PharmaSUG China 2026 - Paper AA-170

Demotion Trace-Lite: An application for demotion management

Sizhen Chen, Pfizer

ABSTRACT

In standardized clinical programming environments, system-level code libraries are typically developed within each organization to maximize reuse, consistency, and efficiency across studies. However, these standardized utilities cannot always fully accommodate study-specific requirements through parameterization alone. In such situations, programmers may need to copy selected “system” utilities into the study-level environment and apply controlled modifications to meet specific analysis needs. This paper uses the term “demotion” to describe the deliberate and well-documented customization of standard, system-level utilities at the study level. Importantly, demotions should not be interpreted as coding errors or deviations from best practice, but represent justified, transparent adaptations required to address unique study requirements, while maintaining traceability. From a quality and governance perspective, demotions should be systematically documented to capture how it differs from the original version. Beyond traceability, a well-maintained demotion tracker also enables continuous improvement by highlighting gaps in standard utilities as they are identified in practice. In many cases, however, demotion details remain embedded in code comments and are managed manually, making them difficult to extract, review, or share - especially across teams with different conventions. To address this, we developed Demotions Trace-Lite, a lightweight Python Streamlit-based application that automatically extracts and consolidates demotion information from code. The tool generates standardized, review-ready outputs and captures key metadata such as change rationale and authorship, supporting both ongoing quality improvement and audit readiness.

INTRODUCTION

Standardized code libraries play a critical role in clinical programming by promoting consistency, efficiency, and quality across studies. Through carefully designed frameworks and parameterized interfaces, system-level programs and macros aim to address a wide range of data preparation and reporting scenarios while minimizing study-specific customization.

Despite these efforts, clinical studies frequently involve study-specific requirements arising from unique study designs, analysis objectives, or reporting needs. While many variations can be handled through parameter setting, certain scenarios fall outside the scope of standard functionality. In such cases, programmers must apply controlled customization to meet the intended programming purpose while maintaining traceability and quality oversight.

In this paper, this controlled migration and modification of system-level standard code into the study-level environment is referred to as “demotion.” A demotion represents a deliberate and justified study-level customization rather than a coding error or deviation.

This paper focuses on the management and review of such demotions and presents an automated approach to managing and reviewing this process in a consistent and scalable manner.

BACKGROUND

In standardized clinical programming environments, system-level code libraries—such as standard SAS macros and programs—are designed to support data preparation and reporting across a wide range of studies. These components typically provide configurable parameters to accommodate common variations in study design and reporting requirements, with the objective of maximizing reuse and minimizing redundant development.

In practice, however, some protocol-specific requirements cannot be fully addressed through parameterization alone. Examples include unique derivation logic, special handling of edge cases, or reporting requirements that extend beyond the intended scope of existing standard code. When this

occurs, programmers often migrate a system-level program or macro into the study environment and implement controlled modifications to meet study-specific needs.

This practice is a common and generally accepted part of real-world project delivery. At the same time, from a quality management and standards governance perspective, organizations aim to maximize the coverage of standard libraries and minimize unnecessary study-specific customization. Uncontrolled or poorly documented customization can reduce transparency during reviews, complicate documentation updates, and limit the ability to identify opportunities for improving standard code coverage.

Because demotions represent intentional study-level adaptations, they should be documented with sufficient context, including the modified component, rationale, author, date, and relationship to the original standard implementation. This level of documentation helps reviewers understand not only what was changed, but also why the change was needed and whether it remains appropriate over time. Such documentation supports traceability and continuous improvement as well as audit and inspection activities by clarifying how the study-level implementation differs from the original system-level version.

Despite the importance of demotion documentation, demotion information is often maintained manually and embedded only within program or macro comments throughout the study lifecycle. Under tight timelines and frequent code iteration, these comments may become inconsistent, incomplete, or difficult to locate.

Retrieving demotion information for specification updates or code review typically requires manual searching across multiple programs and macros within a study. This process is time-consuming and prone to human error, particularly in large or long-running projects. In addition, variations in commenting practices make it difficult to consolidate demotion information in a structured and review-ready manner.

From a code quality perspective, the absence of systematic review mechanisms also makes it challenging to identify redundant or outdated demotions that may no longer be necessary. These limitations highlight the need for a more efficient and standardized approach to demotion management.

To address the challenges, a Streamlit application ‘Demotion Trace-Lite’ was developed to automate the identification and traceability of study-level demotions. The tool assists programmers and reviewers by automatically extracting, consolidating, and standardizing demotion information through scanning relevant SAS programs, macros, and generates a structured, review-ready output.

The tool is intentionally lightweight and focused, targeted solution that complements existing programming workflows rather than replacing them. By automating the collection and standardization of demotion information, it reduces manual effort, improves consistency, and strengthens traceability in demotion management.

WORKFLOW DESIGN AND IMPLETEMENT

WORKFLOW OF DEMOTION TRACE-LITE

User Input
Users specify the path for study, and files to be inclusion. A demotion comment pattern is also provided to guide extraction.

Directory Scanning
The tool scans the specified study directories and identifies relevant programs and macros for analysis.

Demotion Extraction
Structured demotion comments are parsed based on the configured matching pattern. Only comments that conform to the specified structure are extracted, ensuring consistency.

Detect Related Macros and Classification
By leveraging program-level information, the demotion of macro can be classified into different types (for datasets or for TLFs), and linked to the corresponding impacted outputs. This classification provides additional context for downstream dependency tracing and impact analysis.

Standardized Output
Extracted demotion information is consolidated into a structured excel file, suitable for documentation alignment and review.

This modular workflow allows flexibility while maintaining a consistent extraction and review process.

Figure 1 shows the interface of Demotion Trace-Lite.

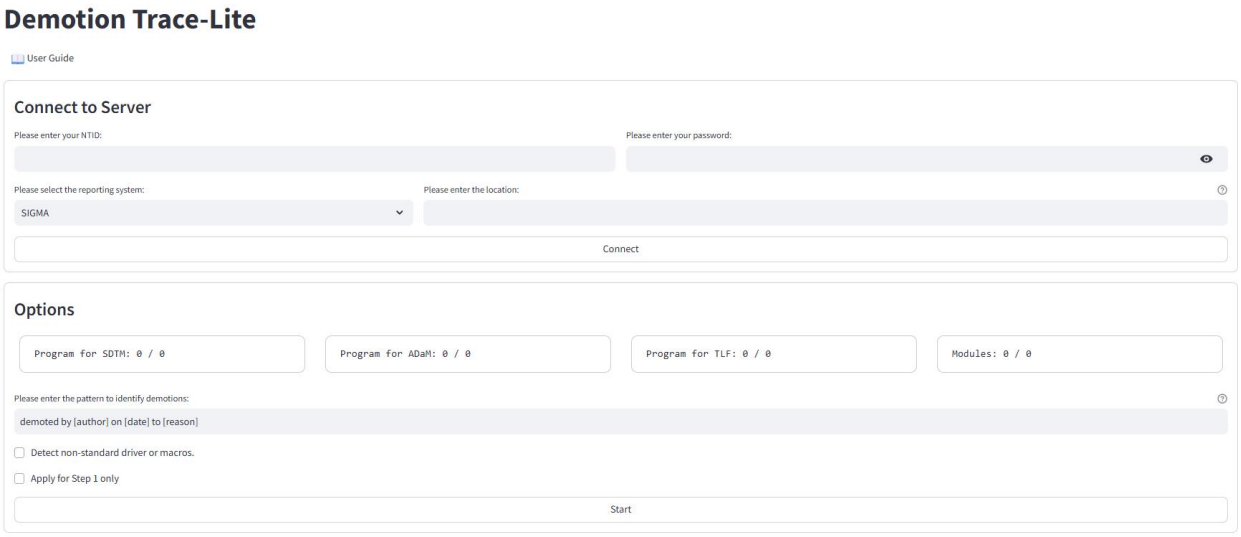


Figure 1. the interface of Demotion Trace-Lite

THE PATTERN OF DEMOTION COMMENT

The core extraction logic of Demotion Trace-Lite is driven by structured comment patterns. A commonly recommended demotion comment format is:

demoted by [author] on [date] to [reason]

This structure captures key elements required for demotion documentation, including author, timestamp, and rationale. Demotion Trace-Lite allows this format to be configured as a matching pattern for automated extraction. Comments in SAS code could be extracted in following examples:

- `/* Demoted by Sizhen on 23Jun2026 to show the usage of the pattern */`
- `* Demoted by Sizhen on 23Jun2026 to show the usage of the pattern;`

In these two comments, “Sizhen”, “23Jun2026” and “show the usage of the pattern” would be recognized as author, date and reason respectively.

To accommodate variations across teams, legacy studies, or evolving conventions, the tool also supports user-defined demotion comment formats. Key words in the pattern should be denoted by [], and One key word except [reason] could only match one word.

The tool converts the pattern into a regular expression to help user easily extract the information from the code comment. Following code shows the logic for conversion.

```
def define_pattern(self, match_pattern:str) -> None:
        info = re.findall(r'(?<=\[)\w+(?=\])', match_pattern)
        self.info = []
        for item in info:
            if item.lower() == "reason":
                self.info.append("Reason")
            else:
                self.info.append(item)

        pattern = re.sub(r'\s+', ' ', match_pattern)
        pattern = re.sub(r'\[reason\]', '(.+)', pattern, re.I)
        pattern = re.sub(r'\[\w+\]', '(\\\S+)', pattern)
        self.pattern = re.compile(pattern, flags=re.I)
```

By externalizing pattern configuration from the extraction engine, Demotion Trace-Lite balances standardization with flexibility and ensures maintainability as conventions evolve.

In practical use, Demotion Trace-Lite has been applied during routine documentation updates and periodic code reviews. This tool enables teams to quickly obtain a comprehensive view of study-level demotions without manual searching, improving efficiency and minimizing the risk of omission.

By consolidating demotion details into a structured and easily reviewable format, the tool supports clearer review discussions and facilitates alignment between code and specification documents. Optional checks further assist reviewers in identifying redundant or outdated demotions that may warrant reassessment.

Overall, the approach improves transparency, reproducibility, and confidence in demotion management.

DETECT RELATED TLFS

Demotion Trace-Lite applies a recursive dependency-tracing workflow to identify macro-level dependencies from SAS programs.

The process begins with an input SAS file, which is first checked against an internal mapping structure that records relationships between programs and their associated macros. This mapping, referred to as the Program-Macro Mapping, stores entries in the format of dictionary: {program: [macros]}. The term “program” here indicates the SAS file to be executed to invoke related macros and generate dataset or TLF; and the term “macro” means the SAS macro to be called and stored with the same file name.

If the input file already exists in the mapping, it is skipped to prevent duplicate processing and avoid circular dependency tracing. Otherwise, the tool proceeds to scan the SAS source code to detect macro invocations following the %xxxx pattern.

During macro detection, non-relevant macro constructs are explicitly excluded to improve accuracy. Specifically, the following elements are filtered out:

- **Macro statements**, such as %do, %macro, and similar control constructs

- **Macro functions**, such as %str, %scan, and other built-in functions
- **Macro labels**, such as %exit:

After filtering, the remaining %xxxx patterns are treated as candidate macro calls. These detected macros are then resolved to their corresponding source files, generating a list of dependent files associated with the current input file.

Each file in the dependency list is subsequently processed using the same workflow. In other words, every newly identified file is treated as an independent input and undergoes the same sequence of checks and macro detection steps. This recursive mechanism enables the tool to capture multi-level dependencies across nested macro hierarchies.

To ensure efficiency and avoid redundant processing, the Program-Macro mapping structure serves as a registry of processed files. Each file is processed only once, even if it appears multiple times across dependency chains. Once all dependent files have been evaluated, the current file and its associated macro information are appended to the mapping.

The resulting mapping provides a consolidated view of program-to-macro relationships, supporting downstream use cases such as demotion impact assessment and dependency analysis and code review support.

For example, there is a SAS program named t_ae.sas, which calls the SAS macro ae_summary to generate AE summary table:

```
%ae_summary;
```

The macro ae_summary is stored in the file ae_summary.sas and calls another SAS macro std_count. And the macro std_count is demoted, and the details are recorded in the code comment. These two macros are defined as:

```
%macro ae_summary;
    …
    %std_count;
    …
%mend ae_summary;

%macro std_count;
    …
    /*Demoted by chens291 on 23Jun2026 to demonstrate the workflow of
detecting related TLF*/
    …

%mend std_count;
```

The tool would generate the Program-Macro Mapping of {“t_ae.sas”: [“ae_summary.sas”, “std_count.sas”]}. Therefore, the impacted output is t_ae.

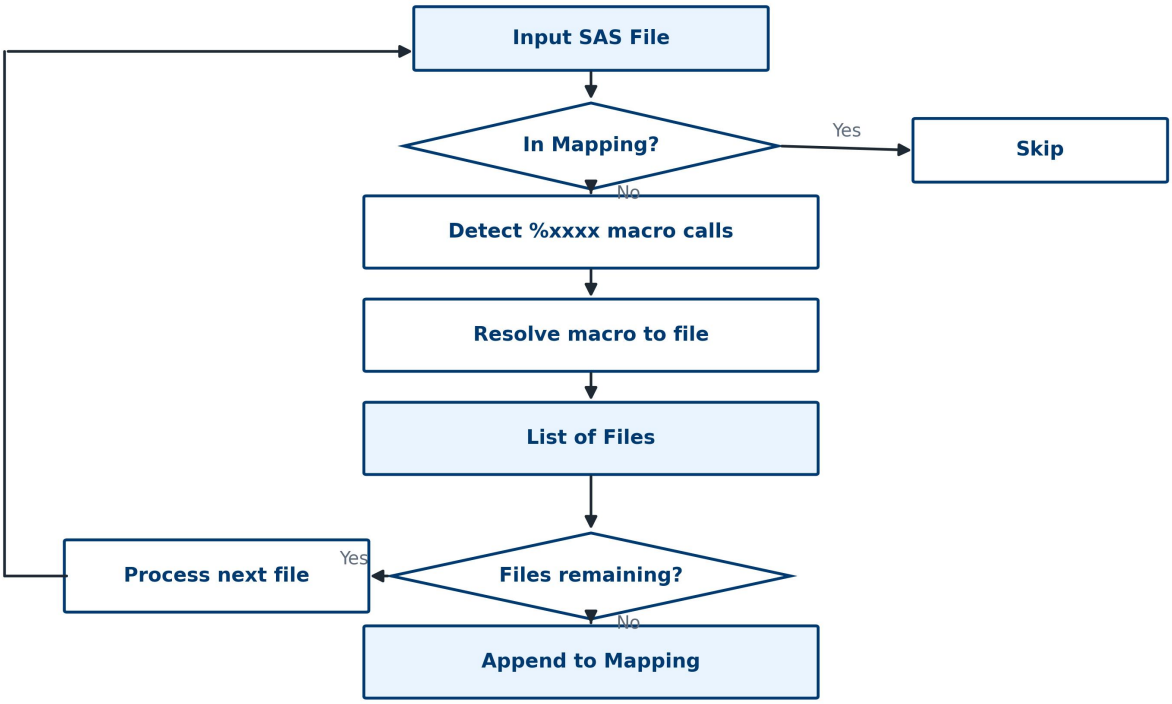


Figure 2. Recursive macro-dependency tracing workflow

RESULT PRESENTATION

The final results are presented in the Streamlit interface, enabling an interactive review of the extracted demotion information. The results can also be exported as a structured Excel file to support downstream analysis and documentation.

Figure 3 and Figure 4 illustrate examples of the corresponding outputs generated by the tool. The key words author, date and reason are successfully extracted and summarized.

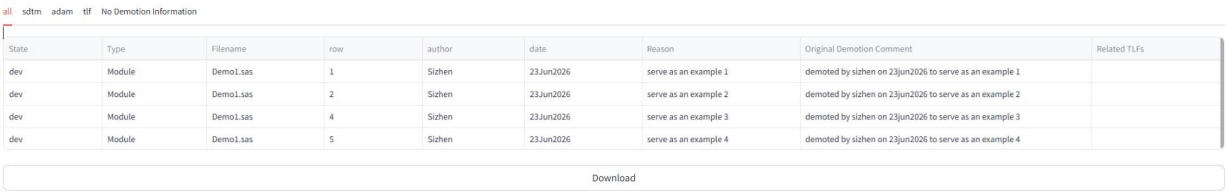
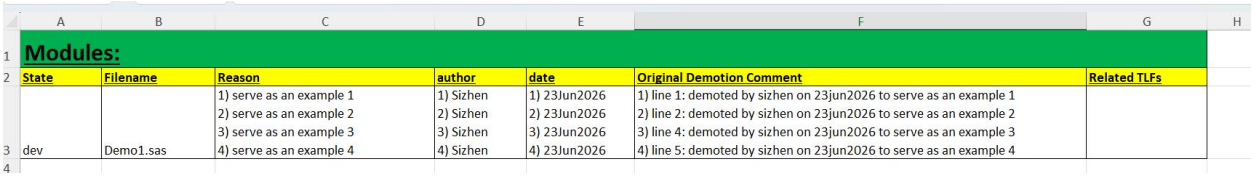


Figure 3. Result in Streamlit Interface



Modules:						
State	Filename	Reason	author	date	Original Demotion Comment	Related TLFs
		1) serve as an example 1	1) Sizhen	1) 23Jun2026	1) line 1: demoted by sizhen on 23jun2026 to serve as an example 1	
		2) serve as an example 2	2) Sizhen	2) 23Jun2026	2) line 2: demoted by sizhen on 23jun2026 to serve as an example 2	
		3) serve as an example 3	3) Sizhen	3) 23Jun2026	3) line 4: demoted by sizhen on 23jun2026 to serve as an example 3	
dev	Demo1.sas	4) serve as an example 4	4) Sizhen	4) 23Jun2026	4) line 5: demoted by sizhen on 23jun2026 to serve as an example 4	

Figure 4. Result in Excel

CONCLUSION

Controlled customization of standard code is an inherent and sometimes necessary aspect of clinical programming. However, when such customization is not consistently documented and reviewed, it can reduce transparency, complicate specification alignment, and limit opportunities for improving standard libraries. Effective demotion management is therefore important not only for single study delivery, but also for broader quality oversight and standards governance.

Demotion Trace-Lite provides a practical and lightweight approach to supporting this process. By scanning SAS programs and macros, extracting structured demotion information from code comments, and consolidating the results into review-ready outputs, the tool reduces manual effort and helps study programming teams obtain a clearer view of study-level adaptations. Its ability to support dependency tracing and classification further strengthens review discussions and impact assessment.

As a focused utility, Demotion Trace-Lite is intended to complement existing programming workflows rather than replace established review processes. Future enhancements may include more flexible pattern recognition, closer integration with version control or study documentation workflows, and expanded reporting to support standards governance. Overall, the tool demonstrates how targeted automation can improve consistency, traceability, and review efficiency in clinical programming.

CONTACT INFORMATION

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

Sizhen Chen
Pfizer
Sizhen.Chen@pfizer.com