Paper SM07

Streamlining Report Reusability in SAS Viya: A REST API Approach

Mary Dolegowski, SAS Institute, Cary, NC, USA
David Weik, SAS Institute, Heidelberg, Germany

ABSTRACT

This paper presents a streamlined process leveraging SAS Viya REST APIs to create a reusable, repeatable method for leveraging existing SAS Visual Analytics reports and applying them to new data sources. To enhance accessibility for users unfamiliar with APIs, we encapsulated the code within a user-friendly Custom Step interface. This Custom Step can be seamlessly integrated into existing Flows to automate data processing and update SAS Visual Analytics reports, or used independently. Additionally, the code can be adapted as a macro, offering flexible implementation options to meet diverse user needs and workflows.

INTRODUCTION

Currently, SAS Viya Visual Analytics offers several ways to reuse dashboard elements, but each requires multiple clicks. We wanted to create a process that enables dashboard reuse without requiring users to manually navigate through Visual Analytics and complete a series of steps. The vision is simple: when a user uploads new data (whether scheduled or ad hoc) a process flow can automatically take that data, process it (if needed), and apply it to one or more reports. This ensures that, as soon as a user is ready to start visually interacting with the data, everything is already prepared. Minimal clicks required.

The code for the custom step that will copy and replace existing data for a Visual Analytics dashboard, shown in Figure 1, can be found at https://github.com/sassoftware/sas-studio-custom-steps.

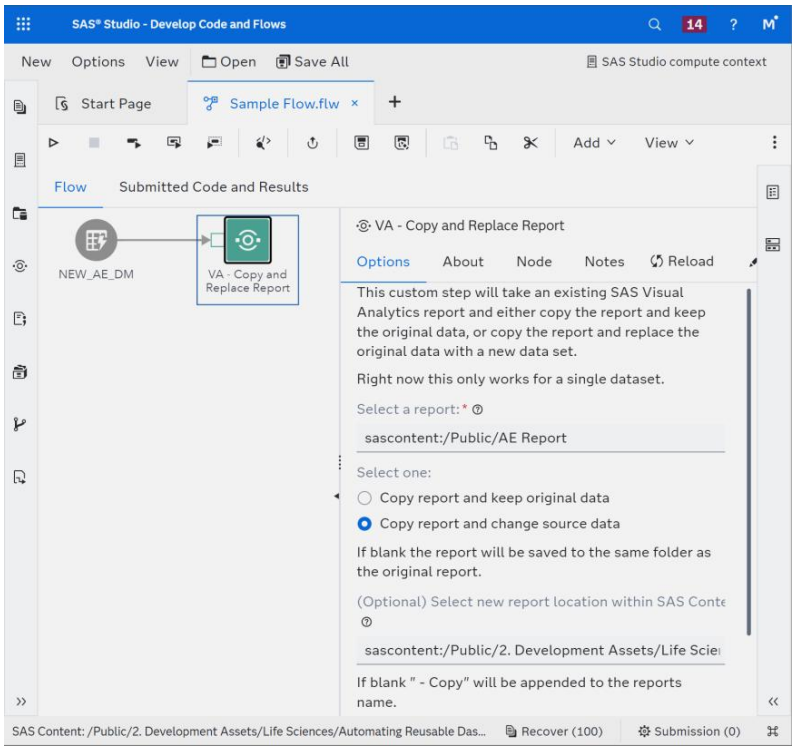

Figure 1: Sample Automation Flow

APIS SAS VIYA

Why start with APIS? We have found the SAS internal APIs to be very helpful when creating a programmatic flow that interacts with multiple domains within Viya. They provide an efficient and effective way for these domains to work

together and are the fastest method we have found for creating more complex, schedulable processes. All the available APIs for the Viya platform can be found here, https://developer.sas.com/, which includes detailed documentation and sample code.

One of the biggest hurdles people encounter when working with the APIs is determining where to get started. Which API domain will contain the details they need? For this process, since the end result is in Visual Analytics, we began in the Visualization and Reports domain and then the Visual Analytics subdomain. Working from the end result backward, we identified the API call to copy a report (Create a copy of a report), which also allows the location of the copied report to be set. Next, by using Update a report while applying the specified operation(s), we were able to take the copied report and apply new data. The next steps were to work backward further and identify all of the other API calls and data elements required to feed into the copy and update APIs. Figure 2 shows a flowchart highlighting all of the data entry points, the information that must be passed between API calls, and the final resulting report.

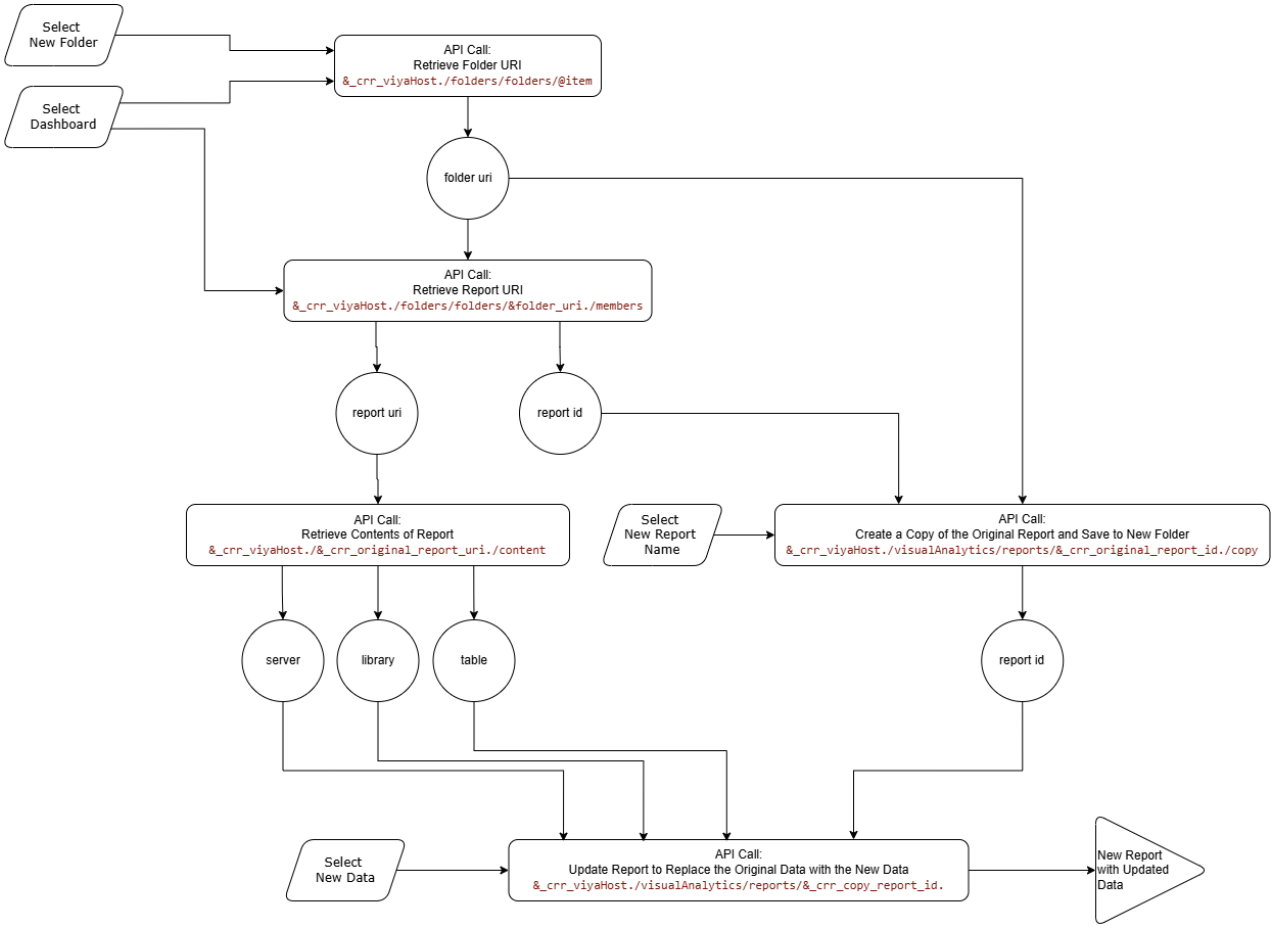

Figure 2: Overall Process Flow Chart

Once one starts looking into the other elements that need to feed into the process, it may become necessary to look outside the original API domain. For example, in this flow, two calls from the Folder domain, under Platform Administration are included: Get a folder with a path or a child URI and Get a list of folder members. These return the folder URI and report URI, respectively, which are required to tell the API which report should be copied and where the copied report should be saved. The report URI is the clearest example of information needed in subsequent calls, as it is a required part of the URLs.

EXAMPLE API BREAKDOWN

While Figure 2, shows the overall flow, each of the API calls require code similar to Code 1. The code in Code 1 was generated by referencing the code provided in the API documentation, and an example is shown in Figure 3. While we used SAS code due to personal preference, the Viya APIs can be interacted with using whichever API coding options works best for the application.

```
* API call to copy the original report and save it in the new folder;
* https://developer.sas.com/rest-apis/visualAnalytics/updateReportCopy;
proc http
    url = "&_crr_viyaHost./visualAnalytics/reports/&_crr_original_report_id./copy"
    method = 'Put'
    in = _crr_in
    out = _crr_out
    oauth_bearer = sas_services;
    headers
        'Accept' = 'application/json,
application/vnd.sas.report.operations.results+json,
application/vnd.sas.report.operations.error+json, application/vnd.sas.error+json'
        'Content-Type' = 'application/json';
run;
```

Code 1: SAS Code API– Copy and Save to New Folder

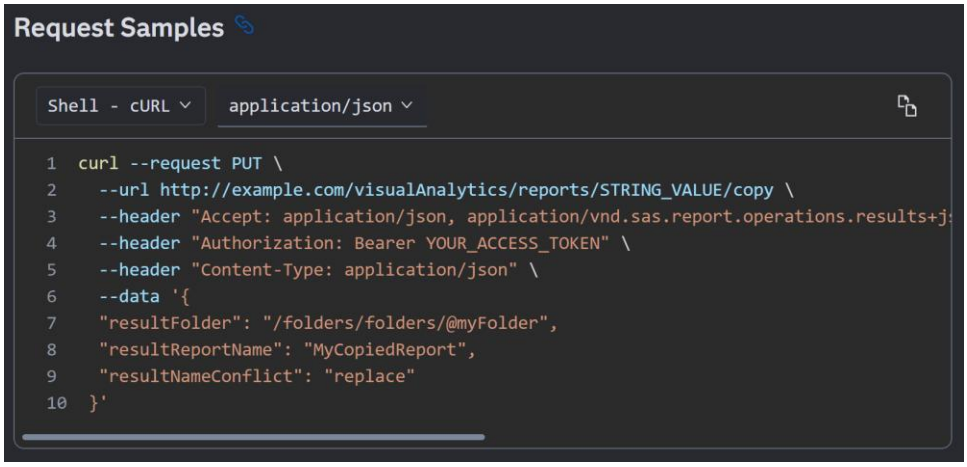

Figure 3: Documentation Sample Request – Copy and Save to New Folder

Note that while the example places the input data directly in the “data” section, our preference in the SAS code was to create an input file and reference it. This approach allows the file to be built programmatically and then easily double-checked, resulting in faster debugging.

It is also important to note that other languages can be used; however, a key benefit of SAS is that the authentication handshake does not need to be completed manually. This is the purpose of the statement “oauth_bearer = sas_services;” in all SAS code API calls. It uses the authentication token already created when the user logs into SAS, eliminating the need for a standard authentication handshake.

PUTTING IT ALL TOGETHER

Now that the process has been defined and the required fields identified, it is simply a matter of stringing together all of the API calls and creating the input and output files. After that, it comes down to user preference for how to provide access to these programs. Custom Steps use a GUI to reduce user error and can be combined into a schedulable flow for automation. Macros, on the other hand, are more commonly used and work well in traditional programming settings.

JSON

Earlier, we covered how to create the API calls in SAS code. In Code 2, we provide an example of one method for reading the out file and creating the in file. There are other ways to accomplish this, but this is the method used in the Custom Step. The section from `set stdJSON; to symputx('_crr_original_report_uri', report_uri, 'G');` reads the out file from an earlier API call. The first three lines read in the JSON data, while the next three iterate over the data and extract the original report URI. This URI is used to identify the report within the Viya system. The final three lines in this section create macro variables that will be used in the next step to help construct the in file for the copy API call.

The final three sections of Code 2 take the data pulled from other API calls and format it into a key-value (or dictionary) structure, allowing for a more seamless conversion into JSON. The first group of three lines performs the key-value formatting. The second group of two lines converts the data into JSON. The final two lines save the JSON output to the out file.

```
* Create the request body for the copy API call;
proc cas;
    set stdJSON;
    resp_file = readPath(&_crr_out_path.);
    resp_string = json2CASL(resp_file);

    do item over resp_string.items;
        report_uri = item.uri;
    end;

    report_id = scan(report_uri, 3, '/');
    symputx('_crr_original_report_id', report_id, 'G');
    symputx('_crr_original_report_uri', report_uri, 'G');

    in_data.resultFolder = "&_crr_new_folder_uri.";
    in_data.resultReportName = "&_crr_name_update.";
    in_data.resultNameConflict = 'replace';

    in_json = casl2JSON(in_data);
    in_json = compbl(in_json);

    file outfile &_crr_in_path.;
    print in_json;
run;
```

Code 2: Sample JSON Reading and Creation Code

CUSTOM STEP

Custom Steps are useful for packaging up code into a more user-friendly GUI. Figure 4, shows one of the two user experiences for Custom Steps. This view is used when just the process contained in the Custom Step needs to be run.

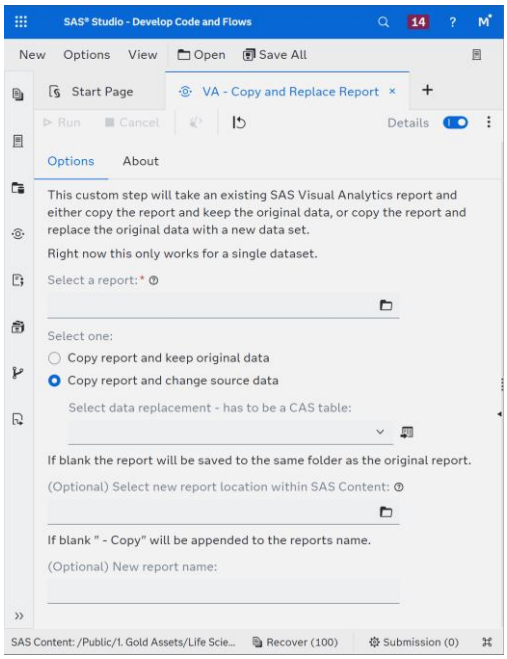

Figure 4: Visual Analytics – Copy and Replace Report 1

Figure 5 shows the other user experience for Custom Steps. The benefit of this option is that users can daisy-chain processes together to create more complex processes that can then be scheduled, copied, and updated with minimal effort.

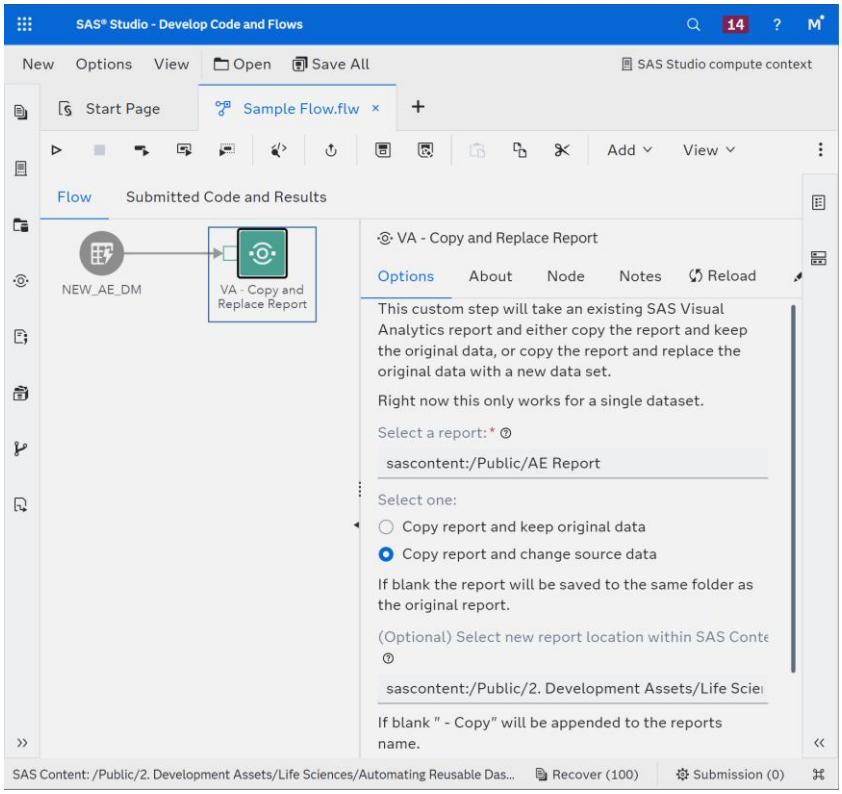

Figure 5: Visual Analytics – Copy and Replace Report 2

CONCLUSION

While at first glance APIs might be dauting, they are also a valuable and dynamic option for automating processes within SAS Viya. Our goal has been to highlight what tools are available to make these more accessible. Programmers can go to https://developer.sas.com/, to find valuable documentation and examples. hey can then provide end users with multiple methods for accessing these processes, allowing them to benefit without needing to understand the underlying code.

REFERENCES

Calling REST API's from SAS : Using proc http to extract report content. (2025, August 26). https://communities.sas.com/t5/SAS-Communities-Library/Calling-REST-API-s-from-SAS-Using-proc-http-to-extract-report/ta-p/721209

Copilot. (2026). Response to "Proofread for grammar and spelling mistakes" (February 9, 2026). Microsoft. https://copilot.microsoft.com/.

sassoftware. (n.d.). GitHub - sassoftware/sas-studio-custom-steps: Repository to share SAS Studio Custom Steps. A custom step enables you to create a user interface for SAS Studio users at your site to complete a specific task. GitHub. https://github.com/sassoftware/sas-studio-custom-steps

Wang, C. (2020, September 16). *Using REST API to transform a Visual Analytics Report - SAS Users*. SAS Users. https://blogs.sas.com/content/sgf/2020/09/15/using-rest-api-to-transform-a-visual-analytics-report/

Weik, David. (2025, April 22). *SAS Visual Analytics API* [Video]. YouTube. https://www.youtube.com/watch?v=b3mAOTfeTe4

ACKNOWLEDGMENTS

Thank you Jim Box and Matt Becker for your advice and reference points, along with the rest of the SAS Life Sciences Team.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:
Author Name: Mary Dolegowski
Company: SAS Institute
City: Cary, NC, USA
Email: mary.dolegowski@sas.com

Author Name: David Weilk
Company: SAS Institute
City: Heidelberg, Germany
Email: David.Weik@sas.com