PAPER BB09-2014

All Your Scripts are Belong to Us: Extending SAS with Server-Side Scripts

Ben Elbert, Alliance Data, Columbus, OH

ABSTRACT

Even if one has previously developed code (in a language that must not be named) or has determined that some data acquisition procedures are easier solved outside SAS, SAS need not be left on the sideline to watch. By leveraging built-in SAS functionality such as URL calls, XML processing, and the beloved Macro Language, we demonstrate how SAS can acquire data produced by server-side scripts. By interacting with some simple scripts, we will demonstrate how automated tools can remain automated and not require one to leave the comfort of their SAS environment. For the adventurous SAS programmer, the techniques described in this paper easily extend to cases where we wish to call C, Perl, Python, or other scripts on a server to accomplish a multitude of tasks without the hassle of converting code or switching between environments, which is especially beneficial in an enterprise scenario where resource constraints may stipulate that developed processes be left as is.

INTRODUCTION

Nearly all enterprise environments include entities which have been built on a variety of different languages and platforms. In most cases it is not feasible to recode solutions into SAS, but this hurdle does not mean that those solutions have to be executed outside of SAS. Whether external solutions are already callable via web services or can be made to do so, built-in SAS functionality such as FILENAME URL, LIBNAME XML, and the SAS Macro Language enables us to access these external processes without having to leave the comfort of our SAS environment. In this paper we will not only introduce readers to the techniques and framework needed to leverage server-side scripts while remaining in session, but also provide helpful tips for making SAS's interaction the server more understandable by the user.

SERVER-SIDE SCRIPTS

We will not provide a complete overview of server-side scripts, but rather a brief introduction that will serve for our needs. Additionally, we avoid too detailed of a survey so that the reader may be imaginative in the ways that SAS can interact with these external programs. Moreover, once a SAS developer learns the basics of controlling programs that can be executed from a shell or command console from a script language, they will find that there are few programs or functions which cannot be utilized by their SAS session.

A server-side script is a script that dynamically generates HTML pages on the server rather than on the client side, and for our purposes we are most interested in server scripts that produce XML pages as their response. Server scripts are able to handle variable parameters which are typically passed in their call, which takes the form of a web address. For example, to call the script TEST_SCRIPT on the server ZEUS we would submit

https://ZEUS/TEST_SCRIPT.cgi?var1=2&var2=4

where VAR1 and VAR2 are parameters that will be passed to the script. Note that the script name is separated from the parameters list by a **?** and the parameters are separated by **&**. SAS will produce a warning in the LOG when addresses like this are used due to the **&** typically indicating that a macro variable is being used. For this reason, it is important to make sure that you do not have macro variables which share names with parameter names that you will be using in your script calls.

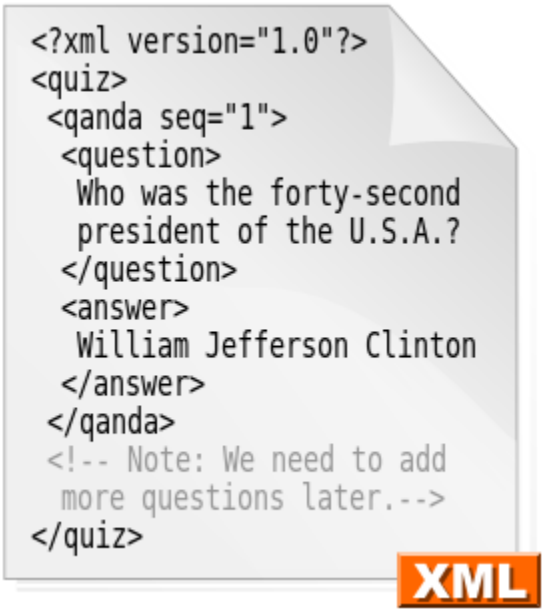
There are many languages in which you can create server scripts and we will not go through them nor generate examples of them, but we will discuss for a moment what we recommend for the XML results of whichever script you have. Let us consider them as 4 rules to follow:

(1) Have a method for handling duplicate requests (this will be important later)
(2) Provide only 1 XML record in your response (SAS code can be developed to log all of these responses if desired)
(3) Have a STATUS variable that indicates whether the process is working, is finished, or has reached an error
(4) Provide a Response Key to the user when they initialize the primary script so that they can request status updates

With this cursory introduction to viewing server-side scripts from a SAS point of view out of the way, let us now explore how SAS can interact with these programs.

THE XML STRUCTURE

If one has not come across XML, it is best to think of it as a way to represent a table (or object, in object oriented programming) in a tree format that is easy to read by both humans and machines. By using a PARENT (or ROOT) wrapper and CHILD (or SUB) elements, the XML format enables one to encode data so that a given parent object is able to contain data which pertains to that parent.



Example of XML Document
(http://commons.wikimedia.org/wiki/File:XML.svg)

The SAS XML LIBNAME Engine is able to parse the XML map formed by the document, and will create a dataset for each table contained within the document wrapper (in the example above, a data set would be created for each QANDA section, with two variables: QUESTION and ANSWER.

For more information about XML documents and how they are read in by SAS, see the SAS XML LIBNAME Engine User Guide for your version of SAS.

PUTTING IT ALL TOGETHER

When we go to call a script from within SAS, we need to do three things:

(1) Declare a FILENAME with URL option that is the address of the script you wish to call and with all needed parameters set
(2) Establish an XML Library using the LIBNAME XML protocol where the library name is the same as the short name given to the FILENAME from step (1)
(3) Read the XML result into a SAS data set by setting the data set created in the XML library

Thus, if we wish to obtain the XML result of calling a script from within SAS, we would have SAS code as such

```
FILENAME test url "WEB-SCRIPT-ADDRESS";
LIBNAME test xml;

DATA script_result;
SET test.RESULT;
RUN;
```

It may take some time for the code to run depending on how long it takes the script to run, but if one has an initialization script and then scripts to check the progress and get the results the XML result from the initialization may only be a small step.

For a thorough treatment of the many uses for the FILENAME statement, consider Schacherer (2012).

An ideal way of handling processes which are initiated by scripts but may take some time to run is via an algorithm such as

(1) Run INITIALIZATION script from within SAS
(2) Was there an ERROR?
    a. If YES, quit and debug
    b. If NO, go to Step 3
(3) Check the status of the program with another script
    a. If ERROR, quit
    b. If FINISHED, go to Step 4
    c. If RUNNING, wait for determined time and repeat Step 3
(4) Run RETRIEVAL Script or Continue with Further Code

To better illustrate this algorithm, we provide a useful macro which, after being slightly modified for one's own scripts and data retrieval steps, is very useful for allowing SAS to interact with server-side scripts.

THE CALL_SCRIPTS MACRO

```
%macro call_scripts(var_1=,var_2=,sleep_time=);

%let addr = https://SERVER/SCRIPT-DIR;

FILENAME init url "&addr./init_script.cgi?var1=&var_1.&var2=&var_2.";
LIBNAME init xml;

DATA init_result;
SET init.RESULT;
if init_status='COMPLETE' then do;
       call symput('init_status',init_status='COMPLETE');
       call symput('reqkey',trim(left(req_key)));
end;
else do;
       call symput('init_status',init_status='COMPLETE');
end;
RUN;

%if &init_status.=1 %then %do;
       %macro get_status(req_key);
       filename progress url "&addr./status_script.cgi?request_key=&req_key.";
       libname progress xml;

       data status1;
       set progress.RESULTS;
       run;
       %mend;
       %get_status(&reqkey.);
       data _null_;
       set status1;
              call symput('cur_stat',status='COMPLETE');
              call symput('err_stat',status='ERROR');
              call symput('pct_done',pct_done);
       run;
       %put WARNING: &PCT_DONE. OF THE PROGRAM HAS PROCESSED...;
       %do %while(&cur_stat. NE 1 AND &err_stat. NE 1);
              data _null_;
```

```
                slept=sleep(&sleep_time.);
                run;
                %get_status(&reqkey.);
                data _null_;
                set status1;
                       call symput('cur_stat',status='COMPLETE');
                       call symput('recs_done',records_processed);
                run;
                %put WARNING: &recs_done OF &recs HAVE BEEN PROCESSED...;
        %end;

        %if &err_stat.=1 %then %do;
                %put ERROR: THERE WAS AN ISSUE WITH THE PROGRAM;
        %end;
        %else %do;
                filename rtrv url "&addr./rtrv_script.cgi?request_key=&req_key.";
                libname rtrv xml;

                data status2;
                set rtrv.RESULTS;
                call symput('xfr_status',xfr_status='COMPLETE');
                run;

                %if &xfr_status.=1 %then %do;
                       /* INSERT DATA RETRIEVAL CODE HERE */
                %end;
                %else %do;
                       %put ERROR: THERE WAS AN ISSUE RETRIEVING THE DATA;
                %end;
        %end;
%end;
%else %do;
        %put ERROR: THE INITIALIZATION FAILED!;
%end;
%mend;
```

We mentioned previously that it is necessary to ensure your script is able to recognize duplicate requests and the reason for this is due to the Program Data Vector (PDV) of SAS Base Certification fame. When we call a script using the DATA/SET process of setting an XML library data set, the PDV first calls the script to determine the fields and format for the variables and then the script is called once more as the data is fed into the data set. To enable the script to be called outside of SAS it is advisable to put a check in place that determines whether the script has been called *too quickly* and directs the request to the response given to the first call.

The CALL_SCRIPTS macro currently overwrites the progress data set, but an extra PROC APPEND step could also create a LOG data set which tracks the progress and may make it easier to debug if there is an error. Also note that there is no current timeout procedure to check whether or not the process has hit an infinite loop as we assume these considerations have been accounted for in the script of server based program.

CONCLUSION

As we have seen, it is not only possible but also relatively easy to interact with server-side scripts from within the SAS environment. The larger hurdle, of course, is the development of control scripts if they do not exist already, but it is typically much faster to develop these scripts than it is to recode a process in SAS. Further extensions of this process include calling C, Perl, and Python with scripts and easy web scraping to bring non-SAS data into SAS more easily.

REFERENCES

Schacherer, C. (2012). The FILENAME Statement: Interacting with the world outside of SAS®. Proceedings of The SAS Global Forum 2012. Cary, NC: SAS Institute, Inc.

ACKNOWLEDGMENTS

A great deal of thanks goes out to John Williams, Alliance Data, whose own scripts providing ample experimentation and practice leading up to this paper.

CONTACT INFORMATION

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

Ben Elbert
Alliance Data
3100 Easton Square Place
Gahanna, OH 43219
ben.elbert@alliancedata.com