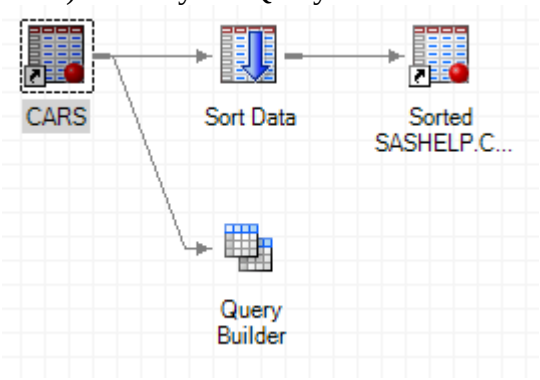
Fall 2012 OASUS Questions and Answers

The following answers are provided to the benefit of the OASUS Users Group and are not meant to replace SAS Technical Support. Also, an Enterprise Guide project is provided as a companion to this document. The project is available on the OASUS web site (www.oasus.ca) under the fall 2012 meeting since this is when these answers have been presented formally.

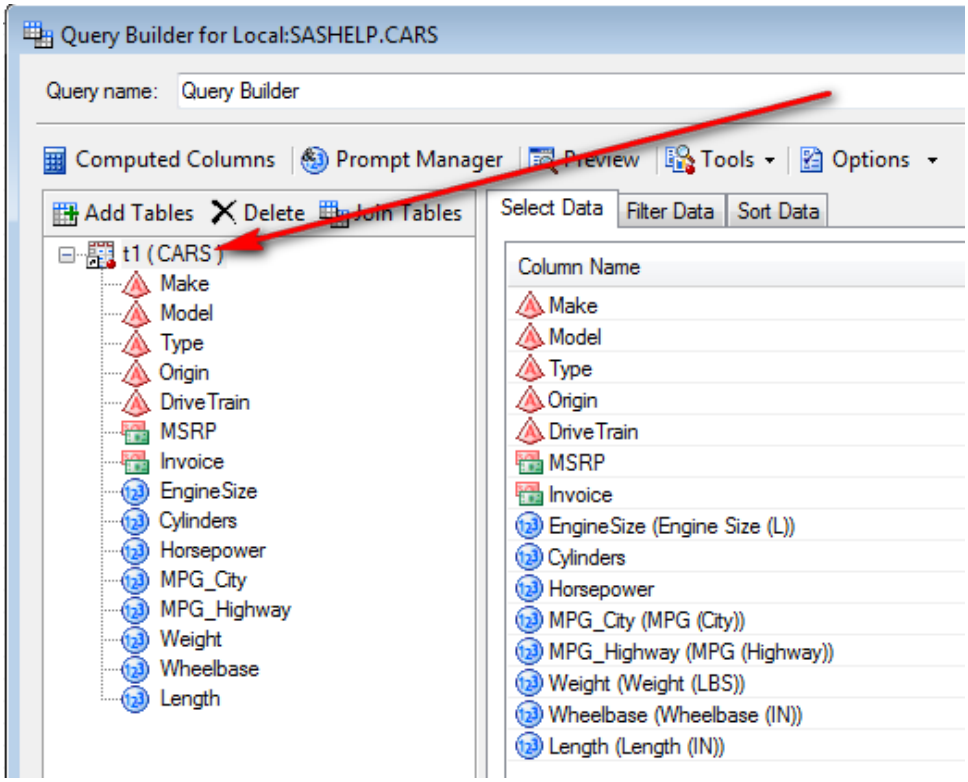
Question 1 : In EG, is there an easy way to switch the input a query uses (repoint to a different table without deleting everything associated with the first table and recreating it for the new table but with same structure)?

If the table you wish to switch to has the same structure as the one currently reference, it is quite straightforward to do so by following these few steps:

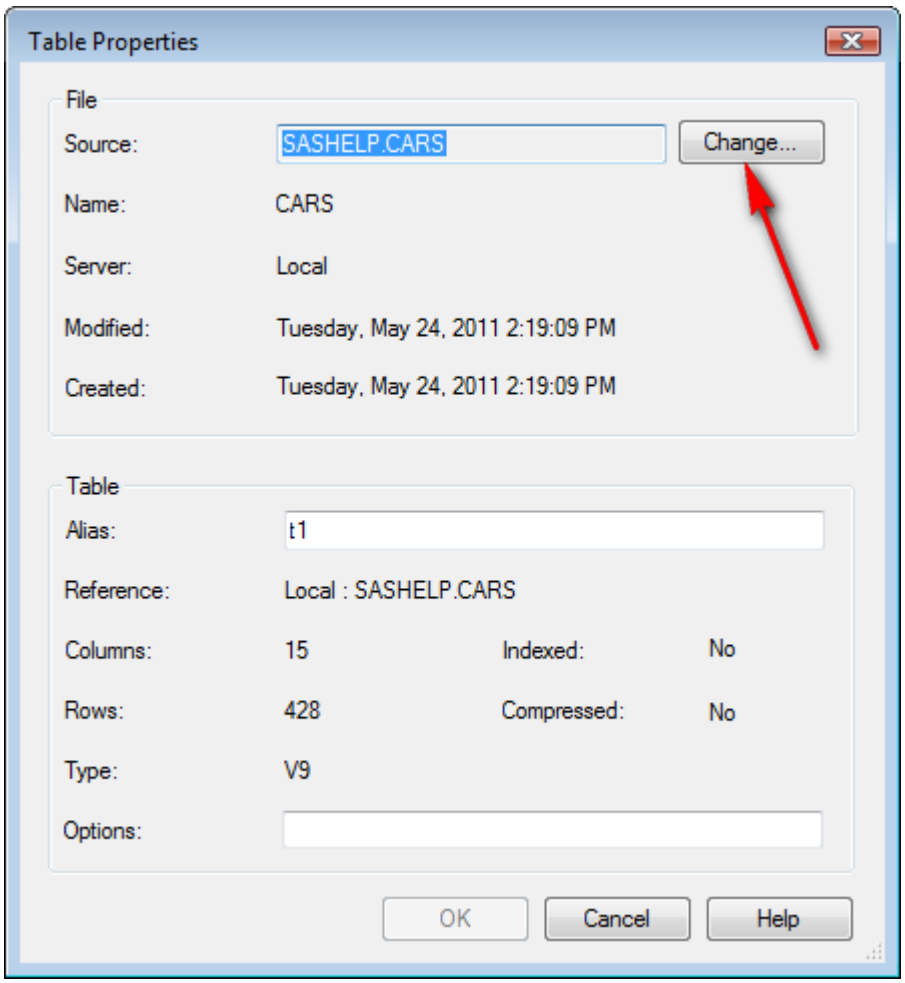
1) Modify the Query Builder task.



2) Right click the table you wish to replace; Select **Properties;**



3) On the File Source panel, select the **Change…** button.



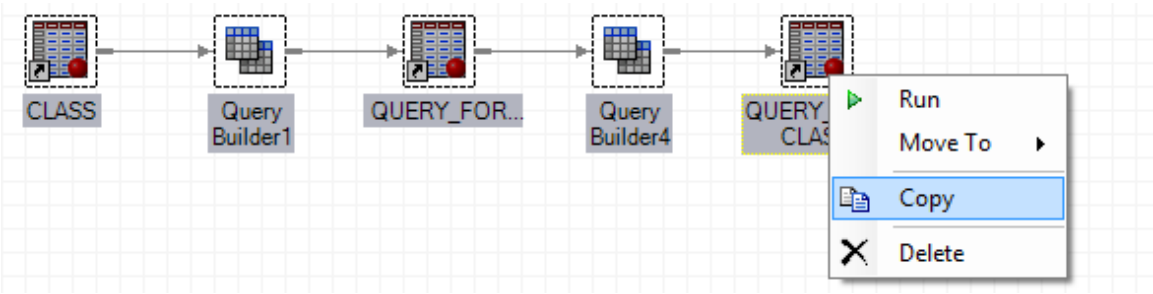
Note that the new source file does not have to be exactly the same as the original file but any variables that are used in the query must be present in the new source file.

Question 2 : In EG, is there a way to copy a sequence of queries (like if you want to make a change in one sequence but still keep the original)?

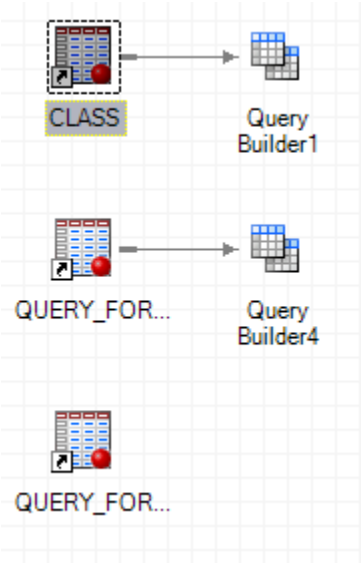
Yes it is possible to do so. Depending of what you intending to do, you can use at least three (3) approaches:

1. Simple copy paste.

    a. Select the tasks and copy/paste



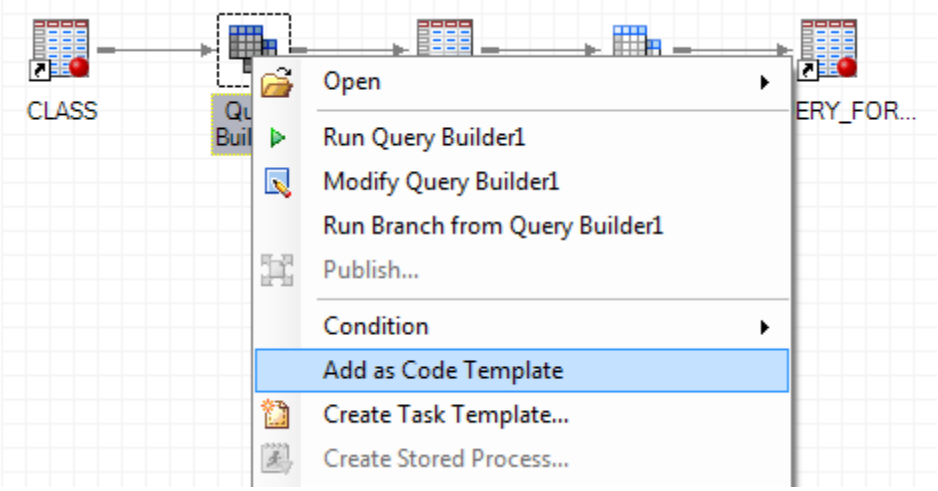
    b. Edit the new tasks



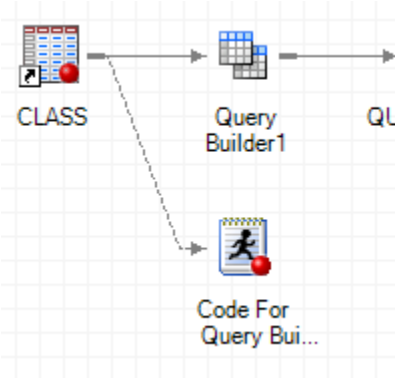
2. "Add as a code template"

    This is like a copy/paste but instead of a task being copied, the programming code is copied and made into a program task.

    a. First create the code template



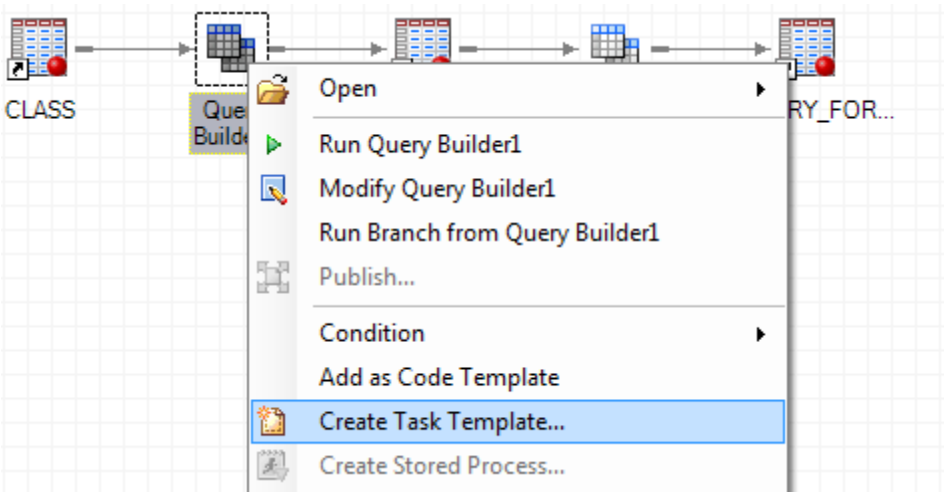
b. Modify the code to your liking

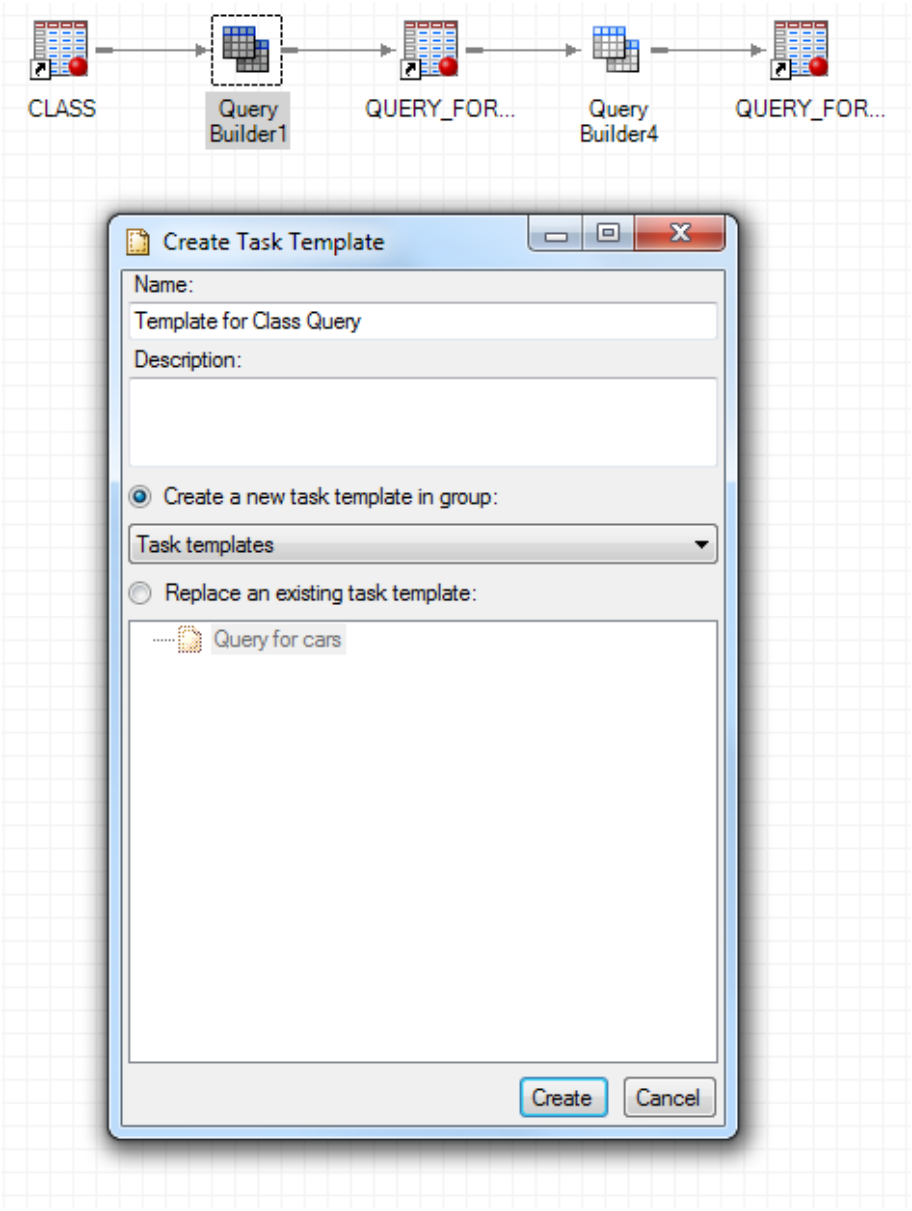


3. Task Template

A task template represents a task that has already been set-up and that can be re-used throughout any of your project.

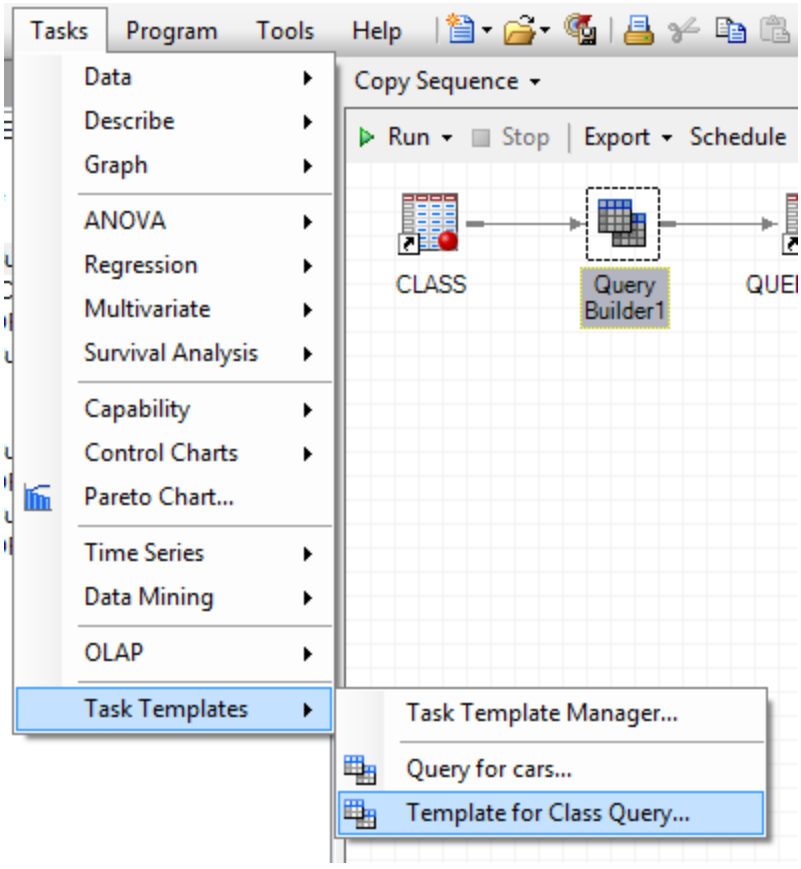
a. Create a Template


CLASS
Open
Run Query Builder1
Modify Query Builder1
Run Branch from Query Builder1
Publish...
Condition
Add as Code Template
Create Task Template...
Create Stored Process...


CLASS
Query Builder1
QUERY_FOR...
Query Builder4
QUERY_FOR...
Create Task Template
Name:
Template for Class Query
Description:
Create a new task template in group:
Task templates
Replace an existing task template:
Query for cars
Create
Cancel

b. Use the newly created task template

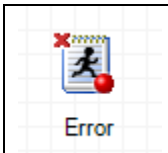
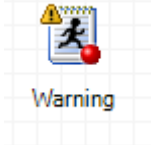
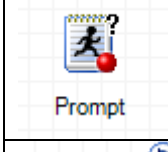
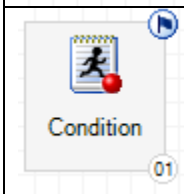
Using the task template is straightforward and will replay the task as it was when the template was created. Ideally, the input data file should contain the same variables at least for those who are selected.



Question 3 : In EG, after I run a program, there's a question mark in the right-hand side of the icon. What is that?

It means that the program is associated with at least one prompt.

Many indicators can be added to an EG task icon. The most common are:

	An X in the upper left corner indicates that at least one error was encountered in the last run of the task.
	A ! in the upper left corner indicates that at least one warning (but no errors) was encountered in the last run of the task.
	A ? in the upper right corner indicates that the task is associated with at least one prompt.
	A flag in the upper right corner indicates that a condition is associated with the task.

Question 4 : Is there an easy way to export variable labels into Excel, not variable names?

This can be easily done by using the label option in the PROC EXPORT.

```
/* Use LABEL option in PROC EXPORT */
proc export data=sashelp.cars
outfile='c:\temp\cars1.xls'
      label dbms=excel replace;
run;
```

You can achieved a similar result (but more fancy formatting) by using an ODS ExcelXPtagsets destination. Rather than creating a spreadsheet in native format, the ExcelXPtagsets will create an XML file using a set of tags that can be interpreted by Excel.

Once you have the ExcelXPtagsets destination open, you can run PROC PRINT with the LABEL option to print labels rather than variables names. The results of PROC PRINT will be saved in an XML file readable by Excel.

```
/* This code creates an Excel spreadsheet with the
labels as column headings */
ods tagsets.ExcelXP file="C:\temp\Cars2.xls";
   proc print data=sashelp.cars noobs label;
   run;
ods tagsets.ExcelXP close;
```

Finally, if you have the desire to do so, you can rename the variable with their corresponding labels prior to sending the dataset to Excel.

```
options validvarname=any;
/* Make a working copy of CARS dataset*/
data newcars;
      set sashelp.cars;
run;
/* Set-up a call to PROC DATASETS to rename the variable to
their corresponding labels */
data _null_;
      set sashelp.vcolumn(keep=libname memname name label
where=(libname='WORK' and memname='NEWCARS')) end=last;

      if _n_ eq 1 then
            call execute('proc datasets library=work
nolist;modify newcars;rename ');
      if label ne '' then
            call execute(cats(name,"='",label,"'n" ));

      if last then
            call execute(';quit;');
run;
/* Just export with no special option */
proc export data=newcars outfile='c:\temp\cars3.xls'
      dbms=excel replace;
run;
```

Reference:

DelGobbo, Vincent. 2011. "Creating Stylish Multi-Sheet Microsoft Excel Workbooks the Easy Way with SAS®". *Proceedings of the SAS Global Forum 2011 Conference*. Cary, NC: SAS Institute Inc.

Question 5 : What is PROC DOCUMENT? Can you give an example?

PROC DOCUMENT is part of the ODS DOCUMENT facility which allows you to:

- Save an ODS output into a ODS document which is a repository for ODS output, graphs and datasets;
- Replay the output to a specific format;
- Search and filter an ODS document;
- Add notes and change titles, footnotes, etc.

Before you can use PROC DOCUMENT, you have to create an ODS document by directing your output to an ODS document destination instead of a destination such as pdf or html that you would normally specify. The ODS document is saved in a special SAS item store file but you can use the normal <libref.>member-name nomenclature to designate the file. The following code provides an example:

```
ods document name=work.quickstart;
proc contents data=sashelp.cars ;
run;
proc means nmiss mean max min data=sashelp.cars;
class origin type;
run;
ods document close;
```

Now that you have an ODS document, you use PROC DOCUMENT to replay the outpit to another ODS destination:

```
/* replay the ODS document into an html output */
/*ods html file="c:\temp\output1.html";*/
proc document name=work.quickstart;
     replay;
     run;
quit;
/*ods html close;*/
```

You can keep adding to an existing ODS document by referring it as an ODS destination in other blocks of code:

```
/* Adding new content to a new ODS document */
ods document name=work.quickstart;
proc freq data=sashelp.cars;
     table origin  * type / list;
     run;
ods document close;
```

You can list the content of an existing ODS document:

```
/* Summary listing of the content of an ODS document*/
proc document name=work.quickstart;
     list;
     run;
quit;
/* Detailed listing of the content of an ODS document*/
proc document name=work.quickstart;
     list / levels=all details  ;
     run;
quit;
```

You can replay only certain ODS objects stored in the ODS document. In order to do that you need to understand how ODS name those object and how they are organized. Typically, each run of a procedure has a corresponding folder in the ODS document which is named according to the name of the procedure and is assigned a sequence number (example Freq#1). Sub-folders can also exist depending on the output produced by the procedure and eventually individual ODS objects are stored in a specific folders. You can replay specific object or all the objects contained within a folder.

```
/* Replaying only the objects generated by Freq#1 */
ods html file="c:\temp\output2.html";
proc document name=work.quickstart;
     replay freq#1;
     run;
quit;
ods html close;
```

There are many other operations you can do with PROC DOCUMENT. The book listed in reference is highly recommended to learn more.

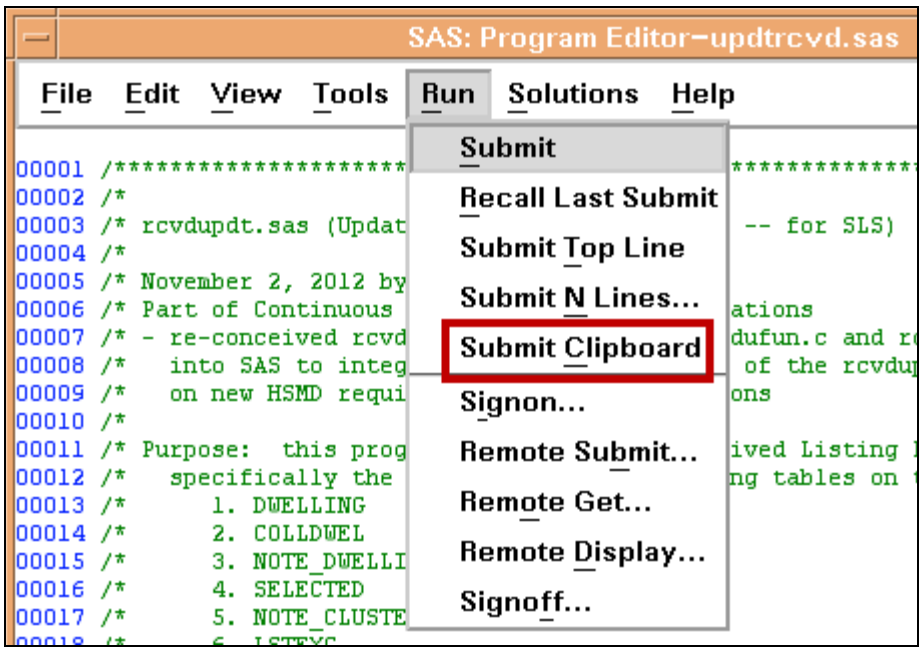
Reference:

Tuchman, Michael. 2012. **PROC DOCUMENT by Example Using SAS**®. Cary, NC: **SAS** Institute Inc.

Question 6 : How can we highlight a part of SAS code in program editor in UNIX and run the selected code?

In UNIX, there is a way to submit a selection via the clipboard. If one selects and highlights some SAS code in the Program Editor and then chooses "Submit Clipboard", that will only run the code that was selected.

Below is a snapshot of the drop down menu available from Run, with the "Submit Clipboard" boxed in red:



So, in other words, highlighting some code will select and copy it to the buffer (Clipboard), so that when "Submit Clipboard" is chosen as the option to Run, only the selected code stored on the buffer (Clipboard) will execute.

Question 7 : How much effort is required to migrate from EG 4.3 to EG 5.1 and from SAS 9.2 to SAS 9.3M1?

In general the migration is quite straightforward:

1) *Migration from EG 4.3 to EG 5.1*: EG has a built-in migration tool and is invoked automatically when opening a project created from an earlier release. You can also use the migration tool to migrate multiple projects before opening them in EG. This migration tool was fully covered in a the fall 2011 Q&A. Once the migration is complete, it is recommended to review the resulting project to ensure everything is laid out as before and produces the same results.

2) *Migration from SAS 9.2 to SAS 9.3M1*: assuming that this question pertains to Base SAS, the migration is also straightforward. Because Base SAS code is upward compatible, there should be no change in the SAS applications per se. SAS files created by an earlier version of SAS should be readable. , you have to pay attention to the SAS files used by your SAS applications. In particular, it is recommended to re-compile catalogues and re-create files for which you want to use new features. This can be done easily using the PROC MIGRATE facility.

In summary, migrating upward is quite easy and straightforward but it is mandatory to test the migrated application/project to ensure that the process was successful and that the application behaves as expected.

Question 8 : What is the most efficient (fastest) method for "transferring" a very large dataset to a SQL Server database table, where the dataset and the table are on different servers?

There are a few way to load data from a SAS dataset into SQL. You can create a text file and then load the data into SQL Server and then load the text file into the database or you can use the ODBC engine to load the SAS dataset directly.

Below is an example to load the dataset directly into the database. The example uses some options to improve performance such as the BULKLOAD and the DIRECT_EXE options, and the TRUNCATE SQL*server statement. The SQL pass-through is used as well. If you are not familiar with the SQL Pass-through, it simply executes SQL code directly on the database server, you can use it to do things you can't do using a library reference to the database like execute a stored procedure or issue T-SQL commands.

```
/* Loading a SAS dataset into SQL Server

  The BULKLOAD=YES option increases performance
  significantly (in this case 13 seconds to 2 seconds)
  Note 1: Special permissions needed to do a BULK Load
  Note 2: DIRECT_EXE=DELETE will have huge performance gains
          if deleting some records from an SQL table */

libname advworks odbc bulkload=YES  DIRECT_EXE=DELETE
noprompt="driver=sql server;server=server_name;
           database=adventureworks;
           trusted_connection=true"  schema=Sales;
libname testdata "<directory>";


/* The SQL pass-through is used to drop the SQL table */
proc sql;
   connect to sqlservr as testdb (server="server_name"
      database="adventureworks");
   Execute (DROP TABLE Sales.test) by testdb;
   disconnect from testdb;
quit;

/* Creating a new table, BULKLOAD option will be used */
data  advworks.test;
   set test;
run;

/* The SQL pass-through can be used to truncate the SQL table (Huge
performance again)*/
proc sql;
   connect to sqlservr as testdb
      (server="c184293\v2008r2"
      database="adventureworks");
   Execute (TRUNCATE TABLE Sales.test) by testdb;
   disconnect from testdb;
quit;
```

Question 9 : Is there any new ODS Excel tagset that will produce an xlsx file. At the moment, we use the Excel tagset that creates only xls.

The XML file created by the ODS ExcelXp tagset destination, is not specific to ExcelXp. Since it is not an Excel binary file, it can be read directly by the latest version of Excel – it is just a matter of using the .xlsx extension when creating the file.

Question 10 : Is there PROC EXPORT we can use to create xlsx files?

Depending on the version of SAS you can use one of these 2 methods:

1) Using DBMS=EXCELCS option

The EXCELCS method works by delegating the Excel read/write operation to a Windows node that has a PC Files Server instance. Therefore, in the context of UNIX it requires a second computer because the PC File server must run on a Windows machine. This method works well with SAS 9.2 and beyond. The following code is an example (the name of the server and the port number is specific to your installation):

```
proc export dbms=excelcs data=mysasdata
            outfile="/directory/filename.xlsx"
            replace;
   sheet='sheet-name';
   server="server-name.company.com";
   port=9621;
run;
```

2) Using the DBMS=XLSX option

The XLSX option creates an xlsx file without using a PC File server. It is available with SAS 9.3 but supports only 26 columns under SAS 9.3 M1 (this is actually a defect) but works without this limitation under SAS 9.3 M2. Here is an example:

```
PROC EXPORT DATA= WORK.LARGE
            OUTFILE="/directory/filename.xlsx"
            DBMS=XLSX REPLACE;
     SHEET="Sheet1";
     NEWFILE=YES;
RUN;
```