PharmaSUG 2026 - PO - 287

Maintaining a Multi-Lingual Code Inventory in Real-World Evidence

Joshna Reddy Nimmala, Yu Feng, Merck & Co., Inc., Rahway, NJ, USA

Abstract

As healthcare claims databases continue to evolve in structure and complexity, statistical programmers are increasingly challenged to maintain consistency, efficiency, and compliance across Real-World Data (RWD) analysis. To address these challenges, programmers rely on multi-lingual code inventories (including SAS, R, and Python scripts) and adapt them as needed to support Real-World Evidence (RWE) generation. However, without a centralized and dynamic approach, these inventories can become fragmented, outdated, and difficult to manage. This paper presents a scalable framework for maintaining a centralized RWE code inventory that supports detection of structural changes in RWD sources. By identifying key structural triggers, such as additions or changes in variable names, coding systems, or data schema, this framework enables intelligent searches across the code inventory to flag impacted scripts and suggest updates. This automation reduces manual effort, accelerates programming workflows, and minimizes the risk of errors. Additionally, the framework incorporates traceability and version control mechanisms to ensure reproducibility and audit readiness, aligning with ICH E6(R2) Good Clinical Practice guidelines. This approach not only enhances operational efficiency but also supports regulatory compliance and cross-study consistency, making it an asset in the evolving RWE landscape.

Background

Pharmaceutical companies increasingly depend on Real-World Data (RWD) to generate insights supporting clinical development, regulatory submissions, and health-economic evaluations. Common external data sources include electronic health record (EHR) datasets and patient-level claims from platforms such as Flatiron, MarketScan, Optum, SEER, and others.

In addition to providing regularly refreshed patient-level data often monthly, quarterly, or at other predefined intervals, these vendors may also implement periodic structural updates. These changes may affect data formats, variable definitions, table structures, coding conventions, or file organization. When such updates occur, vendors typically send detailed communications to analytic teams (see Figure 1 for an example).

Although the RWD code inventory spans programs written in SAS, R, and Python, the maintenance framework is centralized in SAS, functioning as an orchestration layer that consistently searches and evaluates impacts across all supported programming languages.

Figure 1: Data Refresh communication

Dear RWE Platform Users:

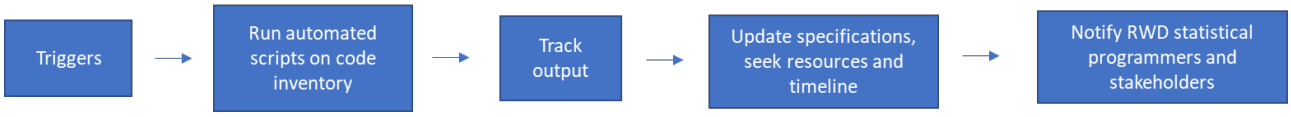
Note: Please be informed that the below Structural changes have been made for this dataset (schema: rwe_raw_medical_claims_data) in this release are as follows:

Table Name	Column	Datatype	Update/changes
ref_tos_cd	tos	varchar(25)	column name **"tos"** bes renamed to **"tos_cd"**
ref_tos_cd	cd	varchar(10)	New column added
desc	desc	varchar(12)	New column added
ref_ipc_code_modifier	e	varchar(10)	Column removed
ref_d_fed_poverty_stats	t	New Table	New Table added
ref_top_svc	refs	Removed	Table name **"ref_tos"** renamed to **ref_cd**

Description	Data Asset	Date Received f Vendor	Date Loaded Into RWE	Date Range Covered
Medical	Medical Claims Data dataset Schema: rwe_raw_ medical.claims	10/13/2025	10/29/2025	Till-20250508

For downstream RWD analytics teams, maintaining a robust and adaptable multi-lingual code inventory becomes essential. Such an inventory enables both statistical programmers and stakeholders to respond quickly to data-structure changes, maintain reproducibility, and minimize disruptions to ongoing analyses. Figure 2 outlines the typical workflow for code-inventory maintenance.

Figure 2: Steps for RWD code inventory maintenance



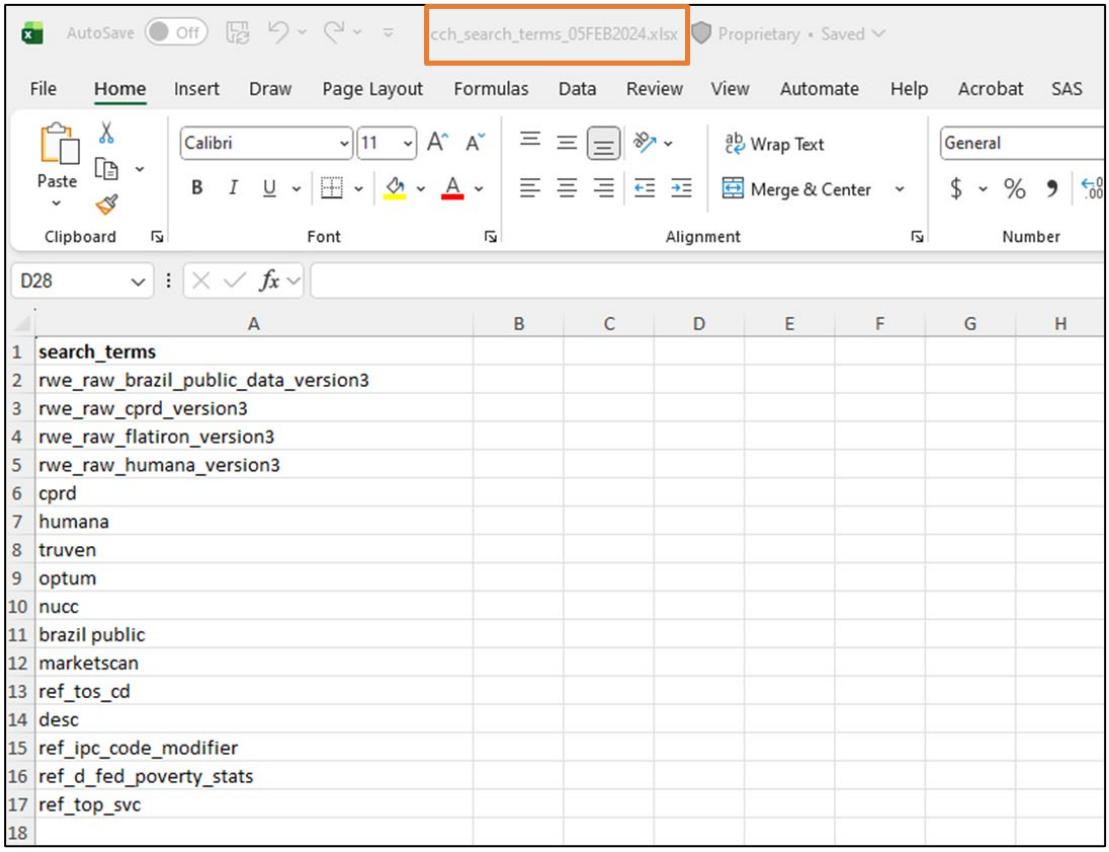
Code Inventory Maintenance Workflow

Vendor communications that indicate database changes beyond routine data refreshes (Figure 1) serve as key triggers to initiate automated code-inventory checks. These checks ensure statistical programmers and other stakeholders are alerted to potential impacts before errors surface in production environments.

A. Identify and Prepare Search Terms

All search terms relevant to the structural changes are compiled and documented in an Excel file. For example, if maintenance is conducted on 05FEB2024, the file may be named cch_search_terms_05FEB2024.xlsx. Figure 3 shows an example of such an input file.

Figure 3: Sample input to run automated scripts



B. Import Search Terms into the Programming Environment

Once the search terms are imported, an automated script (Figure 4) scans the entire RWD code inventory. This script incorporates Unix utilities such as grep and cd within the SAS environment to eliminate manual searching.

Below automated script (Figure 4) reads search terms from an Excel file, prepares them for use as SAS macro variables, and uses grep and directory navigation commands to identify all code locations containing those terms. The comment blocks in the script outline each step in detail.

Key grep options used include:

- -r: recursive search through directories and subdirectories
- -i: case-insensitive matching
- -n: display matching line numbers
- -m: limit or count matching lines

Figure 4: Automated script

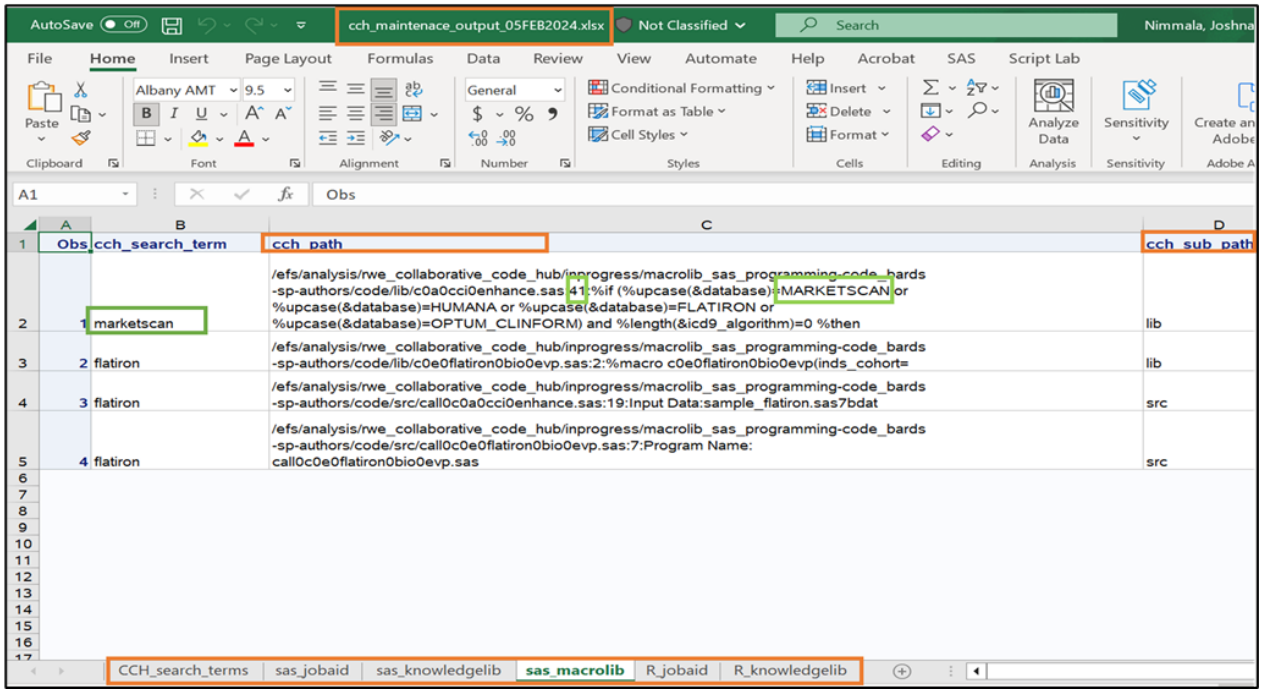
```
*- Step 01:  Import search terms from excel to sas-*;

proc import datafile = "&rawdir.&cch_excel"
            out      = CCH_search
            dbms     = xlsx replace ;
            getnames = yes ;
run ;


*- Step 02: Convert search terms to macro variable to be used by grep search-*;

proc sql noprint;
     select distinct cch_search_terms into :grep_search separated by '\|' from CCH_search ;
     select count(cch_search_terms) into :max_cch_terms from CCH_search ;
quit;

%put &grep_search &max_cch_terms ;

data dummy ;
 set  CCH_search ;
 call symput('all_cch'||strip(put(_n_,best.)), strip(lowcase(CCH_search_terms))) ;
run;

*- Step 03: Insert grep and cd commands in sas data step-*;

%macro cch_check(macro_level1=,  macro_level2= , out = );
filename &out pipe "grep -rinm 1 '&grep_search'  &cch_searchpath.&macro_level1./code/&macro_level2*";
    data pre_&out ;
    infile &out truncover;
    length cch_path $ 300  cch_sub_path $8  cch_search_term $40 ;
    input cch_path $ 300. cch_sub_path $8. cch_search_term $40.;
    cch_sub_path = "&macro_level2" ;
    %do i = 1 %to &max_cch_terms. ;
        if index(lowcase(cch_path), "&&all_cch&i.") > 0 then cch_search_term="&&all_cch&i." ;
    %end;
run;

proc sql noprint ;
    create table &out as
    select cch_search_term, cch_path, cch_sub_path
    from pre_&out ;
%mend;

/* Step 4:  call above macro accordingly for each macro/code inventory and export the output in excel */

%cch_check(macro_level1 = jobaid_sas_programming-code_bards-sp-authors,  macro_level2 = lib, out = SASjoblb);
```

C. Evaluate Output and Prioritize Updates

Figure 5 presents a sample output from the automated search process. The Excel output contains multiple tabs, one for each directory included in the search. In the example shown, the script was executed across five RWD directories, with the first tab providing a summary of all search terms. Before exporting to Excel, users can rename each tab using clear, descriptive labels that follow naming conventions established across RWD teams. Each tab includes three columns: (1) Search Term, (2) Directory, and (3) Sub-directory. As described in Section B, the automated script uses the grep -rinm options, which allow the output to capture not only the file paths but also the specific line numbers where each search term appears.

Statistical programmers review the findings to determine whether program specifications must be updated to align with the latest database structures and functionality.

Figure 5: Sample output from automated scripts



Although this process can be run monthly, it can also be triggered immediately in high-priority or critical situations. After generating the output:

- A summary is communicated to study analysis leads
- Resources are identified for required program updates
- Timelines are established to implement code modifications

This ensures timely and efficient alignment with updated RWD structures.

Conclusion

Frequent structural updates in Real-World Data sources make it critical to maintain a consistent and adaptable multi-lingual code inventory. The automated SAS based framework presented here streamlines detection of these changes, reduces manual searching, and ensures reproducibility across SAS, R, and Python programs. By standardizing how search terms are captured and evaluated, this process helps RWE teams stay aligned with evolving data structures. When paired with a well-organized team responsible for code upkeep, this approach reduces errors and improves efficiency. Further research leveraging Generative AI to more fully automate this process is warranted, as these emerging capabilities may enhance scalability, adaptability, and long-term sustainability of code maintenance workflows.

Acknowledgements

The authors would like to thank our management, Mary Anne Rutkowski and Xingshu Zhu, for their support and valuable feedback during the development of this paper.

Contact Information

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

Joshna Reddy Nimmala
Merck & Co., Inc
Email: joshna.reddy.nimmala@merck.com

Yu Feng
Merck & Co., Inc
Email: yu.feng1@merck.com

Trademarks