Paper RE10

Where Are My Agents? Automating Cohort Building with a Dynamic Cohort Builder

Sherrine Eid, MPH, SAS Institute, Inc., Cary, NC, USA
Mary Dolegowski, MSQM, SAS Institute, Inc., Cary, NC, USA
Sundaresh Sankaran, MS, SAS Institute, Inc., Cary, NC, USA

ABSTRACT

Defining and building patient cohorts involves a mix of automated and manual methods that leverage structured data, unstructured data, and advanced analytics. Producing reliable real-world evidence that informs clinical practice, supports regulatory decisions, and ultimately improves patient outcomes is imperative for regulatory acceptance of integrated evidence packages.

Precisely defining patient cohorts affects study validity and reliability by reducing bias and confounding, ensuring that the evidence generated is robust and valid. This enables meaningful comparisons between treatments and helps in drawing reliable conclusions about effectiveness and safety.

Regulators and payers increasingly rely on RWE to complement randomized clinical trials (RCTs) in decision-making. There is a plethora of examples illustrating how well-defined cohorts support regulatory submissions, label expansions, and post-marketing surveillance.

Identifying distinct patient subgroups can lead to targeted therapies and more personalized treatment strategies. This enables the exploration of treatment effects in real-world subpopulations that might be underrepresented in RCTs.

Standardized and reproducible cohort definitions streamline the research process, making studies more efficient and scalable and facilitates data sharing and collaboration across institutions, which is crucial for large-scale observational studies.

This session will explore the many options researchers have at their disposal to efficiently and accurately define cohorts and accelerate evidence generation in support of drug development and integrated evidence packages.

INTRODUCTION

The process of defining and building patient cohorts in pharma today involves a mix of automated and manual methods that leverage structured data, unstructured data, and advanced analytics. Each approach carries trade-offs between scalability, accuracy, interpretability, and resource requirements. The importance of these methods lies in their ability to produce reliable real-world evidence that informs clinical practice, supports regulatory decisions, and ultimately improves patient outcomes.

Properly defining patient cohorts affects study validity and reliability as accurate definitions reduce bias and confounding, ensuring that the evidence generated is robust and valid. They enable meaningful comparisons between treatments and help in drawing reliable conclusions about effectiveness and safety.

Regulators and payers increasingly rely on RWE to complement randomized clinical trials (RCTs) in decision-making. There is a plethora of examples illustrating how well-defined cohorts support regulatory submissions, label expansions, and post-marketing surveillance.

CUSTOM STEPS – A TOOL FOR BUILDING COHORTS THROUGH A REPEATABLE PROCESS

Cohort identification makes use of multiple techniques, ranging from simple rule-based matches to unsupervised techniques such as clustering. These techniques are codified in the form of SAS programs, and in recent times, also through other open-source interfaces such as Python or R. This implies that the code processed could turn out to be complex and unwieldy for user personas of a non-programming background.

Every technique requires appropriate parameters to output high-quality cohorts. SAS Viya Custom Steps allow users to create repeatable code chunks that are integrated with a graphical user interface (GUI) that capture parameters, process code and make output (cohorts) available for downstream application. Custom steps provide user-friendly interfaces, enhance accessibility for downstream users who may not be familiar with the underlying code, allow for control and defensive programming techniques to capture and handle errors, and also reduce the likelihood of errors related to data selection and manipulation.

In short, use of tools such as SAS Studio Custom Steps improves “first-pass yield”. A metric that originated in the manufacturing domain but increasingly recognized and applied in all processes, first-pass yield measures the efficiency of a process in providing the correct output with minimal rework. This translates into higher productivity for users across the organization.

BUILDING A CUSTOM STEP

We continue with the factory mindset by outlining approaches that help users rapidly build custom steps. Firstly, let’s look at the basic architecture of a custom step file. A custom step is quite a simple, serialised object which can be surfaced and viewed as a JSON file. The configuration of a step is defined through two areas:

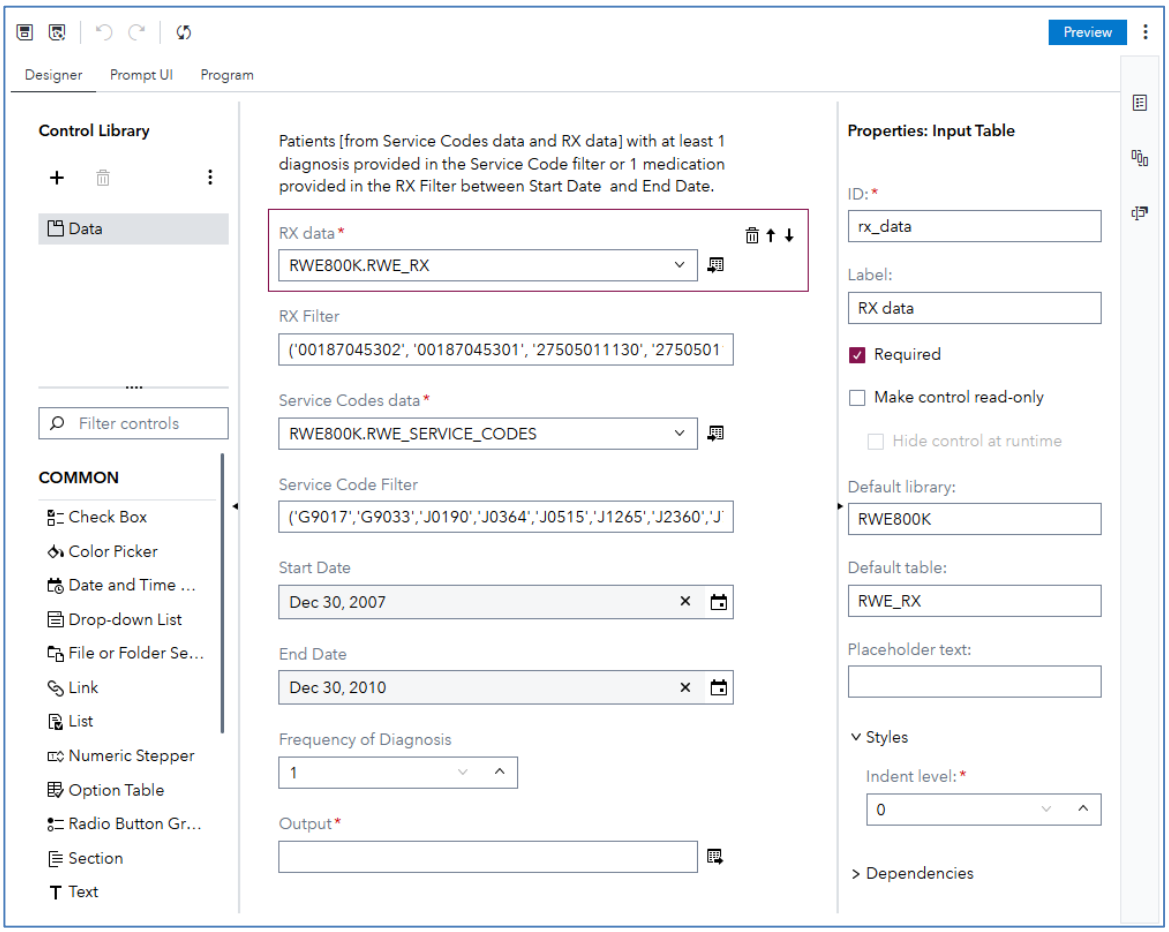
- Code: executed through a SAS program (which may in turn call other applications and languages as appropriate)
- Components: User interface components that capture input parameters and output specification

Based on their awareness of programming, users can choose from low-code or programmatic interfaces to build custom steps.

Graphical User Interface (GUI)

The low-code interface is accessed in SAS Studio using a Custom Step designer. Upon creating a new custom step, the user notices three tabs: Designer, Prompt UI, and Program. The Designer tab provides a GUI for building the controls that will be presented to the end user. Controls are added using drag-and-drop functionality, and their specific properties can be configured in the Properties panel, as shown in dations, predictions, or judgments. Display 1.

For example, the control highlighted in Display 1 is an input data control with its ID set to 'rx_data'. This ID will be referenced in the Program tab, where it’s surfaced as a macro variable. The designer can configure additional settings using the Properties panel, such as making the field required, setting default values, or establishing more complex behaviors like dependencies. This flexibility allows designers to create user interfaces that range from simple to highly customized, depending on the needs of end users.



Display 1: Custom Step – Designer

In Program 1, we see the code associated with the design shown in Display 1. This code uses a collection of macro variables that correspond to various controls defined in the Designer tab. For example, 'select CONSISTENT_MEMBER_ID from &rx_data. where CODE_VALUE in &RX_filter.' references the macro variables &rx_data and &rx_filter. These variables are directly linked to the controls labeled RX Data and RX Filter in Display 1. This relationship between the Designer and Program tabs makes it easy to adapt existing code for use within a Custom Step. By simply converting user-configurable elements into macro variables, developers can integrate a GUI interface without rewriting the entire program.

```
/* SAS templated code goes here */

proc fedsql sessref = mysession;

create table and_Filtered_Member_IDs as
select DISTINCT _RX_Members.CONSISTENT_MEMBER_ID
from
       (select CONSISTENT_MEMBER_ID from &rx_data. where CODE_VALUE in &RX_filter.)
       as _RX_Members
INNER JOIN
       (select CONSISTENT_MEMBER_ID from &code_data. where CODE_VALUE in &code_filter.)
       as _Code_Members
ON _RX_Members.CONSISTENT_MEMBER_ID = _Code_Members.CONSISTENT_MEMBER_ID
;

create table Filtered_RX as
select RX.*
from &rx_data as RX
INNER JOIN
(
       select DISTINCT _RX_Members.CONSISTENT_MEMBER_ID
       from
       (select CONSISTENT_MEMBER_ID from &rx_data. where CODE_VALUE in &RX_filter.)
       as _RX_Members
       INNER JOIN
       (select CONSISTENT_MEMBER_ID from &code_data. where CODE_VALUE in &code_filter.)
       as _Code_Members
       ON _RX_Members.CONSISTENT_MEMBER_ID = _Code_Members.CONSISTENT_MEMBER_ID
)
as F
ON RX.CONSISTENT_MEMBER_ID = F.CONSISTENT_MEMBER_ID
;

create table Filtered_Service_Code as
select code.*
from &code_data as code
INNER JOIN
(
       select DISTINCT _RX_Members.CONSISTENT_MEMBER_ID
       from
       (select CONSISTENT_MEMBER_ID from &rx_data. where CODE_VALUE in &RX_filter.)
       as _RX_Members
       INNER JOIN
       (select CONSISTENT_MEMBER_ID from &code_data. where CODE_VALUE in &code_filter.)
       as _Code_Members
       ON _RX_Members.CONSISTENT_MEMBER_ID = _Code_Members.CONSISTENT_MEMBER_ID
)
as F
ON code.CONSISTENT_MEMBER_ID = F.CONSISTENT_MEMBER_ID
;

 create table &Step1_output. as
 select CONSISTENT_MEMBER_ID
 from
        (select CONSISTENT_MEMBER_ID,
                     count(CODE_VALUE) as count
        from FILTERED_SERVICE_CODE
        where
              SERVICE_BEGIN_DT >= date%str(%')&start_date.%str(%') AND
              service_end_dt < date%str(%')&end_date.%str(%') AND
              CODE_VALUE in &code_filter.
        group by CONSISTENT_MEMBER_ID) as subset1
where count >= &diag_freq.
;

run;
quit;
```

Program 1: Custom Step - Program

Programmatic Interface

Custom step designers appreciate programmatic interfaces to automate the entire process of building a custom step for cohort build. Advantages to using a programmatic interface over a drag-and-drop interface include:

1. Large-scale automation: The dynamicity of the number of query filters and their variety may require the designer to create multiple custom steps of similar nature, but with different specifications. A programmatic interface helps handle this demanding workload.

2. "Baked-in" parameters: In some cases, the designer may wish to specify hard limits and constraints which are not editable by the end-users and make some other parameters editable. A programmatic interface provides designers with a mechanism to enforce these non-editable values in a governed manner.

3. Standardisation: Programmatic interfaces allow designers convenient ways to enforce standards in terms of wording, choice of control, parameter values, visibility and enablement etc. all of which render the custom step more consumable to downstream users.

An open-source project – py-sas-studio-custom-steps (https://pypi.org/project/py-sas-studio-custom-steps/) - is an example of a programmatic interface to help create SAS Studio Custom Steps at scale. It is a Python package and can be installed through standard pip package installers as follows:

```
pip install py-sas-studio-custom-steps
```

Reminding ourselves again that a custom step is a simple, serialized object which has a JSON structure, we view this Python package as a set of methods which make it easy to create a custom step file through programmatic methods. The step designer can construct a custom step, attach a SAS program and UI component json to the same, and finally test the same before making it available for consumption.

Here's a simple snippet which shows how easy it is to create a custom step, assuming that the SAS code and the components UI has been finalized.

```
from py_sas_studio_custom_steps import CustomStep

custom_step = CustomStep()

# attach SAS program to custom step
custom_step.attach_sas_program(sas_file="/path/to/a/SAS/program.sas")

# attach UI components to custom step
custom_step.attach_ui(ui_json_file= "/path/to/a/UI/components.json")

# Generate a README (optional)
custom_step.generate_readme(description= "This custom step builds cohorts..")

# save custom step on storage
custom_step.create_custom_step(custom_step_path="/path/to/where/you/save/this_file.step")
```

A crucial final step – **always** test your code and run once before releasing to the "downstream user", even if the downstream user happens to be yourself. Right click on the step where you uploaded it, edit it and only after testing, go ahead and save it in SAS Content. SAS Content is the preferred home for most custom steps (even though it is possible to access and execute custom steps in storage) because it holds information about the custom step metadata within its registry and can form part of future transfer objects, which enables portability and use across environments. Custom steps are basically another form of SAS program at the end of the day and it's important that they go through the full cycle of testing and validation. Many programs have failed in the past due to simple issues like mismatched environmental variables or faulty assumptions and testing your step, irrespective of how they were built, is an ideal way to weed them out.

INTERACTING WITH CUSTOM STEPS

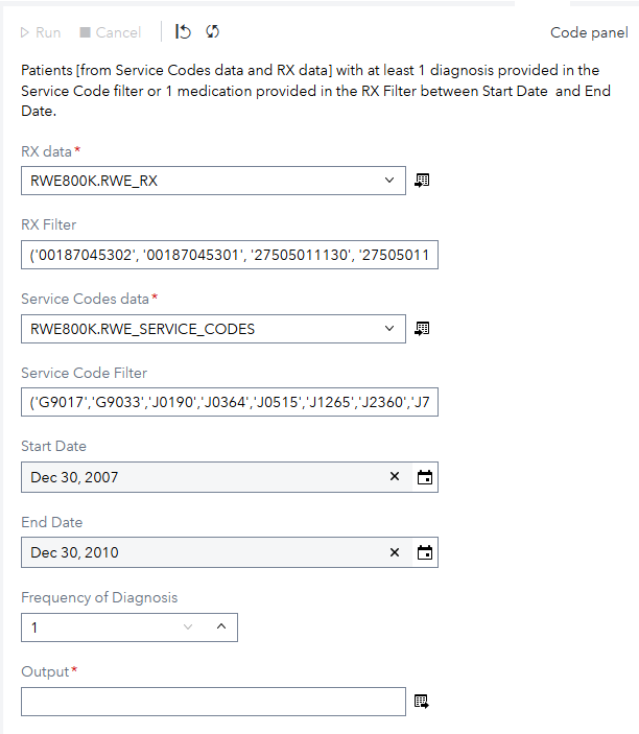
There are two primary ways to interact with Custom Steps once they have been built: as independent steps or as components within a SAS Flow. Running a Custom Step independently allows the specific process defined within it to be executed as a one-off action. However, this mode does not support scheduling. When a Custom Step is included within a Flow (even if the Flow consists of only a single step) it gains additional flexibility. The entire Flow can be executed either as a one-time run or scheduled as a recurring job.

INDEPENDENT

To interact with a Custom Step as a standalone component, right-click the step in question and select Open in a Tab. This action opens the Custom Step in a new tab, as shown in Display 2. In this view, you can see the output of the same program referenced in Display 1, with all associated controls rendered as interactive elements.

For instance, the RX Data control includes a link that lets users browse available libraries and datasets to select the desired input. Similarly, the calendar icon next to the Start Date field opens an interactive calendar for easy date selection.

It's important to note that in Display 2, the Run option is disabled because no output dataset has been selected. This is due to the output field being configured as a required variable within the Designer tab.

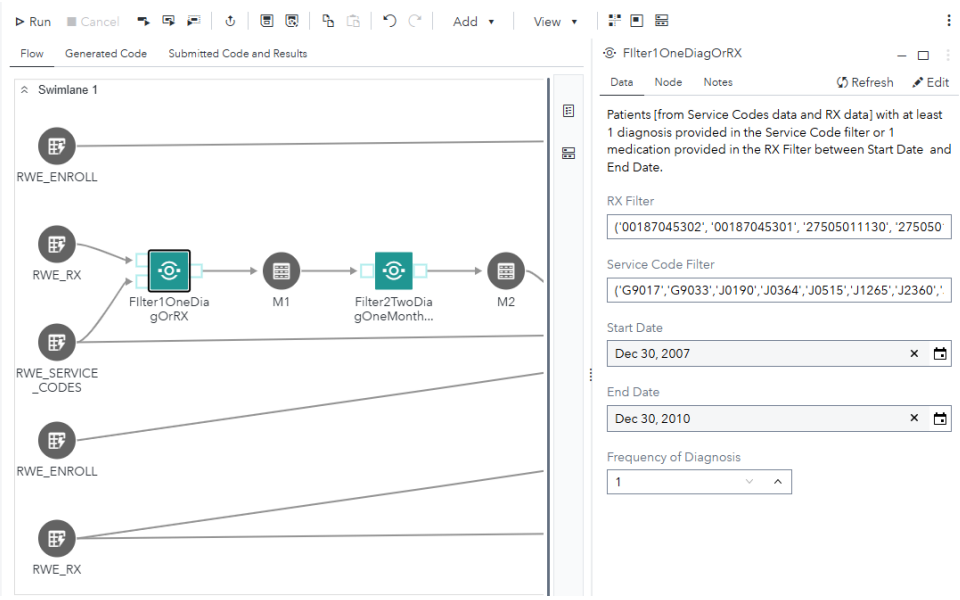


Display 2: Custom Step - Single Step

PROCESS FLOW

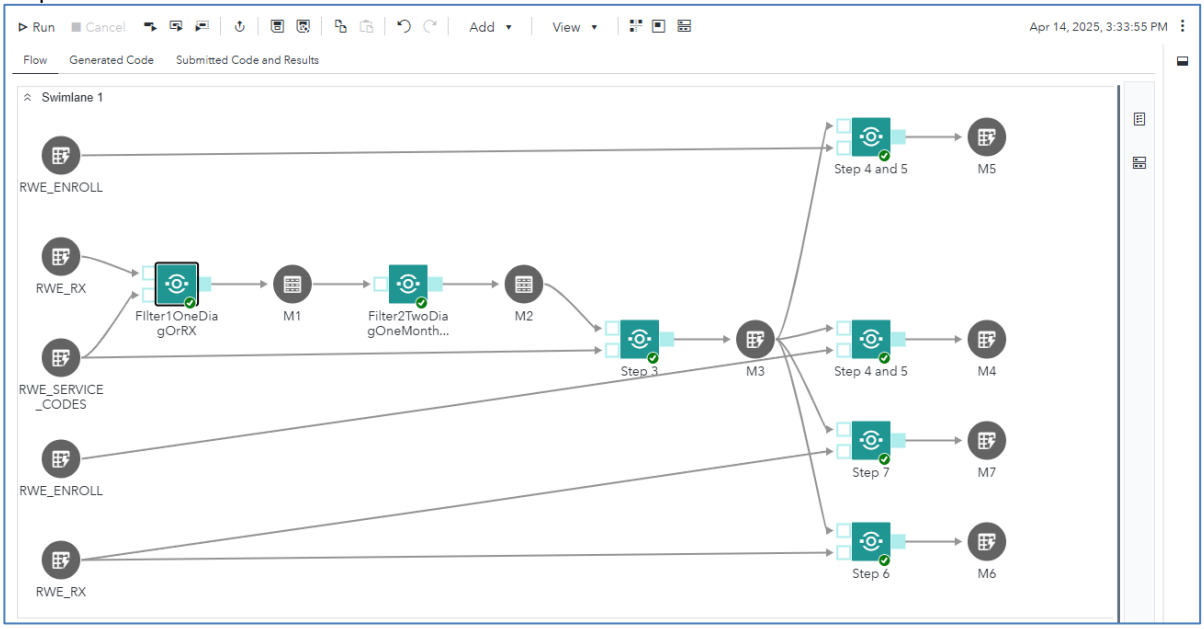
To use a Custom Step within a Flow, the user can either drag and drop the step into the Flow or right-click the step and select Add to Flow. In both cases, the Flow must already be open. The Custom Step can be selected from the user's program list, where it was originally saved, or, if it's stored in a shared location, accessed through the Step domain in SAS Studio under the Shared tab. This second option provides easy access to a centralized catalog of available Custom Steps.

The Custom Step has all the same interactive controls in a Flow as it does when used as a standalone component, as shown in Display 3.



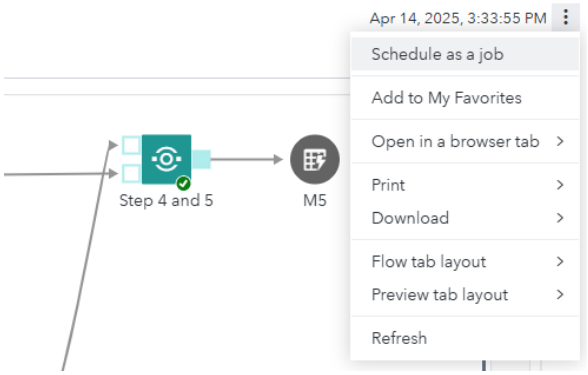
Display 3: Flow – Custom Step Interface

Display 4 shows an example of a completed Flow containing eight Custom Steps. Each step represents a different cohort filtering process. Notably, one of the steps is labeled Step 4 and 5, demonstrating how a single Custom Step can be reused in multiple contexts based on its configuration. This highlights the repeatability and flexibility of Custom Steps within a Flow.



Display 4: Flow

Finally, while the Flow and all of its internal Custom Steps can be run on an ad hoc basis by clicking the Run button in the top-right corner (as shown in Display 4), users also have the option to schedule the Flow as a background job. Display 5 illustrates this process: by selecting the ellipsis menu in the upper-right corner and choosing Schedule as a Job, users can open the New Trigger window. This interface guides them through the options for configuring when and how the Flow should be executed automatically.



Display 5: Flow - Schedule as a Job

WHY NOT HAVE AN 'AGENT' DO THIS FOR YOU?

Due to the popularity (and some weary folks may say almost criminal overuse) of the word "agent" in automated workflows, let us pause to acknowledge that analytics practitioners vary in their definition and understanding of what an agent is. For purpose of simplicity, we adopt a standard definition that establishes an agent as follows:

"An Agent is as Agent does."

Were we to adopt this definition, the role of the SAS Studio Flow in the above section becomes clearer. The entire flow can be considered as an agent because it has a defined purpose (to build cohorts given some specification), a set of specifications governing the range to which it operates, and a defined output it should strive to provide. This "SAS Studio Flow" agent is also characterized by the human triggers which intervene at certain endpoints. In this case, the human intervenes prior to the flow to provide parameters and after the flow execution to validate output.

The other characterization of the SAS Studio Flow agent is its determinism – the execution path is already well defined, and the flow of execution is modified only on hard conditions that have been prescribed beforehand. We do not suggest that these conditions change – in fact it is desirable for purpose of uniform execution. But we would like to also build some flexibility into the system to be able to understand when to run a series of steps in sequence and determine the sequence at which to run them in real time.

This is where we introduce the concept of tools. Tools are functions made available to an agent for use in a workflow. The choice whether to use a specific tool or not depends on the question at hand. A Large Language Model (LLM), a Generative AI component makes this decision and acts as the orchestrator of tool decisions.

We demonstrate this through SAS Retrieval Agent Manager (RAM), a new offering from SAS that enables agent build, LLM interactions and retrieval against unstructured data sources. Using RAM, the authors identified and created a data dictionary encapsulating the objective, input and output contracts and description of the process of each custom step in a collection. Every custom step can be considered a "tool" that is made available to the agent.

These tools need to be made available from a central place or "service" from where the LLM can access them. Here, we introduce the concept of a Model Context Protocol (MCP) server. An MCP server exposes tools that the agent chooses from, based on guidance from the LLM. MCP servers makes it possible to access local resources and functions within an agentic flow, components that a Large Language Model may not have direct access or ability to execute.

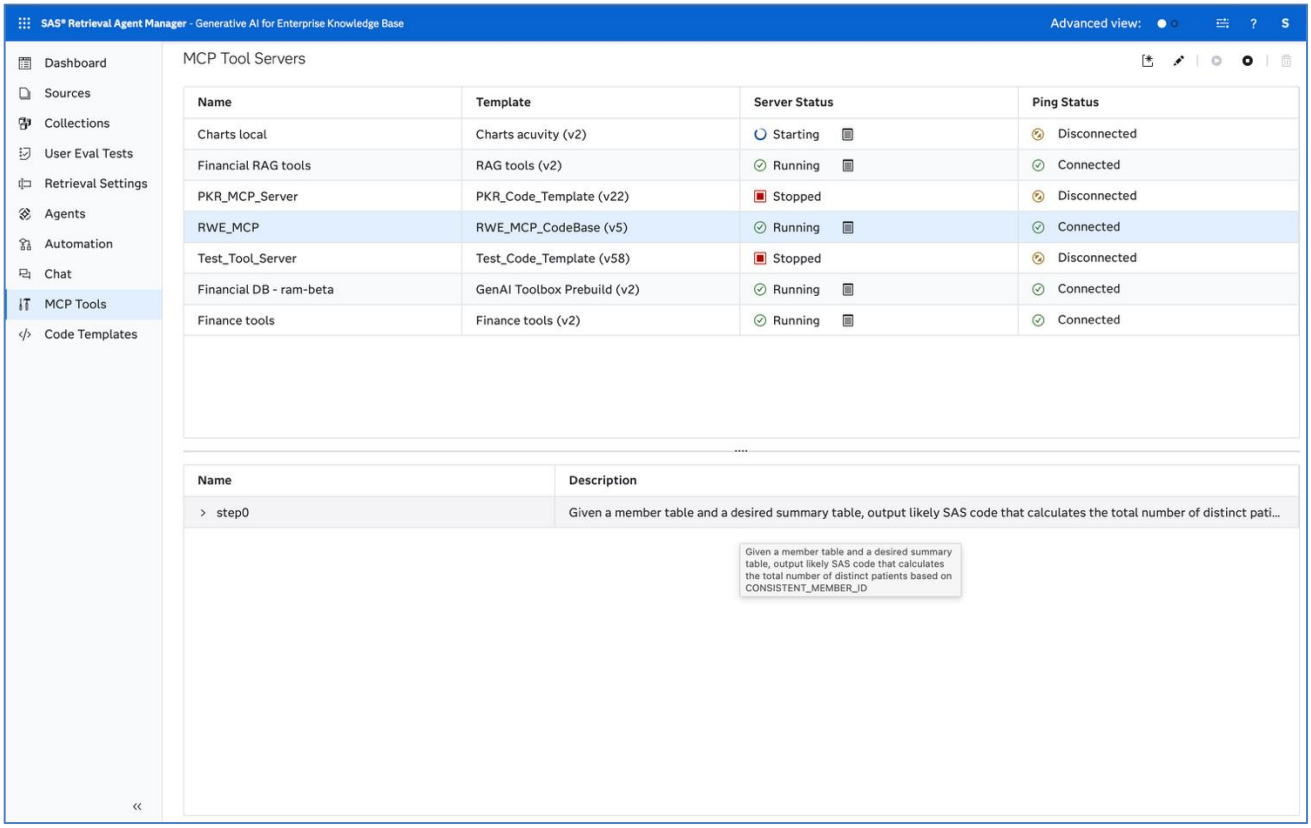


Figure 1: "Starting Block" example of an MCP tool server with a single tool for Step 0 (counting distinct patients)

In this case, the individual cohort builder programs are exposed as tools on a dedicated MCP server which is then available to an Agent within a RAM workflow. The Agent, upon receiving a query, makes use of Retrieval Augmented Generation (RAG) to identify the tools best suited for its purpose and then calls the MCP server to request execution of those identified tools in a decided sequence. Note the immediate benefit over a SAS Studio Flow in terms of not having to specify the execution path in a rigid manner. Further, arguments to provide to each code to populate input parameters are automatically input by the agent.

PROS AND CONS

Every approach has its pros and cons. When considering a decision whether to build cohorts through a SAS Studio Flow or a conversational agent that makes use of an LLM, designers should keep the following in mind.

SAS Studio Flow Agent – Pros and Cons:

Pros:

- Deterministic execution paths
- Programs do not deviate from planned flow
- Lineage and governance due to status as a SAS content “object”

Cons:

- Rigid execution paths
- Changes need to be hard-coded
- SAS / SQL based paradigms for filter – requires conformation to syntax

Conversational Agent (through Retrieval Agent Manager / other platforms) – Pros and Cons:

Pros:

- Flexible execution path
- Agent decides flow on case-by-case basis
- Conversational interface accommodating wide range of filters

Cons:

- Hallucination risks
- Need for stronger, explicit guardrails
- Need to track each execution path separately

ACCESSING YOUR AGENT

Both SAS Studio Flow agents and agents designed using Retrieval Agent Manager can be accessed and triggered via Application Programming Interfaces (APIs). SAS Studio Flows follow the jobs paradigm and are deployed as a job once designed. Designers follow the same method for deploying a SAS Studio Flow as a job as they would for any other job.

Agents designed in Retrieval Agent Manager exist as API endpoints and run on ephemeral resources – i.e. it's possible to deactivate an agent, make it sleep, and then make it operational at various points in time. This optimizes operational costs since resources to run an agent can be accessed only upon need. A typical practice is to design end user web applications (which may differ from client to client) which call RAM, establish credentials and make requests for various operations such as chatting with an agent, or querying tools from the MCP server etc.

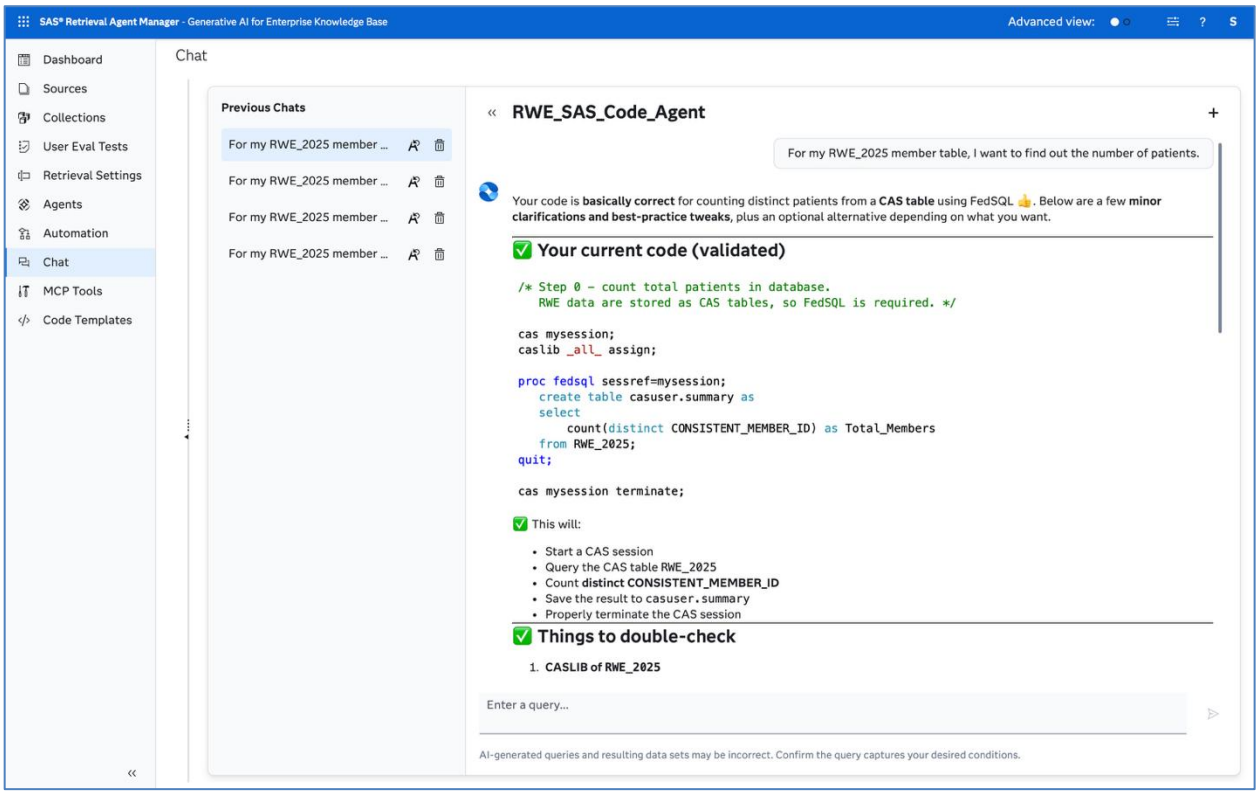


Figure 2: An example of an RWE Agent serving appropriate code

CONCLUSION

There are many options that researchers have at their disposal to efficiently and accurately define cohorts and accelerate evidence generation in support of drug development and integrated evidence packages. We demonstrated the ease of standardizing and reproducing cohort definitions to streamline the research process and make studies more efficient and scalable and facilitate data sharing and collaboration across institutions, which is crucial for large-scale observational studies.

The question is not whether we can build an agent to orchestrate cohort construction and analytical workflows. The

question is how we do so **responsibly, reproducibly, and in alignment with enterprise governance standards.**

The introduction of a conversational, LLM-orchestrated agent—enabled through Retrieval Agent Manager (RAM), Retrieval-Augmented Generation (RAG), and Model Context Protocol (MCP) servers—represents a meaningful architectural evolution. Here, tools (encapsulated analytical programs) are exposed as governed services. The LLM does not replace code; rather, it acts as a reasoning layer that selects, sequences, and parameterizes validated tools at runtime. The intelligence lies not in rewriting analytics logic, but in orchestrating it dynamically.

Agentic flexibility does not eliminate the need for governance. On the contrary, it amplifies it. Hallucination risk, tool misuse, and execution variability require explicit guardrails, contractual input/output definitions, tool validation, execution logging, and audit trails. An enterprise-ready agent is not a free-form chatbot; it is a controlled orchestration engine operating within defined constraints.

For life sciences, regulatory analytics, and other high-stakes domains, this hybrid architecture offers a path forward: intelligent automation without sacrificing compliance; flexibility without abandoning lineage; conversational access without relinquishing control.

Ultimately, an “agent” should not be defined by novelty or trend, but by capability and accountability.

An agent is as an agent does.

And when designed with clear contracts, governed tools, explicit guardrails, and enterprise controls, an agent becomes not a buzzword—but a disciplined extension of analytic expertise.

The future of cohort construction and governed analytics is not purely deterministic, nor purely generative. It is intelligently orchestrated, rigorously governed, and architected for trust.

CONTACT INFORMATION

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

Author Name: Sherrine Eid, MPH
Company: SAS Institute, Inc
Address: 100 SAS Campus Drive, Cary, NC
Email: Sherrine.Eid@sas.com
Website: www.sas.com

Author Name: Mary Dolegowski, MQSM
Company: SAS Institute, Inc
Address: 100 SAS Campus Drive, Cary, NC
Email: Mary.Dolegowski@sas.com
Website: www.sas.com

Author Name: Sundaresh Sankaran
Company: SAS Institute, Inc
Address: 100 SAS Campus Drive, Cary, NC
Email: Sundaresh.Sankaran@sas.com
Website: www.sas.com