%rtf_combined(A3);
```

This works for a handful of outputs but becomes hard to manage as the list grows: the programmer maintains the call list by hand, keeps it in the right order, and edits it every time the set of outputs changes. PDT Manager replaces the hand-maintained list with direct selection. The programmer selects the deliverable rows to combine in the grid, and the tool assembles the file from that selection.

Each selected output's RTF file is read, its body extracted, and the bodies are concatenated under a generated cover page and a hyperlinked table of contents. The cover page draws its protocol title, sponsor, and language from the study's setup file when available, and the report can be produced in Chinese or English. Page numbering within each output is rewritten so the combined document reads continuously, and after the file is written the application reopens it through Word automation to recompute the true page of each bookmark and correct the table-of-contents page references. The combined report can additionally be converted to PDF or DOCX. Parsing of the individual files runs in parallel to keep assembly fast for large packages.

CONCLUSION

PDT Manager shows that a tracking workbook does not have to be replaced to be made more usable. By keeping the PDT workbook as the system of record and building a cached, write-back interface over it, the tool gives programmers a fast view of their own work while preserving the formatting and shared access that the team already relies on. The design decisions that made this safe are reusable: optimistic concurrency that checks for a conflict at write time, a minimal diff so only changed cells are written, and Excel's own automation so the file is never reformatted by an external writer.

The features layered on top of tracking address the work that surrounds it. The freshness indicator turns an implicit "is this still current" question into a visible state and propagates it along the team's own dependency graph. The SAS execution layer brings running and log review into the same window and parses the log into navigable, summarized results. The report assembler removes the manual cut-and-paste of combining outputs. Each of these can be adopted independently by teams facing the same problems. Future work includes broadening the automatic dependency inference and reducing the application's dependence on a locally installed Excel for the save path.

REFERENCES

[1] SAS Institute Inc. 2023. *SASpy Documentation*. Cary, NC: SAS Institute Inc. Available at https://sassoftware.github.io/saspy/.

[2] Qt Company. 2024. *Qt for Python (PySide6) Documentation*. Available at https://doc.qt.io/qtforpython/.

[3] ZoomerAnalytics. 2024. *xlwings Documentation*. Available at https://docs.xlwings.org/.

ACKNOWLEDGMENTS

The author thanks colleagues in the statistical programming team at Jiangsu Hengrui Pharmaceuticals for testing the application against production studies and for the feedback that shaped its design.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Mingjie Fu

Jiangsu Hengrui Pharmaceuticals Co., Ltd.

mingjie.fu.mf7@hengrui.com