Paper SAS 0927-2017

Improving Efficiency in SAS® Enterprise Guide®:

Parallel Processing and Other Hidden Gems

Benjamin First, US Bank; Steven First, Systems Seminar Consultants

ABSTRACT

In the past 15 years, SAS® Enterprise Guide® (EG) has developed into the go-to application to access the power of SAS®. With each new release, SAS continues to add functionality that makes the SAS user's life easier. We will take a closer look at some of the built-in features within SAS Enterprise Guide and how they can make your life easier. One of the most exciting and powerful features we explore is allowing parallel execution on the same server, giving the ability to run multiple SAS processes at the same time regardless of whether you have a SAS® Grid Computing environment or not.

Other topics covered include conditional processing within SAS Enterprise Guide, how to securely store database login and password information, setting up autoexec files in SAS Enterprise Guide, exploiting process flows, the new EG data step debugger, and much more.

INTRODUCTION

As Enterprise Guide has evolved, especially in the last 10 years or so, it has improved tremendously. EG has moved just providing an easy menu driven system to also providing an excellent platform for developing and running full production systems, with a minimum of coding. This paper emphasizes some of the lesser-known features of EG.

PROCESS FLOWS

A subdivision of an EG project is the Process Flow. All projects start with a single process flow named “Process Flow”. As objects are added to a project they appear both in the Project Tree window and in the EG work area where objects are shown with connecting lines to show dependencies and data flow.

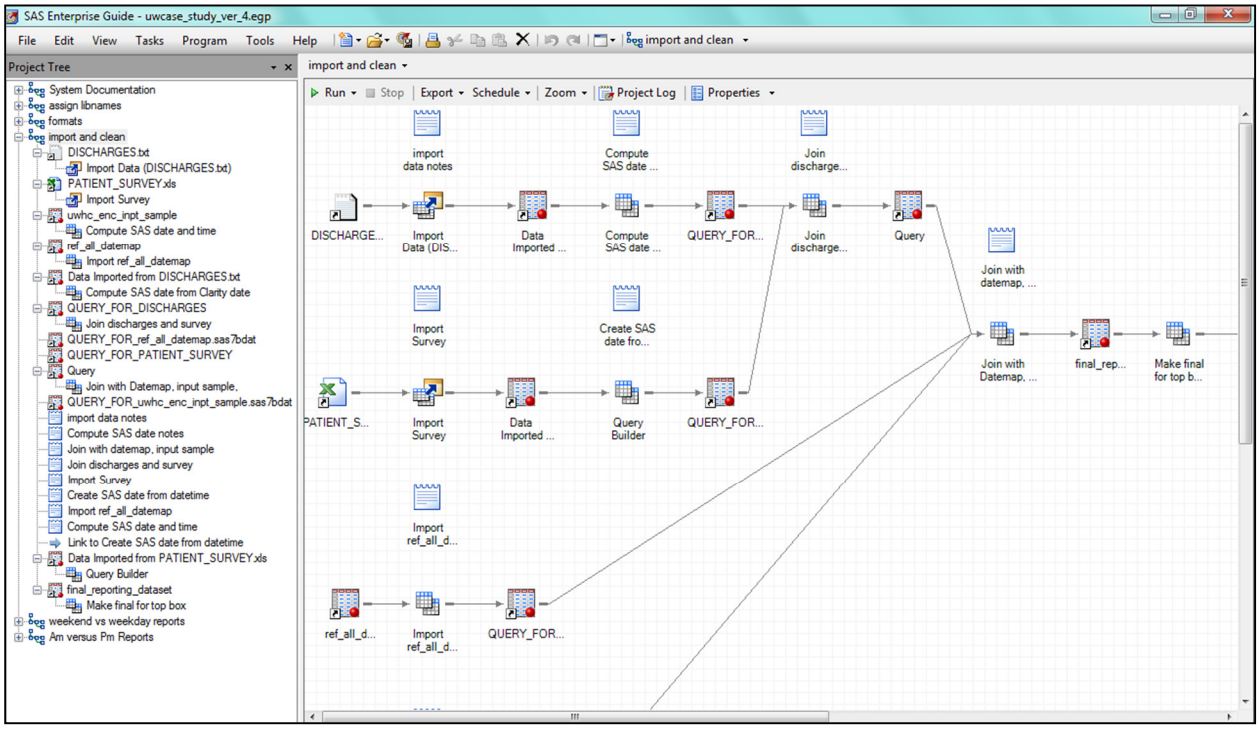
This is an excellent view of exactly how the “processes flow” when the project is executed. It can be an excellent source of documentation. As the number of objects increases, the process flow can get unwieldly by:

1. Showing just too many objects in a window. This causes problems viewing, understanding, and printing the flow. These issues can be remedied by adding multiple process flows and showing smaller numbers of objects in several different process flows.
2. Many of the objects use automatically generated names that are not especially useful. Each of the objects and the process flows themselves can be renamed by right clicking and substituting meaningful names.
3. Connecting lines between objects are automatically drawn, and sometimes lines become a tangle and cross each other. Right clicking anywhere in the white space of a process flow and unchecking “Auto Arrange” allows the user to drag objects so that connectors are neatly positioned.
4. Lack of documentation can occur. Text notes can be created that contain text documentation that is overall to the process flow or connected to an individual object within the flow to remedy this issue.

Note that process flows can be defined to show functional or logical breaks between them.

When running objects in EG, one can run a single node, a branch of a process flow, an entire process flow, or an entire project. In addition, if a process flow that is named “autoexec’ exists, it will automatically run when the project is opened.

Display 1 shows an example of a project that is broken into multiple functional breaks with one of the process flows expanded and the remainder condensed.



Display 1. A project with multiple process flows.

ALLOWING PARALLEL EXECUTION ON THE SAME SERVER

Allowing parallel execution on the same server is a useful option for speeding up SAS code. It allows the user to run independent code in parallel. Once the process is complete, data can be merged together downstream. It is especially useful to pull data from multiple sources. Once all the data is ready, EG merges it all together. This option works in most SAS environments, which could be as simple as a local install or as complex as a multi node Grid environment.

Behind the scenes, this option acts much like opening multiple Enterprise Guide sessions and submitting all a user’s code at the same time. At some point, the computer SAS is processing on may run out of CPU resources, and the advantage from running multiple processes in parallel will become less advantageous. In addition, if a user is running multiple processes against the same database it may slow processing on that database machine.

Setup Instructions:

1. Within the Enterprise Guide Project where the user wants to allow parallel execution, select File>Project Properties>Code Submission>Allow Parallel execution on the same server
2. Break apart the code into separate program nodes. Each of the nodes will get a separate SAS process
3. Right click on the program node and select “Link Program to” and choose the program to link to. (This will control the order and dependencies of the programs)

4. Navigate to %appdata%\SAS\SharedSettings\7.1\Engine and edit SystemSettings.xml file

   Update the WSLongLife parameter from 7200 to 5. This parameter represents the number of seconds the session will remain open after processing completes. This was recommended by SAS support as an option to prevent a user from locking resources up when the SAS process completed. For example, say one is running 10 processes in Parallel. If nine finish in a few minutes, those will be released and not have to wait for that 10th process to finish.

   <WSLongLife>5</WSLongLife>

5. To submit, one can run the entire process flow, and it will run in the order it was defined.

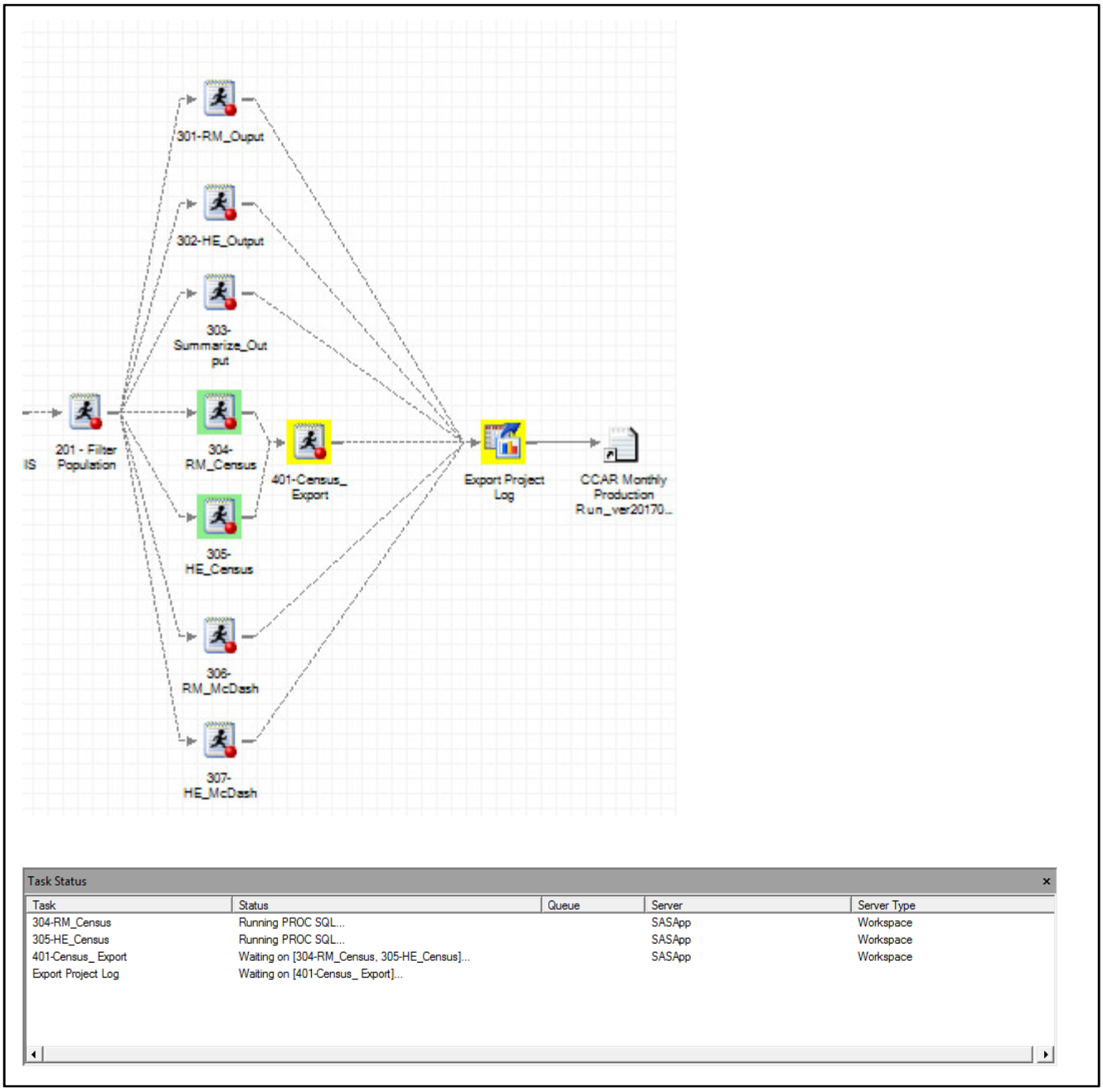
Passing information between sessions:

Because these sessions that are running in parallel are independent of one another, they can't directly share macro variables or work files. To get around this, one can store the output from the process on a file system that is accessible from all grid nodes.

For macro variables, one can use a similar method. This is a two part process where one can store the parameters in a permanent SAS file that is accessible by all grid nodes. Then the user can run this code at the beginning of each downstream SAS process that takes data step variables and uses call symput to put them into macro variables.

```
proc sql noprint;
select distinct "call symputx('"!!strip(name)!!"','"!!strip(name)!!");"
  into :symputx separated by ' '
from sashelp.vcolumn
WHERE libname="CCAR1"
and memname="CCAR_PARMS";
quit;
/*Use variable created above and execute call symput*/
data _null_;
set CCAR1.CCAR_PARMS;
&symputx;
run;
```

Display 2 shows multiple sessions in green running in parallel:



Display 2. Process Flow running in parallel.

CONTROLLING OUTPUT DATASETS IN THE PROCESS FLOW

Process Flows in Enterprise Guide offer a great way to document a process. It can be easy to get a lot of clutter in the process flow which makes it difficult to read and see what is going on. There are some techniques that allow one to control the output datasets that appear in a process flow. This can be quite useful in minimizing numerous output datasets from adding confusion.

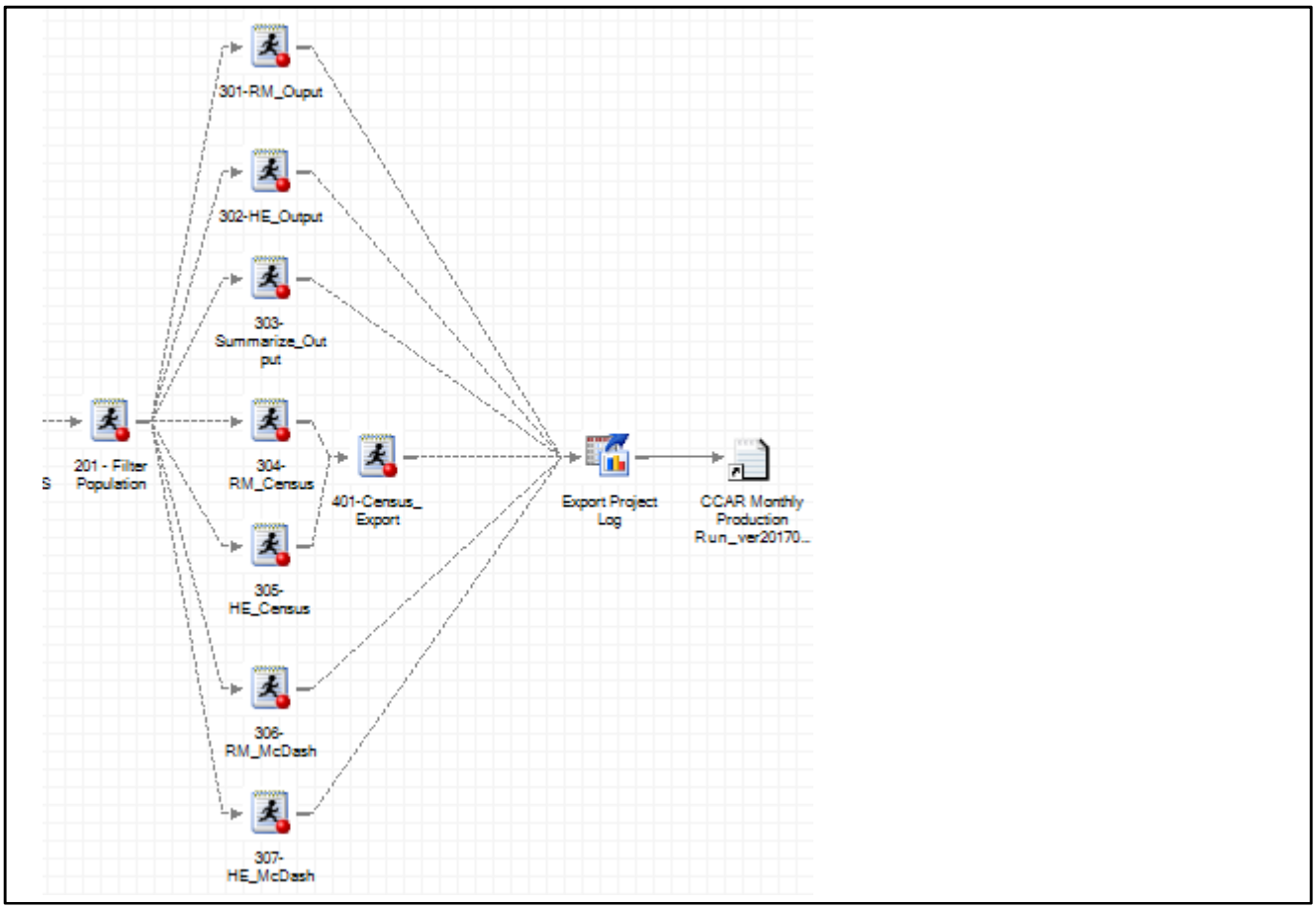
1. The first option is to limit the number of output datasets. If one follows these steps: File>Project Properties>Output datasets and then enter a value for "Maximum number of output data sets to add to the project". The nice thing about this technique is it can be set once per project. The default if 50, but it can be set as low as zero. If the program reaches the maximum one can enter, then it will still create the files, but any dataset created over the maximum won't be added to the process flow automatically. There will be a note that is auto created that will say the maximum has been hit.
2. Another technique that can be used is to use PROC DATASETS to delete any datasets that aren't used in another program node. For example, maybe a user runs a program that creates 10 files but they are only interested in displaying the final dataset. If they delete datasets 1-9 they will not show up in the process flow. A best practice is in most cases is to delete all the WORK datasets using the supplied macro below and store a permanent SAS file with the results the user is interested in. This example has a parameter that allows the user to turn this off by setting the discard_intermediate macro variable to anything other the Y.

```
%let discard_intermediate=Y;
%macro discard_intermediate;
%if &discard_intermediate=Y
%then %do;

/* Delete Temp datasets*/
proc datasets library=WORK kill nolist;
run;

%end;

%mend discard_intermediate;

%discard_intermediate
```

3. By clearing a libname, it will prevent any datasets written to that library from being added to the process flow. This option works well in conjunction with option two above and will prevent any datasets from being added to the process flow. Simply run a libname clear at the end of the program.

```
/* Prevents dataset written to library from being added to flow */
libname temp clear;
```

Using the above methods will result in a clear concise flow that is easy to follow as shown in display 3.



Display 3. Clean Process Flow.

ANALYZING CODE FOR GRID COMPUTING

This is another option that allows one to run code in parallel. In this case, having a SAS Grid isn't required as the name implies. One can still gain performance improvements by using this method. Enterprise Guide will run a given SAS program and analyze it to determine if pieces can be run in parallel. Then it gives the user an output program that has the code the user supplied wrapped in some SAS generated code that will run the code in parallel. One disclaimer is that it isn't recommended to make changes to this SAS generated code. Of course one can ignore this, and it could cause some issue when upgrading, so modify at one's own risk.

Setup Instructions:

1. Within the program to analyze for grid computing, select Analyze Program"> Analyze for Grid Computing. It will display a screen like display 4 below:
2. Click Begin Analysis (This will run your program)
3. Choose the option to Add to Project. Give the program a name at this point or use the default.

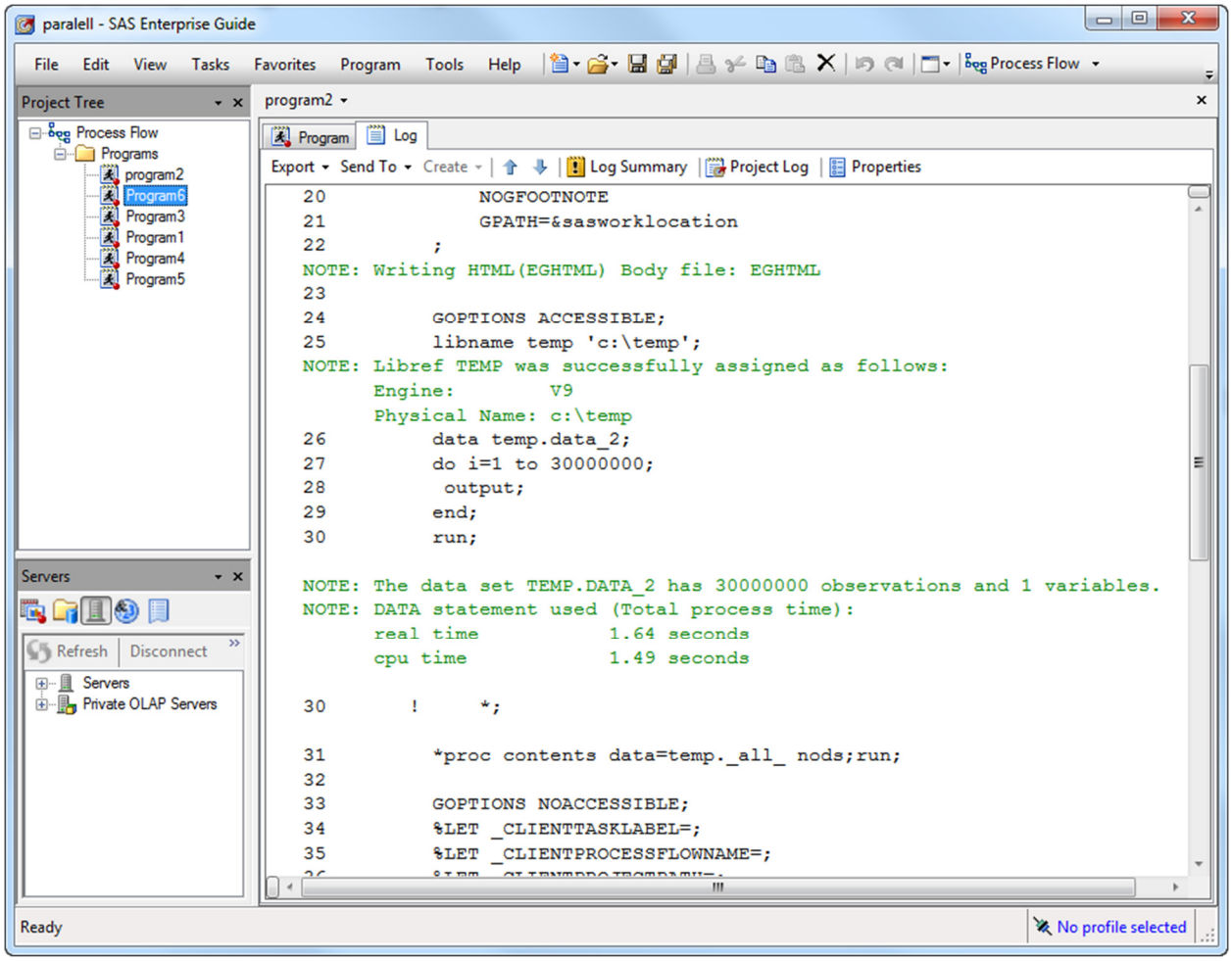
Once added to the project, it can run the newly generated program.

Display 4. Analyze code for grid computing.

THE PROJECT LOG

Each time that an EG task or program is run, a separate SAS log is generated for each task or program. Log icons can be shown alongside the tasks or programs in the process flow. These rapidly clutter up the process flows, and if options are set to not show log icons, the log can still be accessed by goint to the individual tasks and then clicking on the Log tab. The familiar SAS log shows results of automatic "wrapper code" positioned before and after the program or task code. Each time the task is rerun, the previous log is replaced with the latest log.

Display 5 shows a log generated by a single program.



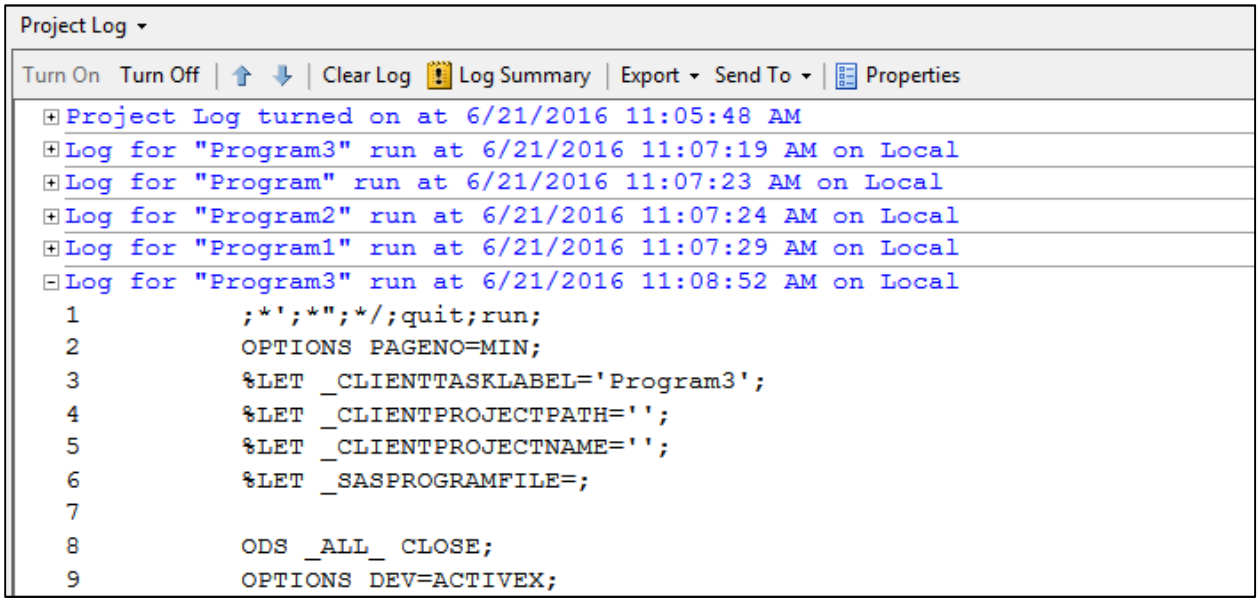
Display 5. Viewing a single program's log.

Another approach is to save the Project Log, which is a document with *all* program logs accessible in a single place. This provides an excellent one-stop storage for all SAS logs in a single location. The Project Log can be turned on and off and cleared using the project log menu choice in the process flow. This allows the developer to look at all tasks and programs run for as long as the project has been in existence. This can be a very handy item for the developer to look back a run or more ago for possible debugging. One EG user described it as a great audit trail when asked by a client how logic had been developed weeks early. The user was able to click on the project log and determine the code that was used earlier, along with row counts and other log contents. The project log can be cleared while testing and turned off and on as appropriate.

The Project Log can be exported to a text file for safe-keeping outside the project by using the manual Export Project Log menu item or by using the Export Project Log As A Step In Project to automatically export the project log each time the process flow is run.

One disadvantage to the Project Log is that if it exceeds 5 megabytes in size, the project stops executing. Only by going to the File Project Log can it be cleared to make it smaller. It would be very useful if EG developers would enhance EG to automatically file and clear the Project Log (or provide another solution) when the log fills so that the run could continue.

Display 6 shows the Project Log with one log expanded.



Display 6. The Project Log.

CONDITIONAL PROCESSING

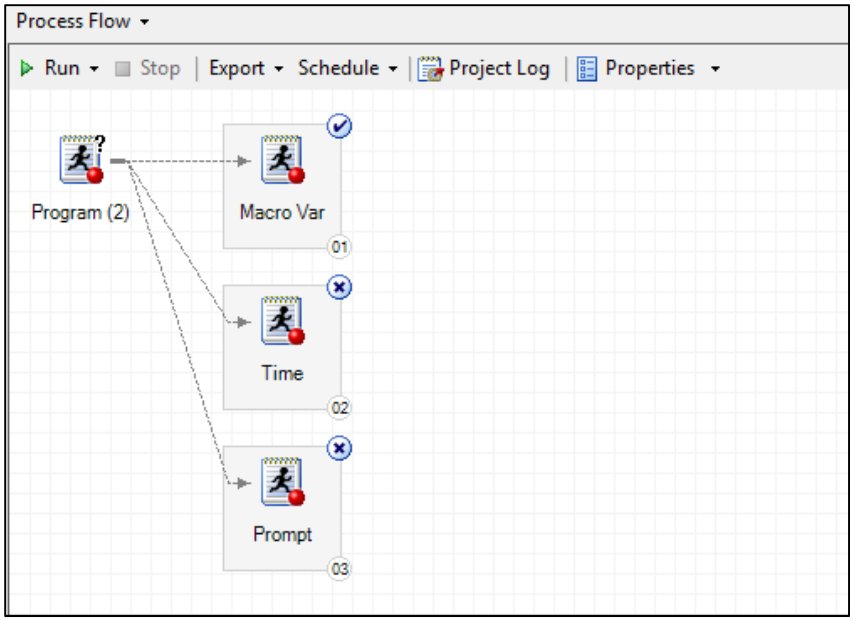
The ability to conditionally run any node, (program, query, task) can be accomplished by right clicking on the node and using the condition menu choice to add a condition. The types of values that are compared in conditions can be:

1. A previously set macro variable.
2. A prompt where the user is prompted to enter or select a value at run time. The response essentially ends up being stored in a macro variable.
3. Various date and time values such as day of the week.

Standard comparisons such as equal, not equal etc. can be used in the condition.

This essentially gives macro logic abilities to EG so that entire blocks of code can be run or omitted.

As the node runs, the condition is evaluated, and the node runs when the condition is true and is skipped if false. After the run is complete, the process flow objects will be marked with a checkmark if the node ran and an x if it did not as in Display 7.



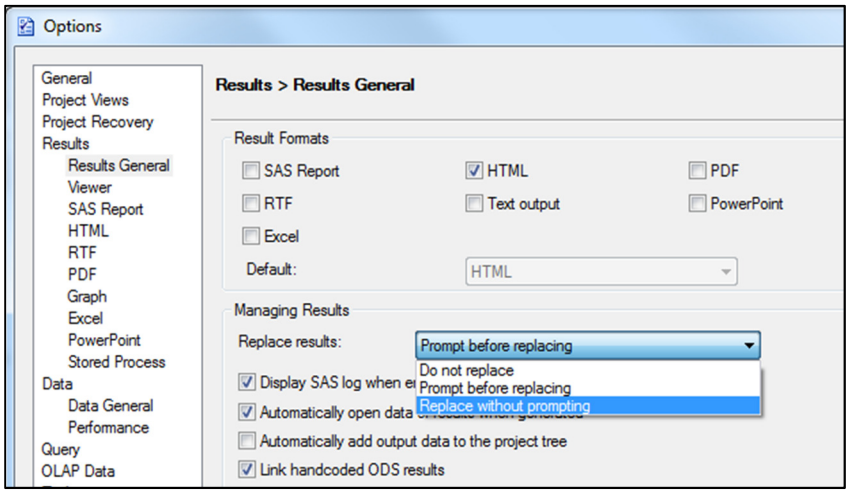
Display 7. Nodes After Conditional Processing.

A FEW USEFUL EG OPTIONS

There is an extensive list of options that can be set for an EG system. Like so many things in the SAS system, if there is something that is tedious or bothering users, there is probably a way to change it.

An example is shown below in Tool Options where one can specify to automatically replace results without asking. Also note that recent versions of EG now produce results in new destinations, such as Excel and PowerPoint. Taking the time to browse through the large number of options might make the EG experience more convenient.

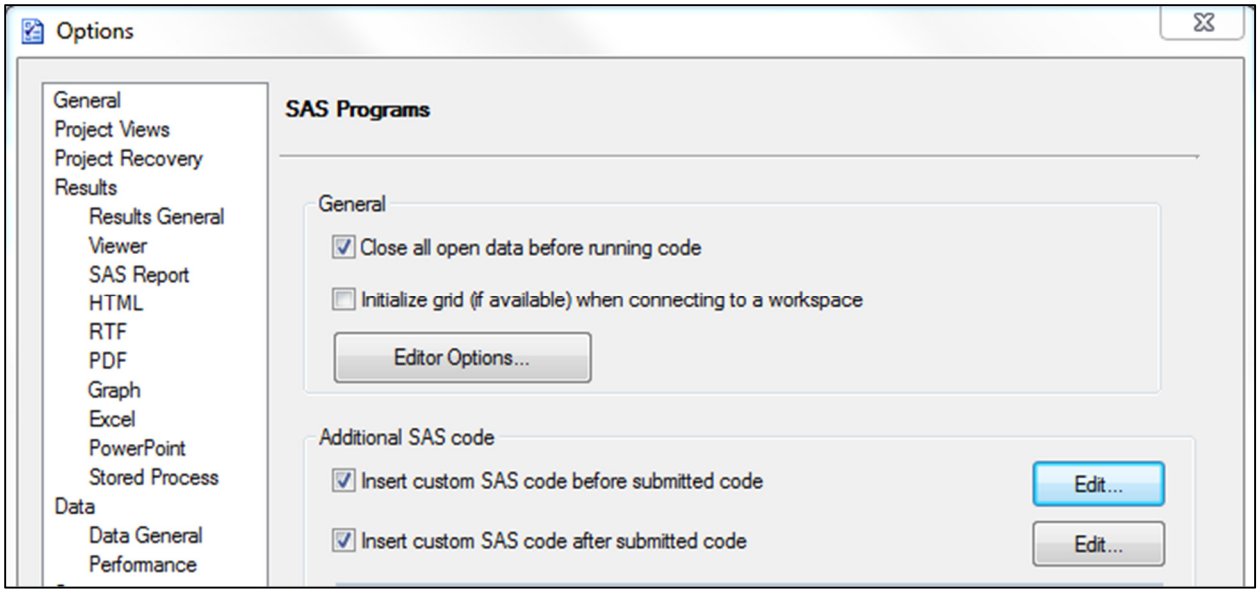
Display 8 shows menu choices:



Display 8. Tools Options Menu Choices.

INSERTING CODE BEFORE AND AFTER EACH SUBMITTED NODE

Tools, Options, SAS Programs has choices to insert custom code before each node as it runs:



Display 9. Tools Options SAS Programs.

It is very useful, especially in a Grid environment, to display in the SAS log, Start time, userid, host name, work library locations. Any code can be inserted as appropriate. Here is sample code:

```
data _null_;
JobStarted = datetime();
call symput('JobStarted', JobStarted);
User="&SYSUSERID";
Host="&SYSHOSTNAME";
Work=&SASWORKLOCATION;
putlog "************************************";
putlog "JobStarted -" JobStarted datetime18.;
putlog "User -------  " User;
putlog "Host -------  " Host;
putlog "Work -------  " Work;
putlog "************************************";
run;
```

After code is keyed or pasted into the edit window, before submitting, it can be saved. This setting is by user, and it will apply to all new projects as projects are added.

The code used after submits is a bit more involved, as it checks to see what the SAS source option before the custom code, turns off the source, then resets it to the original setting. Without doing this, the log is cluttered near the end of the job. Other values displayed are some of the original values again, along with end time, elapsed time, the last dataset name create, along with the number of rows in that last file.

```
proc sql noprint;
select setting
  into :optname_setting
```

```
  from sashelp.voption
  where optname='SOURCE'
  ;
quit;

options nosource;

data _null_;
JobStarted = &JobStarted;
JobEnded = datetime();
RealTime=JobEnded-JobStarted;
User="&SYSUSERID";
Host="&SYSHOSTNAME";
Work=&SASWORKLOCATION;
Last_DSN="&SYSLAST";
Last_OBS=&SYSNOBS;
format Last_OBS comma20.;

putlog "************************************";
putlog "JobStarted -" JobStarted datetime18.;
putlog "JobEnded ---" JobEnded datetime18.;
putlog "RealTime --- " RealTime time8.;
putlog "User -------  " User;
putlog "Host -------  " Host;
putlog "Work -------  " Work;
putlog "Last DSN ---  " Last_DSN;
putlog "Last OBS----  " Last_OBS;
putlog "************************************";
run;

options &optname_setting;
```

A partial log is shown below:

log lines near beginning

```
**********************************
JobStarted -  22FEB17:14:59:47
User -------  user
Host -------  servername
Work -------  /sas/grid/SAS_work2A44000044D5_servername
**********************************
```

More log lines

```
...
```

log lines near end

```
**********************************
JobStarted -  22FEB17:14:59:47
JobEnded ---  22FEB17:15:03:46
RealTime ---  0:03:58
User -------  user
Host -------  servername
Work -------  /sas/grid/SAS_work2A44000044D5_servername
Last DSN ---  TEMP.COMBINE_MIS
Last OBS----  3,488,028
**********************************
```

OPTIMIZING FILE IMPORT WIZARDS

The import wizard is a very handy tool for reading text files. It is a point and click method for generating data step code that can parse the file and create a SAS file. When running SAS on an external server, it is possible that you can connect to a file through Windows and Enterprise Guide actually copies the file to the server and cleanses the file before SAS reads it into a data step. This is done behind the scenes and it is important to understand because in general it is best to have the file stored accessible by SAS directly on the server. Depending on the network connection between the SAS Server and the Windows file system this can be a big time waste. The key to optimizing the import is to store the file directly on the SAS Server (or in some location that the server can access), and the import wizard will not run the internal routine to transfer the file to the SAS server.

Notes from SAS Log for non-optimized import:

```
1          ;*';*";*/;quit;run;
2          OPTIONS PAGENO=MIN;
3          /* ---------------------------------------------------------
4             Code generated by a SAS task
5
6             Generated on Monday, February 27, 2017 at 2:18:47 PM
7             By task:     Import Data Wizard
8
9             Source file: H:\test.txt
10            Server:      Local File System
11
12            Output data: WORK.TEST_0000
13            Server:      SASApp
14
15           Note: In preparation for running the following code, the Import
16           Data wizard has used internal routines to transfer the source data
17           file from the local file system to SASApp. There is no SAS code
18           available to represent this action.
19         --------------------------------------------------------- */
```

What one can see from lines 15 and 16 of the log is that it ran the internal routine to transfer the file to the server. Again, if the file is already stored on the server, this step is completely skipped and the data step will reference the file using the server path.

STORING PASSWORDS IN METADATA USING AUTHDOMAIN

Storing passwords in Metadata gives one a secure method to connect to external databases without storing passwords in open code. This method can be used with both libname connections as well as pass through code. This has been tested in SQL Server, Oracle and DB2 databases.

Setup Instructions:

1. User's SAS Administrator must setup an Authdomain on the SAS Management Console.

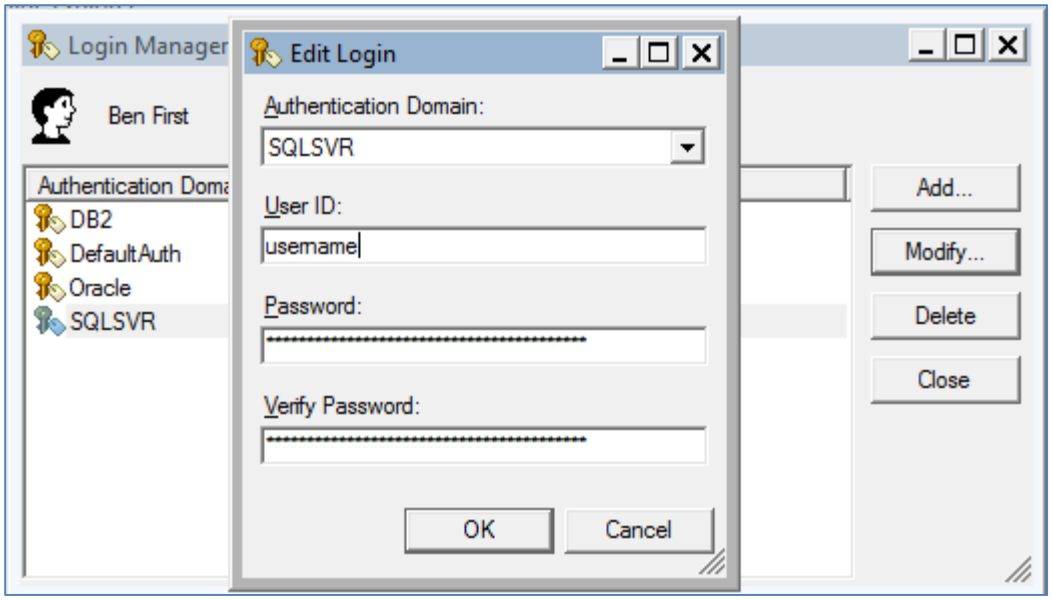
   See Link for Details:

   http://support.sas.com/documentation/cdl/en/mcsecug/61708/HTML/default/viewer.htm#n11xfvoau0jli0n1cl6rtfh5idfx.htm

2. Individual SAS users assign database username and password to Authdomain.

   The instructions below assume the site allows users to access the SAS Enterprise Guide Explorer from Enterprise Guide. Some sites don't allow this for security reasons. If a site does not allow access the SAS Enterprise Guide Explorer, one can achieve the same thing using the SAS Personal Login Manager.

   - Open Enterprise Guide
   - Go to tools>SAS Enterprise Guide Explorer
   - Click File>manage logins
   - Click Add
   - Select desired Authdomain for authentication domain
   - Enter database id
   - Enter database password
   - Click ok and open a program and run the new code that references the authdomain



Display 10. Login Manager.

3. SAS User includes authdomain in libname or pass through code

Libname example:

```
libname test sqlsvr dsn=test_db authdomain=sqlsvr schema='dbo';
```

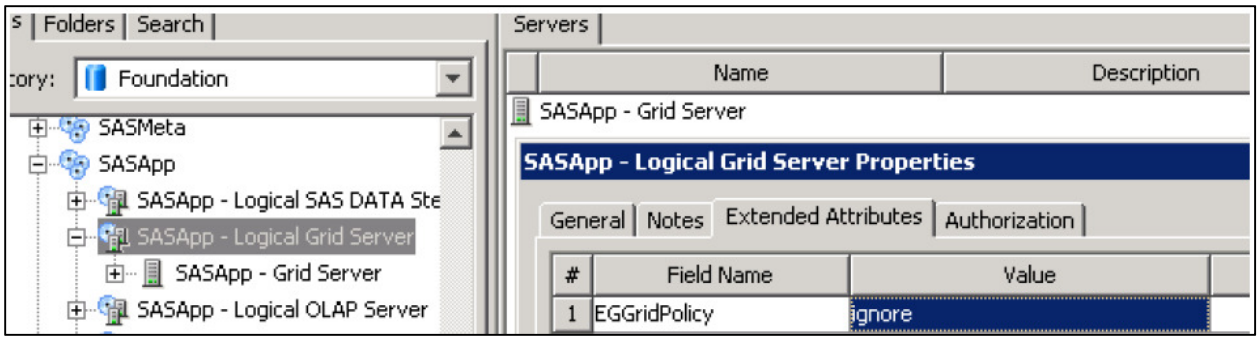
Pass through example:

```
proc sql;
connect to sqlsvr(dsn="test_db" authdomain=SQLSVR);
create table test as
select * from connection to sqlsvr
(
SELECT   top 10 *
FROM  dbo.test_201612
);
disconnect from sqlsvr;
quit;
```

GRID LAUNCHED WORKSPACE SERVERS

This is an option that was newly added in SAS 9.4. Prior to SAS 9.4, if one wanted to take full advantage of grid balancing, one needed to use special wrapper code to rsubmit the code onto the execution server. From an Enterprise Guide perspective, this complicated things because there were multiple work libraries (WORK, RMTWORK, GRIDWORK). In addition, Enterprise Guide wasn't as interactive. When code was submitted, the log wouldn't update as the program progressed. This made it difficult and confusing because one couldn't monitor the progress of a program.

Using Grid Launched Workspace Servers with a SAS 9.4 Grid environment works much better. Everything can run on one server, and all the grid balancing is handled behind the scenes. One setting that can be used in conjunction with this is E.G. Grid Policy set to IGNORE. This has been helpful because regardless how each individual user sets their grid options in Enterprise Guide, they will connect consistently to the Grid and run against the Grid Launched Workspace Server.



Display 11. SAS Management Console setting.

For more information on Grid Launched Workspace Servers see documentation and search for "Launched"

http://support.sas.com/documentation/cdl/en/gridref/67371/PDF/default/gridref.pdf

USER SPECIFIC AUTO EXECS

Autoexecs have always been a quite useful feature within SAS software. As coders have moved to Enterprise Guide and server based processing, the way Autoexecs are invoked has changed. For an administrator who supports hundreds of users, there is a simple method where any user could create a custom script to run automatically when connecting to SAS. This is an excellent solution for having libnames show up in SAS Add-In for MS Office without becoming a huge administration burden. A user could create a personal autoexec and could suddenly connect to any SAS Library they wanted.

Setup Instructions:

1. Insert the code below in an autoexec file that runs for the desired application. There may be a script created by the SAS install that runs on connection to SASApp. Both Enterprise Guide and SAS Add-In for MS Office connect via SASApp. 99% of SAS processes are run through those methods, so this was an ideal solution.

   Location of script:

   /sas/sys/sasconfig_prod/compute/Lev1/SASApp/appserver_autoexec_usermods.sas

Code to insert:

```
/*Code checks to see if file exists and if it does, runs the files*/
data _null_;
file_exists=fileexist("/sas/sasperm1_prod/AutoExecs/&SYSUSERID..sas");
code='%include "/sas/sasperm1_prod/AutoExecs/&SYSUSERID..sas";';
if file_exists=1 then call execute(code);
run;
```

2. Each SAS user creates a SAS file in the folder referenced in the script above. The name of the file will be the value of the users &sysuserid, and the file can contain any SAS code you want to run automatically.

   Example File for user user:

   /sas/sasperm1_prod/AutoExecs/user.sas

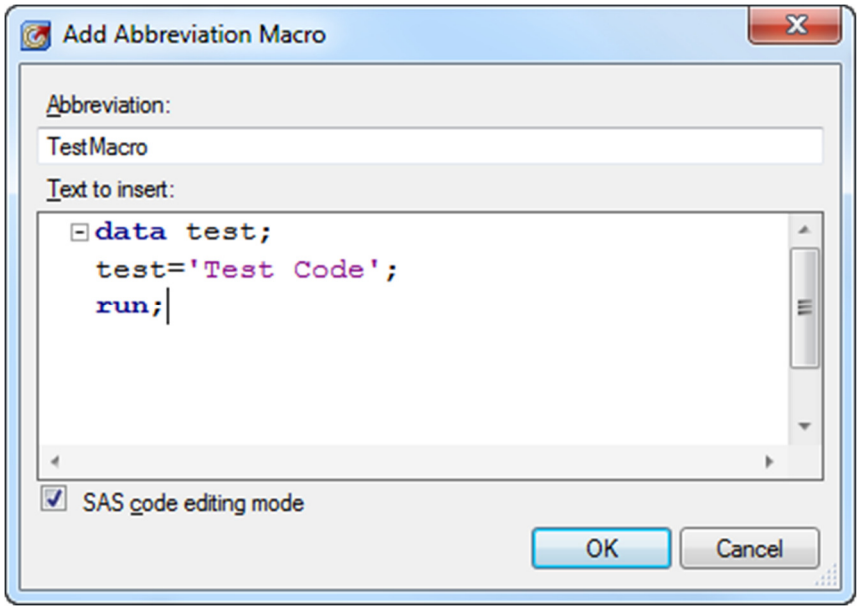
LEVERAGING KEYBOARD MACROS

Keyboard macros are a quick and easy solution to assign an abbreviation to a snippet of code or routine. This comes in very handy when one has the same bit of code they want to reuse when developing new code. It promotes consistency in code and also allows a user to not have to remember as much syntax.

There are two easy ways to create a basic keyboard macro. Both require one to be in a program editor in Enterprise Guide. From there, click Add Abbreviation Macro or use the keyboard shortcut (control + shift + A). This will bring up the Add Abbreviation Macro Screen. It consists of two sections Abbreviation and Text to Insert. The Abbreviation section requires a keyword that will be used to invoke the macro. The Text to insert section is the snippet of code that will appear when the macro is invoked.

Once a user creates their Abbreviation Macro it can now be used from an editor. Simply start by typing the key word, and then hit the tab key and the code snippet will come up.

More advanced macros can be developed under the Program > Editor Macros > Macros > Create. This brings up a Create Keyboard Macro menu. From there one can click the Create button and the user has access to a number of SAS created macros that can be used to build custom macros.

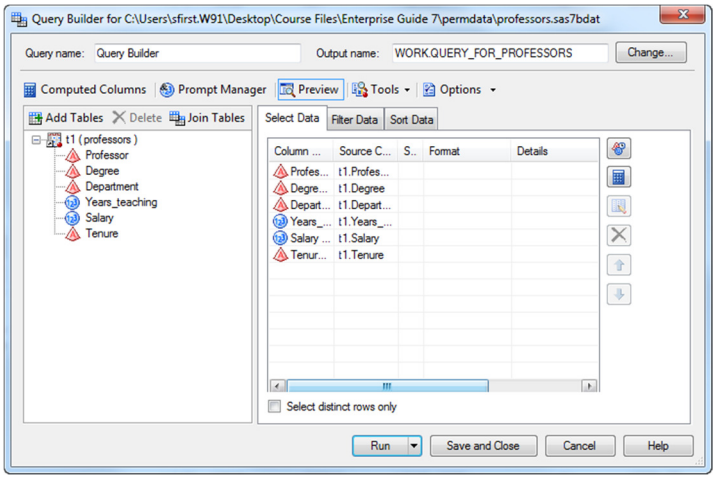


Display 12. A Keyboard Macro.

HIDDEN FEATURES OF THE QUERY BUILDER

The EG Query Builder is a graphical interface that generates PROC SQL code to be run against SAS files and relational databases. Almost all query clauses are available in the interface with the exception of some rarely used clauses such as UNION. For most queries, the normal SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY are more than adequate. The ease of use of selecting a column

name rather than spelling names correctly is a great time saver. Much of what used to require DATA step programming can be built using the query builder. Even if a user needs to eventually alter the generated SQL code, using the query builder to generate most of the code and then copying and pasting the generated code to a program node can still save a lot of tedious spelling, and keying. The display below shows the graphical interface followed by the generated SAS code:



Display 13. The Query Builder.

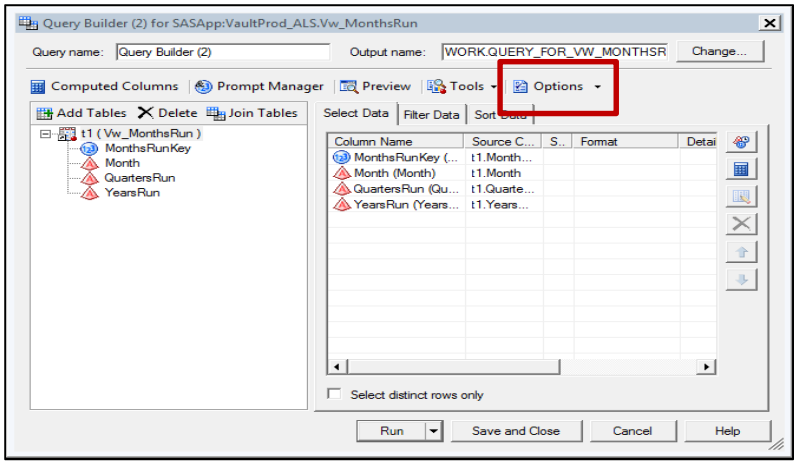
The generated code is below:

```
PROC SQL;
   CREATE TABLE WORK.QUERY_FOR_PROFESSORS AS
   SELECT t1.Professor,
          t1.Degree,
          t1.Department,
          t1.Years_teaching,
          t1.Salary,
          t1.Tenure
      FROM TMP00001.professors t1;
QUIT;
```

When a query uses data from an external database, one can use explicit pass-through mode to send the SQL statements to the database to be processed. If the data files are very large, this can improve performance because the files do not have to be copied to the SAS server for processing. Once the statements are processed, the results are sent back to SAS Enterprise Guide.

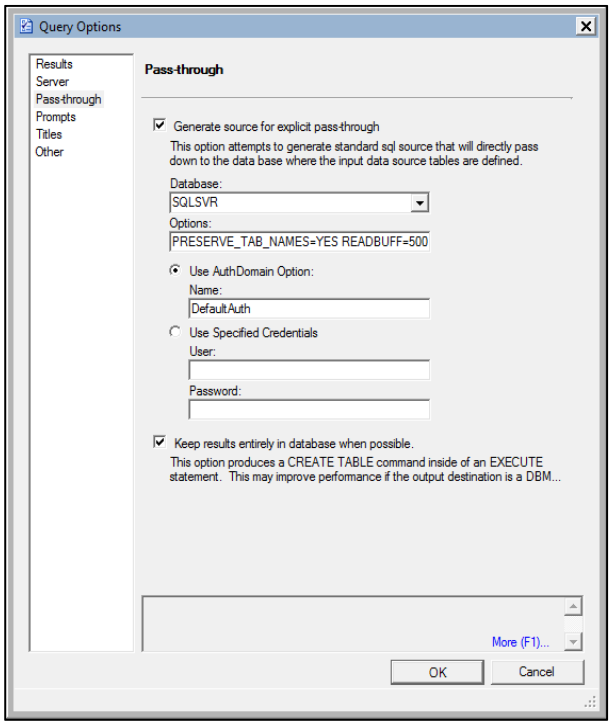
GENERATING PASS-THROUGH CODE

When an EG query uses data from an external database, there is an option to generate explicit pass-through SQL statements that are passed directly to the database to be processed. For very large files, performance improvement can be significant. It is not at first obvious where this is specified. From the query builder, the options button can be clicked as shown below:



Display 14. The Query Builder Options Button.

The Pass-through menu choice allows one to specify options for the database tables and options as shown in the following display:



Display 15. The Query Builder Pass-through Options Page.

The generated code is below:

```
PROC SQL;
   CONNECT TO SQLSVR as con1
    (PRESERVE_TAB_NAMES=YES READBUFF=500 DBSLICEPARM=(THREADED_APPS,2)
     Datasrc=vaultprod_als authdomain=DefaultAuth);
   CREATE TABLE WORK.QUERY_FOR_VW_MONTHSRUN AS
   SELECT *
```

```
      FROM CONNECTION TO con1 (
      SELECT "t1"."MonthsRunKey",
          "t1"."Month",
          "t1"."QuartersRun",
          "t1"."YearsRun"
         FROM "Vw_MonthsRun" "t1");
   DISCONNECT FROM con1;
QUIT;
```

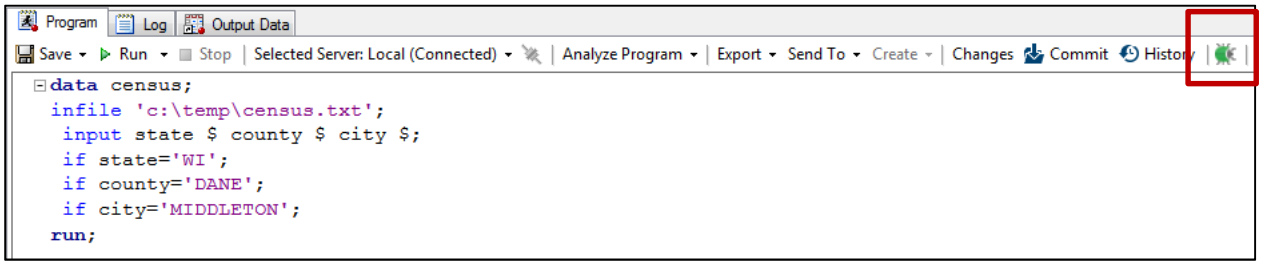
THE ENTERPRISE GUIDE DATA STEP DEBUGGER

In release 7.13 of EG, released in November of 2016, a new data step debugger was included. This debugger is a big improvement from the earlier DATA step debugger. Chris Hemedinger's blog, as referenced below, goes into a fair amount of detail demonstrating the debugger. This debugger gives a major productivity boost to debugging SAS data steps. A brief example shows some of the debugger's capabilities.

Suppose a user has a small census file with State, County, and City columns. They would like to select the Wisconsin, Dane County, City of Middleton row.

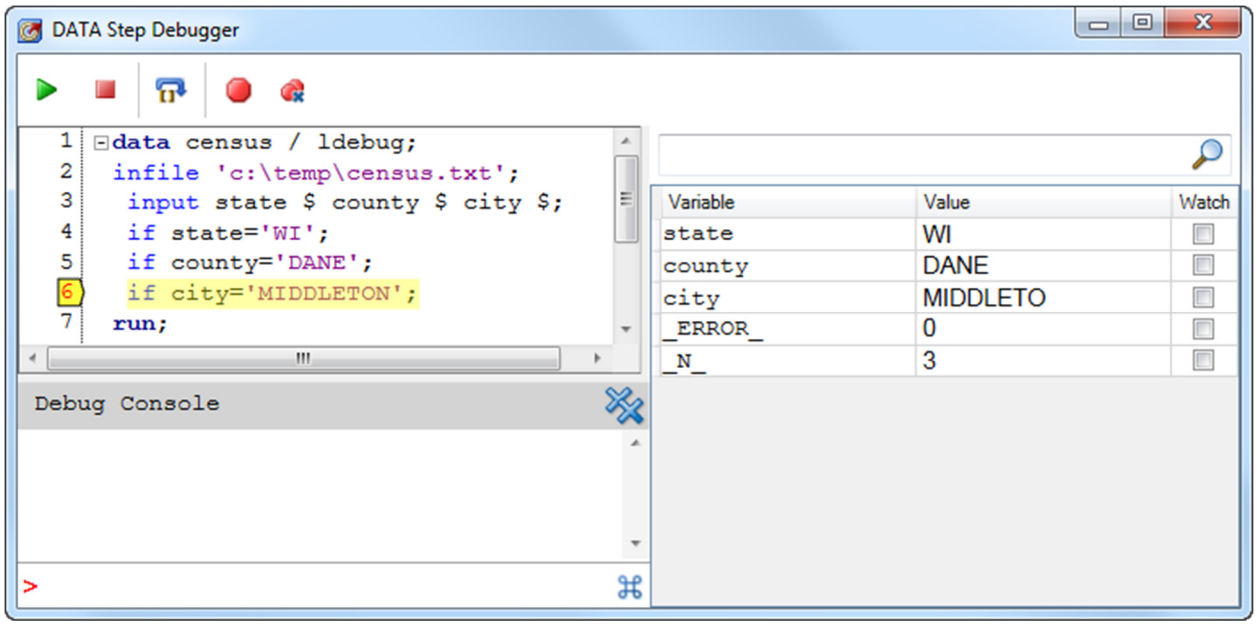
```
WI DANE      MONONA
WI WAUKESHA  WAUKESHA
WI DANE      MIDDLETON
MN HENNEPIN  PLYMOUTH
```

The following program should work for, but when run, no records are selected. Clearly there is an error, but where is it?



Display 16. An Incorrect Program

There is a new menu choice on the right looks like a green bug. Clicking on that bug, and then also clicking on the bug on the left side (shown below), brings up a debugging console, a display of most data step variables, which allows one to step through your program and much more. After stepping through until the third record is read, the debugger shows that while the code is checking for "MIDDLETON" in the program, the city name was truncated at input to "MIDDLETO". By adding a length statement before the input to set City to a length of 9 solves the problem.



Display 17. The EG Data Step Debugger

CONCLUSION

More than ever, Enterprise Guide is an increasing versatile product for non-IT and IT folks alike. Often some of the small and obscure features can help to build better systems with shorter efforts.

ACKNOWLEDGMENTS

Website http://blogs.sas.com Chis Hemedinger, “Using the DATA step debugger in SAS Enterprise Guide”, November 2016.
http://blogs.sas.com/content/sasdummy/2016/11/30/data-step-debugger-sas-eg/.
http://support.sas.com/resources/papers/proceedings14/SAS375-2014.pdf

CONTACT INFORMATION

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

Benjamin First
US Bank
608 609-1513
Benjamin.First@usbank.com

Steven First
Systems Seminar Consultants
608 239-4326
sfirst@sys-seminar.com
www.sys-seminar.com