Paper 1668-2014

Generate Cloned Output with a Loop or Splitter Transformation

Laura Liotus, Community Care Behavioral Health, Pittsburgh, PA

ABSTRACT

Based on selection criteria, the SAS® Data Integration Studio loop or splitter transformations can be used to generate multiple output files. The ETL developer or SAS administrator can decide which transformation is better suited for the design, priorities and SAS configuration at their site. Factors to consider are the set-up, maintenance and performance of the ETL job. The loop transformation requires an understanding of macros and a control table. The splitter transformation is more straight-forward and self-documenting. If time allows, creating and running a job with each transformation can provide benchmarking to measure performance. For comparison of these two options, this paper shows an example of the same job using the loop or splitter transformation. For added testing metrics, one can adapt the LOGPARSE SAS macro to parse the job logs.

INTRODUCTION

Often you will hear programmers say there are multiple ways to skin a cat. Just as we are all individuals, we all have our individual style when attacking a programming challenge. Depending on the situation at one's workplace, the priority of the job design will determine the programming path. Due to time constraints, efficiency might take precedence. Perhaps having a self-documented job for a descendent is the goal. Fortunately, SAS Data Integration Studio provides the developer several transformation choices to allow for flexibility in the ETL job design.

TRANSFORMATION CHOICE

Community Care Behavioral Health, a managed care organization, has several lines of business with similar reporting needs. For the particular SAS Data Integration Studio project used in the example for this paper, we produce output files containing claims data. Since the files are of the same format for each business unit, I wrote a job using the loop transformation that includes a control table to produce the cloned files. This was one of my first ETL projects using SAS Data Integration Studio and quite a few years ago. As there was not much documentation on the loop transformation at the time, SAS Technical Support was my main source of inquiry and as always, they provided outstanding assistance.

After creating, implementing and running the loop job on a weekly basis, I attended a SAS training session, *Using SAS Data Integration Studio to Create Efficient ETL Processes.* The purpose of this course was to present methods to improve ETL job performance. After receiving this training in ETL efficiency, I wanted to put my knowledge to use. One of my goals was to compare the performance of the loop transformation I was currently using to produce files, with a splitter transformation. Just as we did in class, some benchmarking would be performed to measure the runtime differences. As time allowed, I finally got around to accomplishing this goal.

In the Transformations tab of SAS Data Integration Studio, the loop transformation is listed under Control and the splitter transformation is listed under Data, Figure 1.

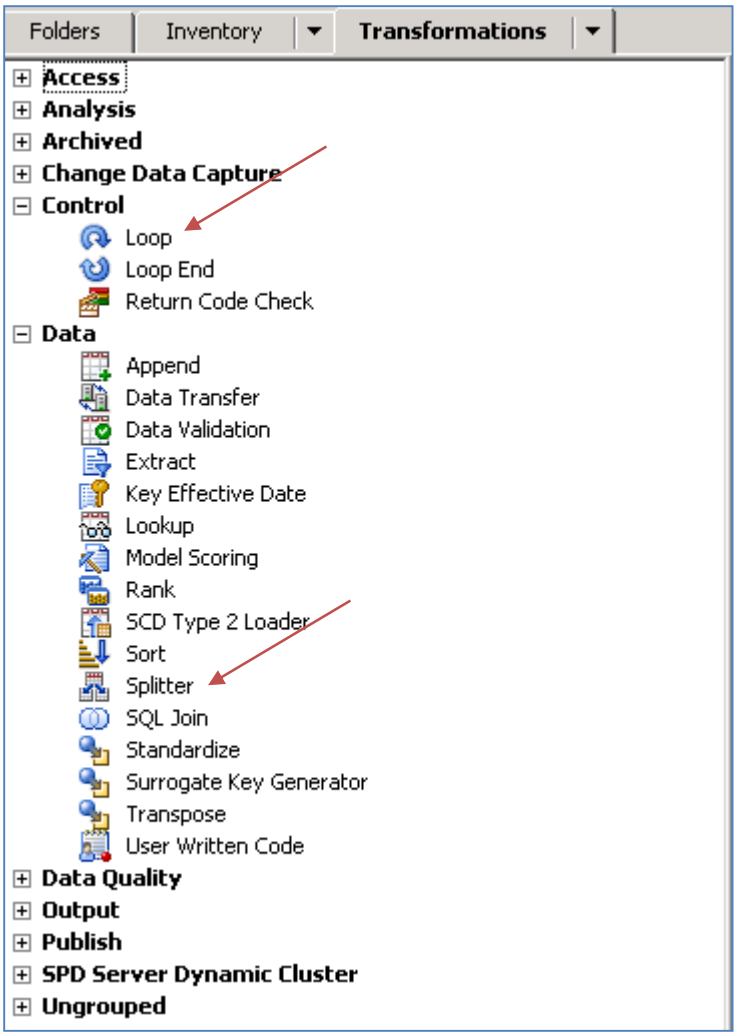


Figure 1 – Transformations Tab

LOOP WITH CONTROL TABLE

As stated above, at Community Care, we use a loop job to produce cloned output files for each of our lines of business, Figure 2.

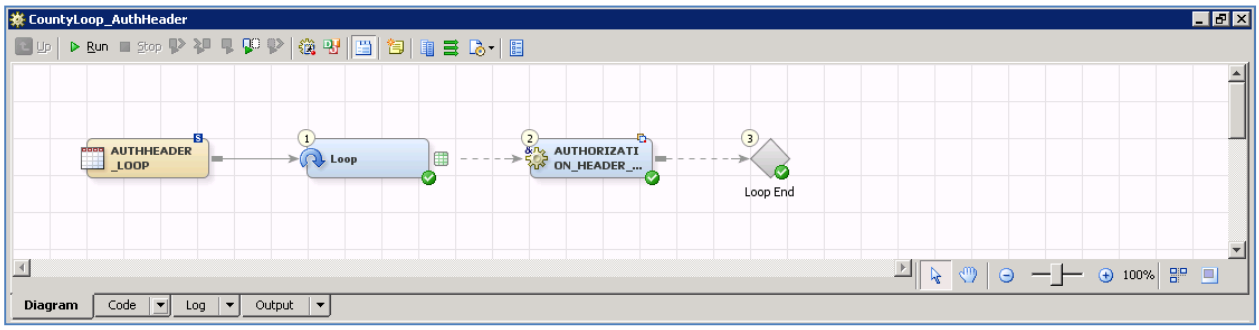
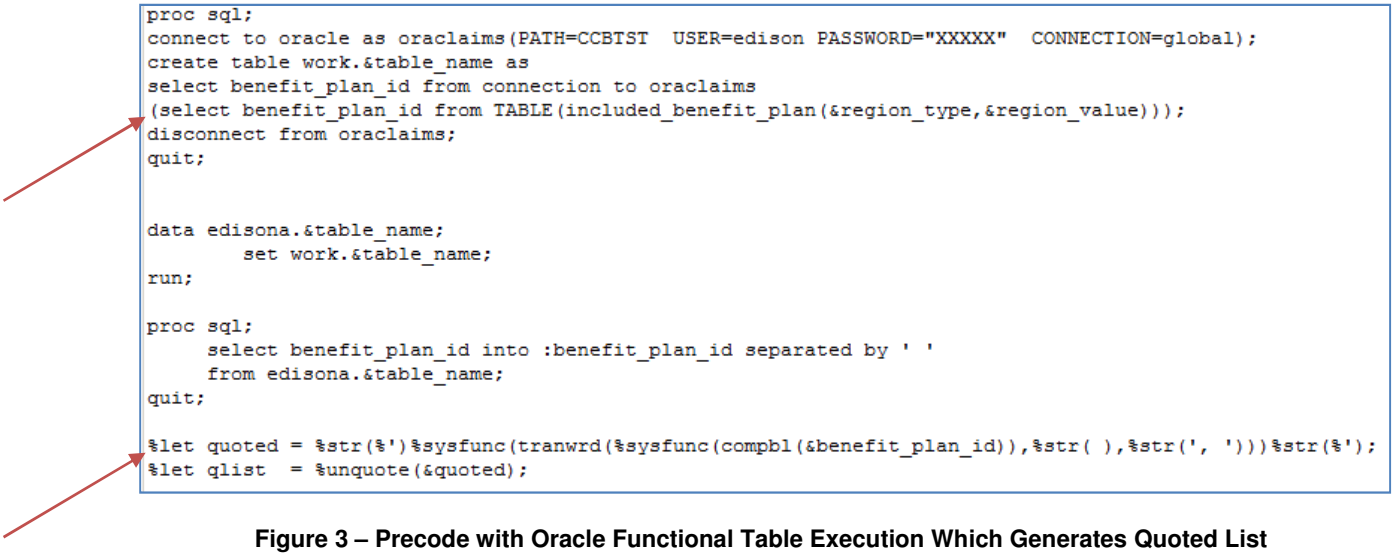


Figure 2 – Loop Job

In addition to the parameters in a control table, the loop job derives part of the parameter data, dynamically, from an Oracle functional table. Due to how Community Care classifies the lines of business, the Oracle function selects units from the functional table. The Oracle function is executed in the precode of the main ETL job that is called by the loop job. Also included in this precode, is syntax to string the results of the function together, with commas and single quotes for use in main ETL job, Figure 3.



```
proc sql;
connect to oracle as oraclaims(PATH=CCBTST  USER=edison PASSWORD="XXXXX"  CONNECTION=global);
create table work.&table_name as
select benefit_plan_id from connection to oraclaims
(select benefit_plan_id from TABLE(included_benefit_plan(&region_type,&region_value)));
disconnect from oraclaims;
quit;


data edisona.&table_name;
        set work.&table_name;
run;

proc sql;
     select benefit_plan_id into :benefit_plan_id separated by ' '
     from edisona.&table_name;
quit;

%let quoted = %str(%')%sysfunc(tranwrd(%sysfunc(compbl(&benefit_plan_id)),%str( ),%str(', ')))%str(%');
%let qlist  = %unquote(&quoted);
```

Figure 3 – Precode with Oracle Functional Table Execution Which Generates Quoted List

Results of the function that is run in the precode, are passed to a SAS macro variable which is placed in an IN clause of the WHERE in a JOIN object, Figure 4. See Appendix A for output results in the SAS log. Other, more straight-forward loop parameters are stored in the control table of the loop job, Figure 5.

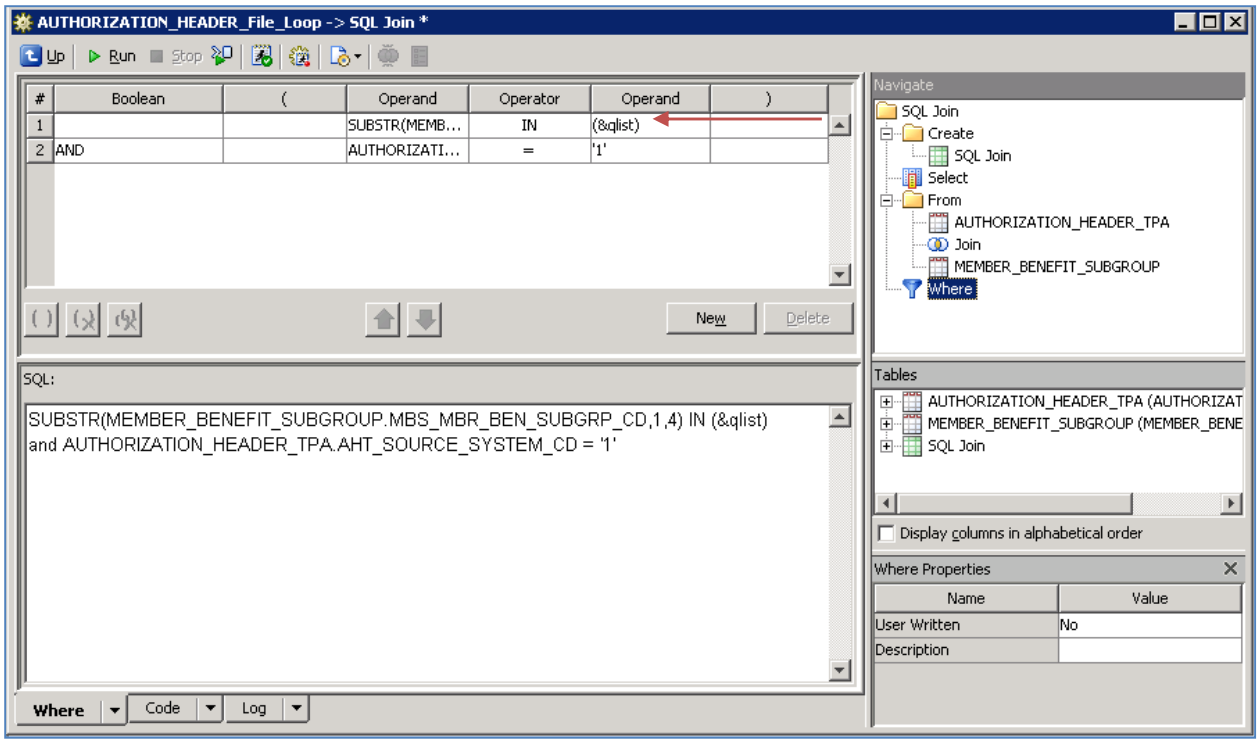


Figure 4 – Macro Variable in WHERE Clause

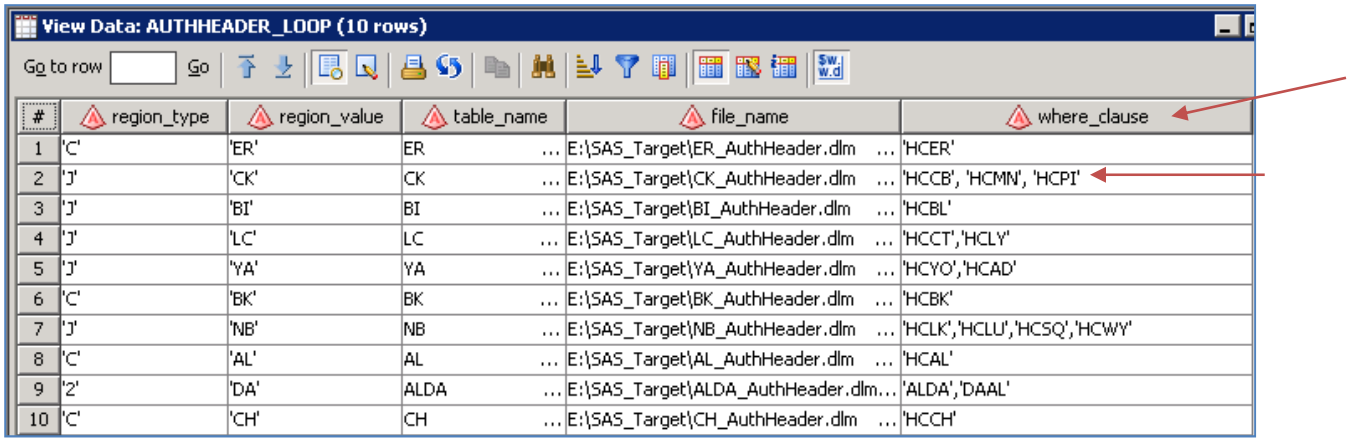


Figure 5 – Loop Control Table Including Where Clause with Literals for Parallel Processing

LOOP LESSONS LEARNED

For the single server configuration used at Community Care, when using the Oracle functional table that creates the macro variable to build the WHERE clause, the loop options should not execute in parallel. This job also needs to wait for all processes to complete before continuing. If one deselects 'Execute iterations in parallel' the single-threaded defaults will be set automatically, Figure 6.

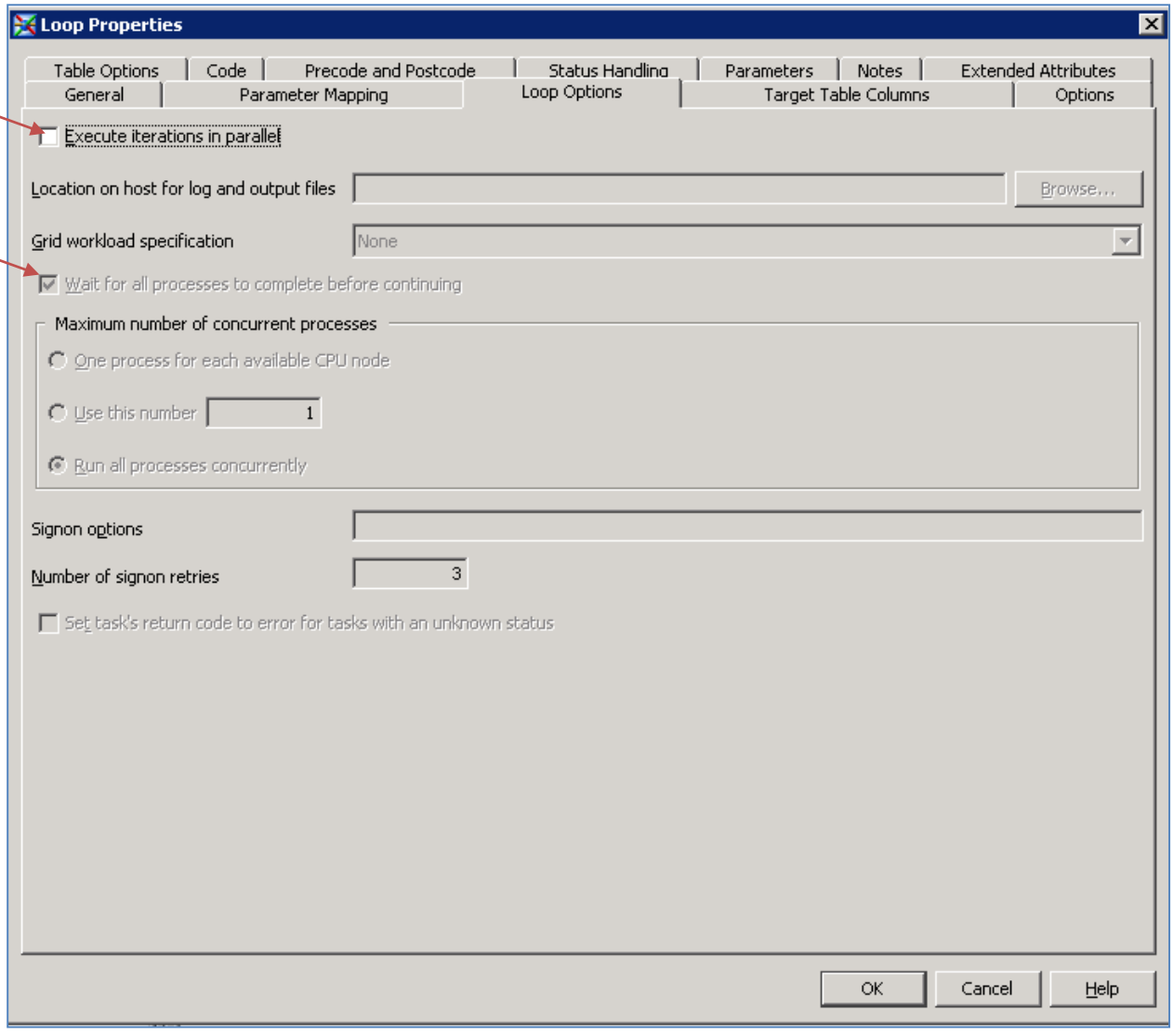


Figure 6 – Loop Properties for Single-Threaded and Parallel Options

If the SAS server contains multiple processors, the option, 'Executing iterations in parallel', is available in a single server environment. This is possible with symmetric multiprocessing (SMP). Community Care has a single server with multiple processors. Therefore, to determine if performance could be improved, I decided to experiment with running the loop jobs in parallel. With the parallel option selected, I initially tried using the same set-up as used in the single-threaded loop job, including the macro from the Oracle functional table described above. When using the functional table call in the parallel set-up, no rows were selected. This is due to the inclusion of RSUBMIT in the generated SAS Data Integration code for a job with parallel processing selected. Due to the combination of RSUBMIT and quotes added to the string of the automatically generated variables, the WHERE clause contained the literal SAS code from the precode of the main ETL job. I was able to figure this out by monitoring the actual database query using Oracle Enterprise Manager, Figure 7. Therefore, I was forced to use literal values in the control table instead of dynamic values produced from the Oracle function, Figure 5.

```
where SUBSTR(CLAIM.CLM_GROUP_ID_RAW,1,4) IN ('%sysfunc(tranwrd(%sysfunc(compbl(&benefit_plan_id)), ,',
'))') and CLAIM.CLM_SOURCE_SYSTEM_CD = '1'
```

Figure 7 – Literal Passed to SAS with RSUBMIT

You can find more detail about RSUBMIT as it relates to a distributed system in this document: http://support.sas.com/techsup/technote/ts697.pdf

To summarize the parallel test findings, in order to run jobs in parallel I could not use the Oracle function. Therefore, for parallel processing in the single server environment, I had to place literal values in the control table to be passed to the WHERE clause, Figure 5. Using these literals, the parallel execution in the single server configuration worked with either option, waiting for all processes to finish or a multi-threaded approach. Obviously, the literals require a little more maintenance upon adding new lines of business as opposed to the Oracle function. The only situation where this did not work is in a larger, more complicated JOIN. This was probably a database limitation as opposed to a SAS limitation.

If you have a distributed environment, you could test the passing of an automatically generated macro string. This combined with the RSUBMIT embedded in a macro might work in a true distributed configuration with SAS/CONNECT.

Another lesson learned with the loop job configuration at Community Care, was the requirement of the output file transformation to be specified with double-quotes on the file, Figure 8. These are needed in either single-threaded or parallel processing. This is due to the file_name parameter entries in the control table that are surrounded by single quotes, which are then used as a macro variable for the File Location, Figure 5 and Figure 8.

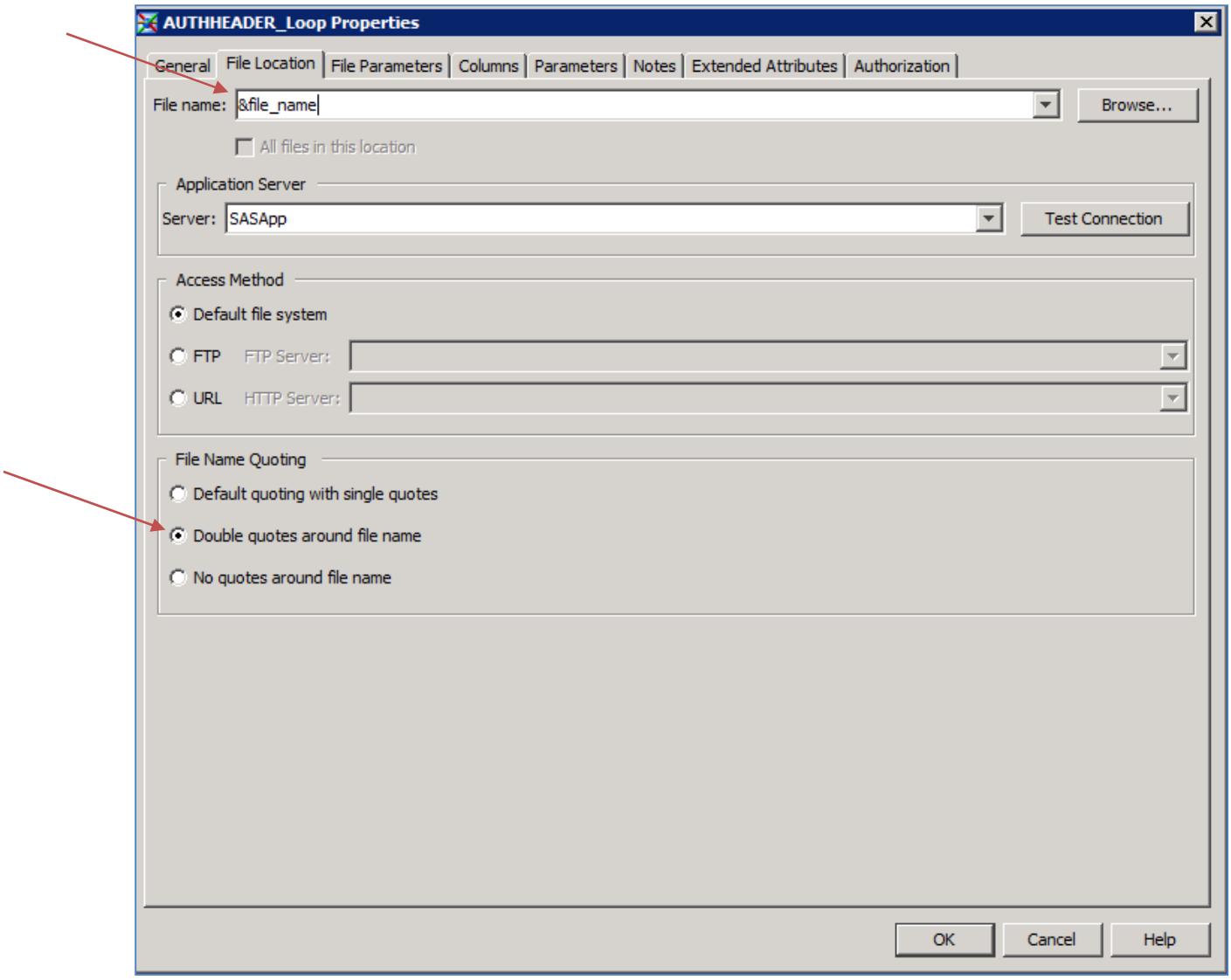


Figure 8 - Output File Transformation

LOOP ADVANTAGES

At Community Care, the job with a loop uses a control table to drive a pass of the process for each line of business. Whether using the more complicated database function or literals in the control table, the advantage of the loop is that only one record needs inserted into the control table, for a new line of business. Also with the loop, there is one output object. If a file specification changes, only one object needs maintained for the change to occur. The loop also makes adding a line of business much more seamless than adding another object when new business is added as required by a splitter.

SPLITTER

The multiple lines of business that require the cloned output or differing output, represents a perfect opportunity to apply the splitter transformation. The splitter transformation is much easier to set-up than a loop, considering macro variables and the more complex Oracle functional table in use at Community Care. The splitter is also self-explanatory therefore self-documenting, Figure 9.

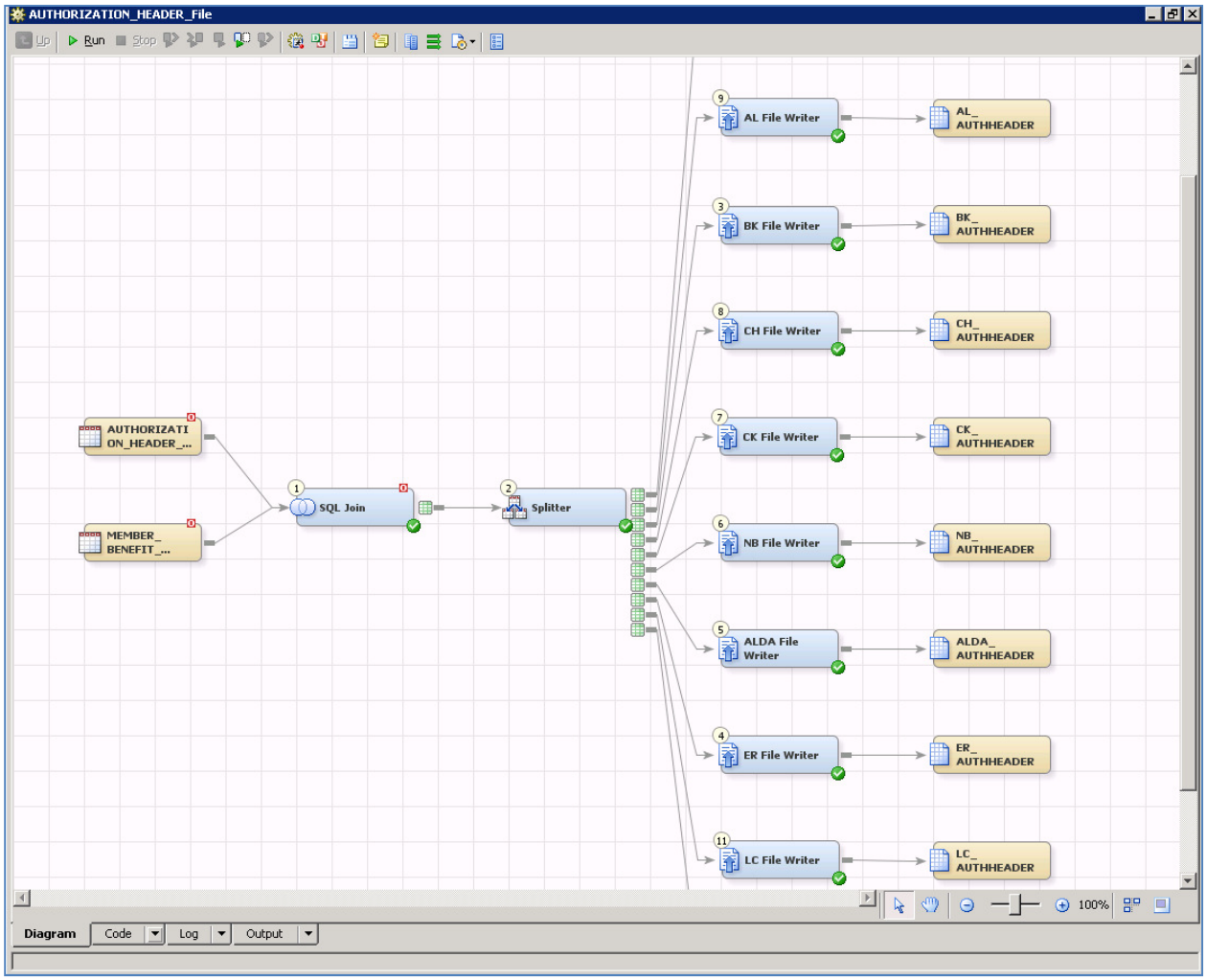


Figure 9 - Splitter Job

Creating a job with a splitter is not complicated. The challenge comes in when a new line of business is added and all of the manual processes that go along with adding the line of business to the splitter. Also, if the output file format requires a change, all output file objects in the job must be updated.

SPLITTER LESSONS LEARNED

In the automatically generated code, verify that the splitter does not contain PROC SQL and a DATA step. The job should use the DATA step only. If the transformation contains both query types, the mappings need reviewed to eliminate any unnecessary mappings.

SPLITTER ADVANTAGES

The splitter transformation is easy to set-up and self-documenting. There is no need to be concerned with passing macros in either the value or literal form. It is obvious from the ETL diagram and code what process the splitter is performing. If your files are of differing formats as opposed to clones, a splitter is the more logical choice.

Even with the maintenance overhead of the splitter, the curiosity after attending the class was to determine which runs faster, the loop running the job single-threaded and/or in parallel or the splitter running the output simultaneously. The results might surprise you.

LOG JOB TIME

For a simple test, one can record timings of both types of jobs manually using the automatically generated logs. If many iterations are required one might want to research using the LOGPARSE SAS macro described in Paper 219-30. Michael A. Raithel. “Programmatically Measure SAS Application Performance On Any Computer Platform With the New LOGPARSE SAS Macro“. http://www2.sas.com/proceedings/sugi30/219-30.pdf

This macro can be adapted to the SAS Data Integration Studio environment. One has to place PASSINFO and FULLSTIMER in the job options, Figure 10.

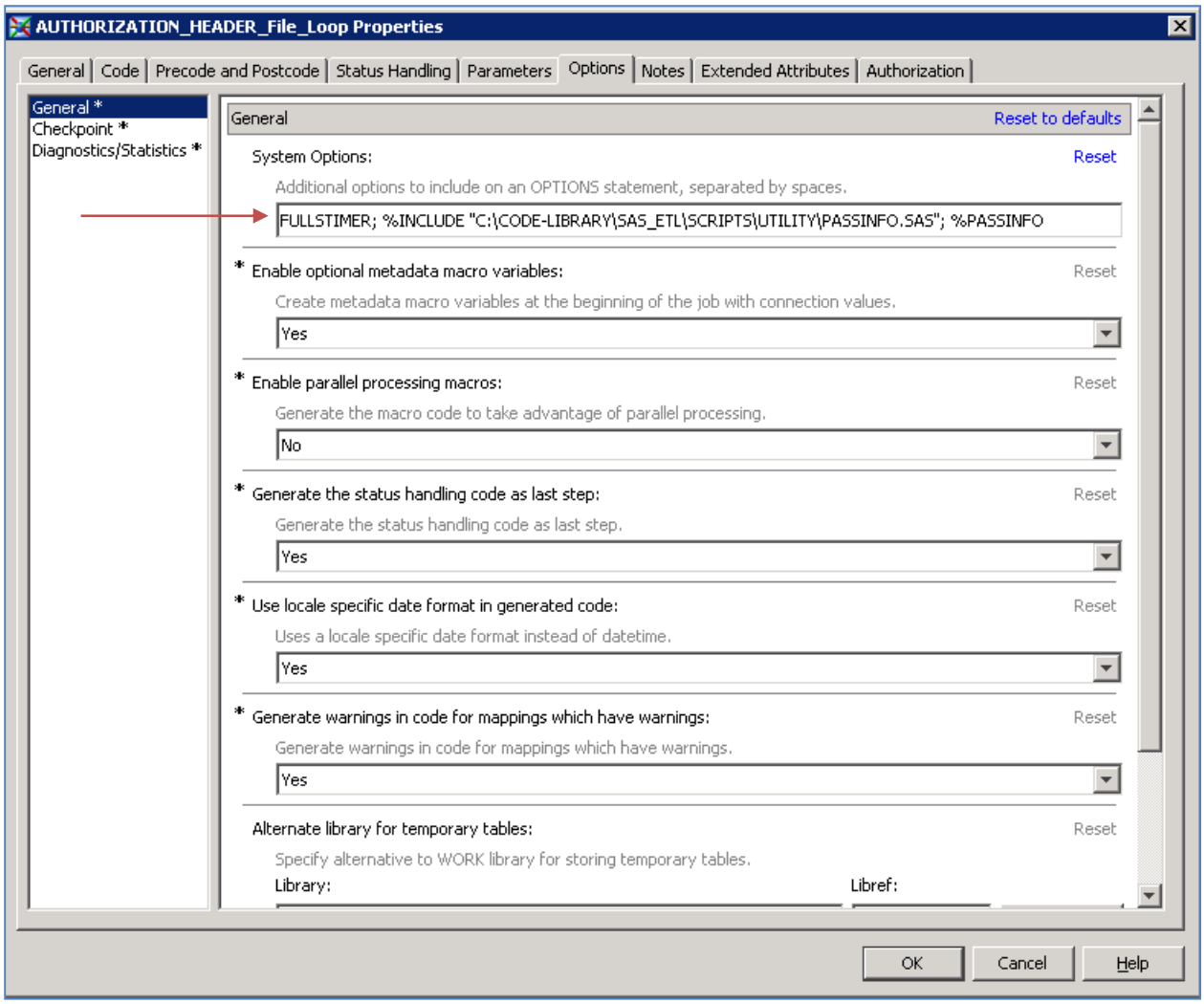


Figure 10 – LOGPARSE in SAS Data Integration Studio

If one wants to collect performance data for multiple SAS Data Integration Jobs, the PASSINFO and LOGPARSE macros can be placed in the AUTOCALL library. Using BASE SAS, one can parse the log for performance data, Figure 11.

```
%include "C:\code-library\SAS_ETL\scripts\Utility\logparse.sas";
%logparse(F:\TuneLog\DiJob.log, perfdata, OTH );
proc print data=perfdata;
run;

data windows / view=windows; /*Logparse view for Windows systems */
set perfdata(keep=logfile stepname portdate platform realtime
usertime systime cputime obsin obsout varsout
datetime host memused osmemused stepcnt);
run;
```

Figure 11 – Performance Data

CONCLUSION

Experimenting with the various transformations of SAS Data Integration Studio to determine the most efficient process provides a learning opportunity for the SAS developer or administrator. The loop and control table ETL job requires less maintenance upon adding or changing specifications. The splitter is straight-forward and self-documenting. In conclusion, for large queries, the loop transformation with the control table, performed the most optimally. For smaller output files, there was not much difference in time between the loop and splitter. Surprisingly, in a single server environment, the splitter out-performs the loop for smaller output files. As Community Care currently produces eleven cloned output files and continues to acquire new business, due to the maintenance factor of a splitter, the loop remains the transformation choice at Community Care, regardless of timing. The ultimate choice of a solution depends on the priorities at your site, whether that be maintenance, performance or leaving a self-documented job that can easily be inherited by the legacy that follows in your SAS footsteps.

CONTACT INFORMATION

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

Laura Liotus
Community Care Behavioral Health
One Chatham Center
Suite 700
Pittsburgh, PA 15219
412-402-8713
liotuslg@upmc.edu
http://www.ccbh.com/

Appendix A - Log from Precode

```
41   %let region_type = 'J';
42   %let region_value = 'CK';
43   %let table_name = CK;
44   %let file_name = E:\SAS_Target\AuthHeader.dlm;
45
46
47   LIBNAME edisona ORACLE  DIRECT_SQL=NO  PATH=CCBTST  SCHEMA=edison  USER=edison
47 ! PASSWORD=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX ;
NOTE: Libref EDISONA was successfully assigned as follows:
      Engine:        ORACLE
      Physical Name: CCBTST
48   proc sql;
49   drop table edisona.&table_name;
NOTE: Table EDISONA.CK has been dropped.
50   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.15 seconds
      cpu time            0.00 seconds
51
52
53
54
55   proc sql;
56   connect to oracle as oraclaims(PATH=CCBTST  USER=edison
56 ! PASSWORD=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  CONNECTION=global);
57   create table work.&table_name as
58   select benefit_plan_id from connection to oraclaims
59   (select benefit_plan_id from TABLE(included_benefit_plan(&region_type,&region_value)));
NOTE: Table WORK.CK created, with 3 rows and 1 columns.
60   disconnect from oraclaims;
61   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.14 seconds
      cpu time            0.01 seconds
62
63
64   data edisona.&table_name;
65       set work.&table_name;
66   run;
NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
NOTE: There were 3 observations read from the data set WORK.CK.
NOTE: The data set EDISONA.CK has 3 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.17 seconds
      cpu time            0.00 seconds
67
68   proc sql;
69        select benefit_plan_id into :benefit_plan_id separated by ' '
70        from edisona.&table_name;
71   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
72
73   %let quoted = %str(%')%sysfunc(tranwrd(%sysfunc(compbl(&benefit_plan_id)),%str( ),%str(',
73 ! ')))%str(%');
NOTE: Line generated by the macro function "SYSFUNC".
1    HCCB', 'HCMN', 'HCPI
       ----   ----
       49     49
NOTE 49-169: The meaning of an identifier after a quoted string may change in a future SAS release.
             Inserting white space between a quoted string and the succeeding identifier is
             recommended.
74   %let qlist  = %unquote(&quoted);
NOTE: Line generated by the macro variable "QUOTED".
1    'HCCB', 'HCMN', 'HCPI'
       ----    ----
       49      49
NOTE 49-169: The meaning of an identifier after a quoted string may change in a future SAS release.
             Inserting white space between a quoted string and the succeeding identifier is
             recommended.
75
76   %put &qlist;
'HCCB', 'HCMN', 'HCPI'
```