

# Using SAS® Enterprise Guide® to Create Standalone Programs

Mike Todd, Nth Analytics, Flemington, NJ

## ABSTRACT

Enterprise Guide (EG) is a visually oriented SAS programming environment that enables programmers to create diagrams linking objects that EG turns into SAS code. The object-oriented tasks in EG write SQL-based code. By incorporating an automated post-processor, EG offers a way to standardize programming style. This is particularly useful for SAS code submitted to regulatory agencies, or SAS code included as project deliverables from CROs. Limitations and cautions for Enterprise Guide are also discussed in this paper.

## INTRODUCTION

Why use SAS to write SAS programs? In the pharmaceutical industry, SAS programs are increasingly becoming deliverables. The FDA often requires sponsors to submit SAS programs, particularly the code for complex efficacy calculations. For contract research organizations (CROs), their clients often require all of the SAS code as contractual deliverables, either to preserve complete traceability for their own internal audits or to submit to regulatory agencies themselves.

Vachal (2001) recommends treating programs delivered to the FDA as a software engineering function. Vachal notes that FDA statistical reviewers spend a large portion of their time verifying and reproducing key efficacy and statistical safety analyses. He states that considerable effort has to be spent systematically testing for error-free performance and stability of the programs. I think the validation effort is actually fairly straightforward exercise in automatically checking the SAS code for warnings, errors, etc. and doing file comparisons on the outputs. Both of these validation steps can be easily automated. In this software engineering exercise, most of the effort comes in producing the programs, particularly in achieving consistency of style and working the limitations of whatever methodology you choose.

How do you provide easily understandable programs without giving away intellectual property? For any company, the brains of the reporting and analysis systems are the macros, i.e., the ability to generate SAS code (through the macro language) in a systematic and reliable fashion. Therefore, by expanding all the code, you remove the key concerns from an intellectual property standpoint while preserving the ability to reproduce the deliverables and therefore provide traceability in an audit.

Most pharmaceutical companies and CROs have big macro systems for dataset creation and reporting. The probability of these macros running on the FDA or client's systems is close to zero. These systems represent a substantial investment in intellectual property. So the question is: how do you provide the SAS code without it being impossible to follow, and without giving away vital methodology?

No reviewer wants to wade through macro code to figure what the program does. It's easier to read the expanded code the macros generate. You might not write a program that has large blocks of repetitive code, but for the purposes of clarity and traceability it is good practice, so long as a validated process creates it.

Both of the methods I discuss in this paper would count as a validated process. One method is to use MFILE/MPRINT options to write the expanded SAS code to a file. This code has all the macro variables resolved. Another method is to use EG as front end to design the program and write the code to a file. EG is an alternative to macros for generating SAS code.

I argue here that EG has several advantages over traditional macro coding. Chief among these is standardization of programming style. If you want to have a company-wide programming style, this is a good way to do it, provided you like the code EG generates. The fact that it is machine-generated has pluses and minuses. On the plus side, the syntax will be correct out of the box. The code will be well structured. EG provides some limited commenting automatically. On the down side, an experienced SAS programmer could certainly write more elegant code. EG generates some junk (EG-specific) code that has needs to be stripped out. The techniques EG provides are somewhat limited, although the ability to include code objects in the process flow and insert code into the built-in tasks can overcome these limitations.

## STANDALONE PROGRAMS

What is a standalone program? In the context of this paper, it means a program that is generated from a production process, and that can reproduce that outputs of that process. In a sense, the standalone program is a product of the system, just like the outputs. It contains all the information to reproduce the outputs.

## CHARACTERISTICS

Standalone programs have the following characteristics:

- All macro code and macro variables are resolved.
- Any input datasets and/or files are explicitly referenced. All libnames and external file references are in the program.
- No %includes: any referenced code is expanded and included in the program.
- Basic commenting. Detailed commenting for internal use should be stripped out.
- Simple standard header block, with program name, author, purpose, and date, noting it is the final, validated version. Internal header blocks with the revision history should be removed.

## REQUIREMENTS

There are three basic requirements for standalone programs:

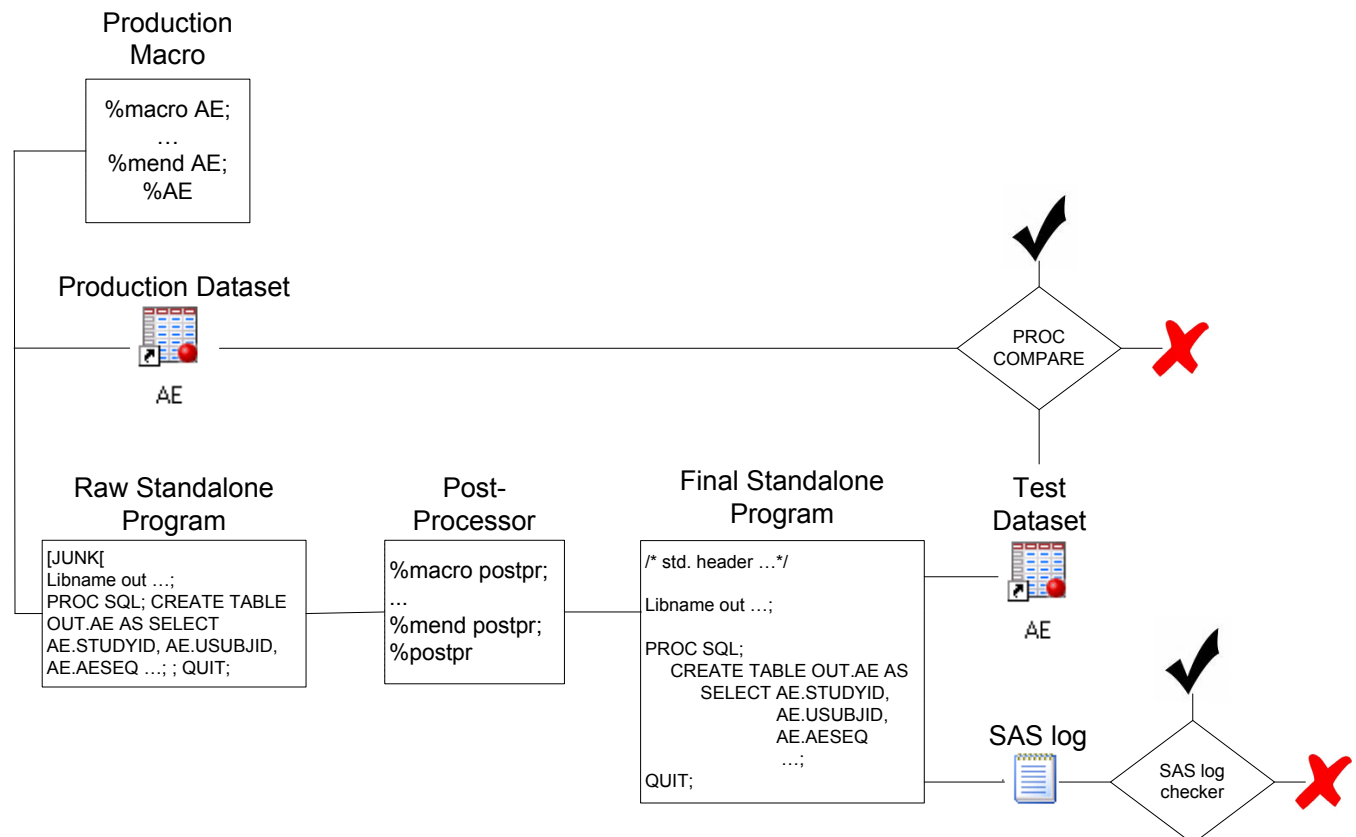
- To completely and accurately reproduce the outputs of the production system
- To be well formatted and easy to read.
- Stylistic consistency

It is easy to verify that the standalone program outputs match those from the production program. Datasets can be checked with PROC COMPARE. For outputs, .lst files can be compared either with an off-the-shelf file compare program, such as Beyond Compare, or file-compare SAS macro. RTF files can be compared using Microsoft Word VBA macros. The term “accurately” also means the program runs without errors or warnings. This can be easily verified with a log-checking macro.

I think the stylistic consistency EG facilitates goes a long way here. It simplifies the FDA reviewer’s task, and for CROs, makes it easier for the client to audit and accept the programs.

## STANDALONE PROGRAM CREATION AND VALIDATION PROCESS

The following diagram illustrates the standalone program creation and validation process from an analysis dataset macro:



The standalone program is considered validated only if the test dataset exactly matches the production dataset, and the SAS log does not have any errors or warnings. For programs that create tables and listings, the outputs have to match, except for the run date/time.

## METHODS FOR CREATING STANDALONE PROGRAMS

### MFILE METHOD

The first method is the "MFILE" method, where SAS creates standalone programs from macros through use of the MFILE and MPRINT options. Several papers have addressed this topic. To my mind, the seminal paper is "Capturing SAS Macro Code into an Executable SAS Program", David J. Jemiolo, DataCeutics, Inc., Pottstown, PA, NESUG, 2002. Other papers include those by Eddington and Zhou (2002) and Litzsinger & Riddle (2002).

The method has two steps:

1. Run the macro using OPTIONS MPRINT MFILE. This writes the expanded code from the macro to a file with all macro code and macro variables resolved.
2. Clean up the code with post-processor. This includes addition of standard header block, basic commenting, indenting, white space, or any other standards to be enforced.

Jemiolo (2002) states

"Although MPRINT and MFILE are very useful in capturing SAS Macro code, they still lack the ability to directly capture non-macro SAS code. The "trick" ... is to enclose the MPRINT, MFILE option, and %INCLUDE statements within a dummy macro called %\_RUNIT\_. When %\_RUNIT\_ is called, not only will the decoded SAS Macro calls be captured, but any other non-macro code that may exist in the original SAS program, will also be retained. All captured code is written to a text file..."

He presents the following example:

```
%macro _runit_;
filename incode "&dir.&del.&infile..sas"
lrecl=2048;
filename mprint "_temp_" lrecl=2048;
options mfile mprint;
%include incode;
options nomprint;
%mend _runit_;
%_runit_;
ods listing;
```

The text file will contain the completely resolved contents of &dir.&del.&infile..sas (the source program).

The major problem with this method is text wrapping across lines of code in the program. The line length limit for a SAS program to run is 256 characters. Therefore, somehow you have to get down to 256 from 2048 (the line length in the example). For example, suppose this is the source program:

```
Libname NTH "\\Sas1\data\nth\data";

PROC SQL;
CREATE TABLE WORK.Query_for_AE AS SELECT AE.STUDYID FORMAT=$15.,
      AE.USUBJID,
      AE.SEX,
      AE.AGE FORMAT=BEST22.,
      AE.SAFETY,
      AE.ARM,
      AE.AEBODSYS,
      AE.AEDECOD,
      AE.AESEV,
      AE.AESER,
      AE.AEACN,
      AE.AEOUT,
      AE.PHASE
FROM NTH.AE AS AE
WHERE AE.SAFETY = "YES" AND AE.AEDECOD IN ("AXILLARY PAIN", "BACK INJURY",
"BACK PAIN", "BACTERIAL INFECTION", "BALANCE DISORDER", "BARTHOLIN'S CYST",
"BASAL CELL CARCINOMA", "BENIGN BREAST NEOPLASM", "BENIGN NEOPLASM OF BLADDER",
"BENIGN PROSTATIC HYPERPLASIA", "BEREAVEMENT REACTION", "BLADDER DISORDER",
"BLADDER SPASM", "BLEPHAROSPASM", "BLINDNESS TRANSIENT", "BLINDNESS UNILATERAL",
"BLISTER", "BLOOD ALKALINE PHOSPHATASE INCREASED", "BLOOD BICARBONATE DECREASED",
```

```
"BLOOD BILIRUBIN INCREASED", "BLOOD CALCIUM INCREASED", "BLOOD CHLORIDE DECREASED");  
QUIT;
```

The file coming out of the MFILE MPRINT is likely to split one or more of the search terms across program lines, like this:

```
Libname NTH " \\Sas1\data\nth\data";  
PROC SQL;  
CREATE TABLE WORK.Query_for_AE AS SELECT AE.STUDYID FORMAT=$15., AE.USUBJID, AE.SEX,  
AE.AGE FORMAT=BEST22., AE.SAFETY, AE.ARM, AE.AEBODSYS, AE.AEDECOD, AE.AESEV,  
AE.AESER, AE.AEACN, AE.AEOUT, AE.PHASE FROM NTH.AE AS AE WHERE AE.SAFETY = "YES" AND  
AE.AEDECOD IN ("AXILLARY PAIN", "BACK INJURY", "BACK PAIN", "BACTERIAL INFECTION",  
"BALANCE DISORDER", "BARTHOLIN'S CYST", "BASAL CELL CARCINOMA", "BENIGN BREAST  
NEOPLASM", "BENIGN NEOPLASM OF BLADDER", "BENIGN PROSTATIC HYPERPLASIA",  
"BEREAVEMENT REACTION", "BLADDER DISORDER", "BLADDER SPASM", "BLEPHAROSPASM",  
"BLINDNESS TRANSIENT", "BLINDNESS UNILATERAL", "BLISTER", "BLOOD ALKALINE  
PHOSPHATASE INCREASED", "BLOOD BICARBONATE DECREASED", "BLOOD BILIRUBIN INCREASED",  
"BLOOD CALCIUM INCREASED",  
"BLOOD CHLORIDE DECREASED");  
QUIT;
```

Obviously, this search won't work. It does not appear possible to stop MFILE from splitting the text across lines. Therefore, it seems that the only solution is to fix it in the post-processor. This is not an easy problem though, and any solution along these lines could cause other problems as well.

In terms of standalone code generation, intelligent text wrapping is one of the main advantages of EG, which handles the text-wrapping automatically, keeping text in quotes together on one line.

#### ENTERPRISE GUIDE METHOD

Enterprise Guide is an object-oriented interface to the SAS system. The main interface is the process flow. Initially, it is a blank screen on which to arrange the objects. The programmer sets properties for objects, and the objects generate SAS code "behind the scenes". "Code objects" can also be included, containing user-written custom code, and these can be linked to the built-in objects in a process flow.

Hemedinger (2005) and Dmitrienko (2005) have good introductory papers on EG. Also SAS has a free online Enterprise Guide training course at [http://www.sas.com/apps/elearning/elearning\\_details.jsp?pubcode=59271](http://www.sas.com/apps/elearning/elearning_details.jsp?pubcode=59271).

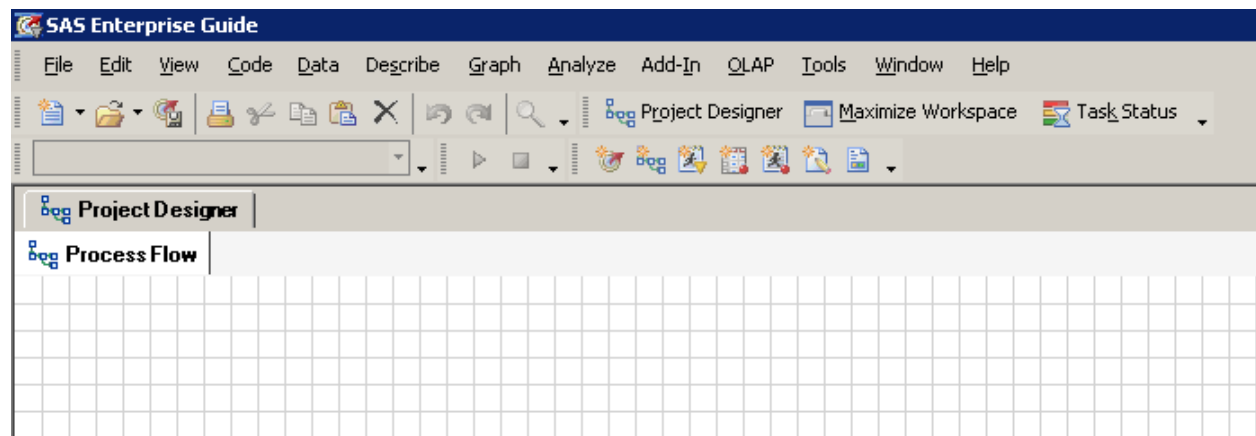
It is important to note that EG is designed to be run interactively by "power user". In order to get it to run in batch, you have to resort to some "tricks" that I describe below.

The process of generating standalone code with EG is as follows:

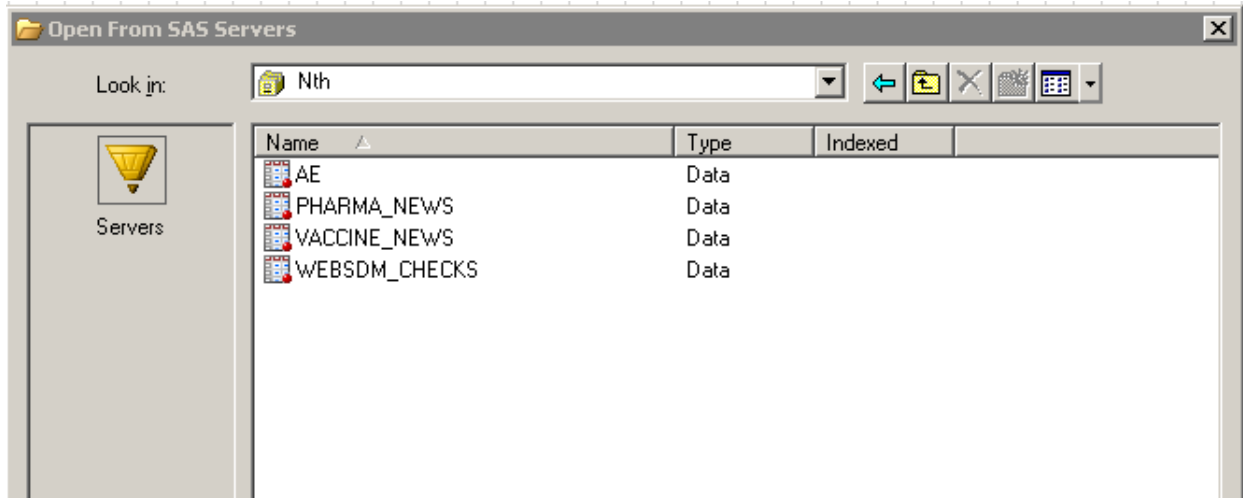
3. Create the program using the process flow.
4. Select all objects and create a stored process
5. Run the stored process through a post processor

For purposes of this paper, I treat EG as a code generator. In this sense, it's like the macro language. Macros expand out and generate SAS code; EG generates SAS code behind the scenes. Let's look at an example of EG doing a simple query.

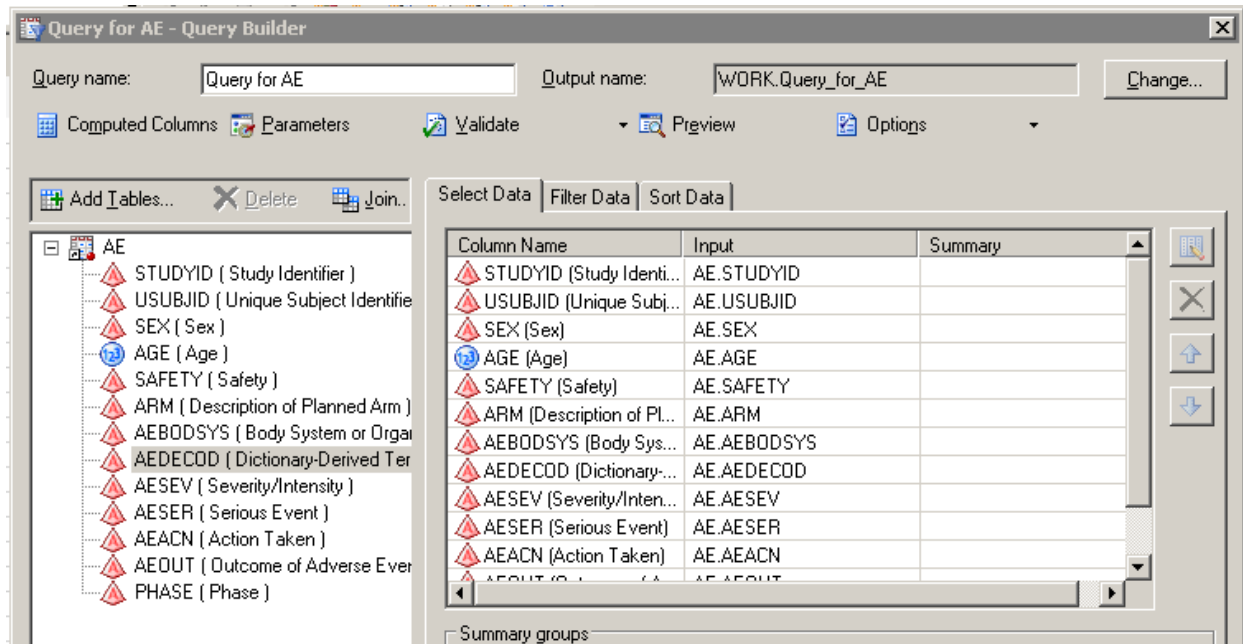
Start with a blank process flow:



Load the data from one of the libraries:



Build the query:



Build the select statement. One of the best things about EG is that it allows you to pick the valid values from a list:



Value	Formatted Value
ABDOMINAL ABSCESS	ABDOMINAL ABSCESS
ABDOMINAL DISCOMFORT	ABDOMINAL DISCOMFORT
ABDOMINAL DISTENSION	ABDOMINAL DISTENSION
ABDOMINAL PAIN	ABDOMINAL PAIN
ABDOMINAL PAIN LOWER	ABDOMINAL PAIN LOWER
ABDOMINAL PAIN UPPER	ABDOMINAL PAIN UPPER
ABDOMINAL TENDERNESS	ABDOMINAL TENDERNESS
ABNORMAL BEHAVIOUR	ABNORMAL BEHAVIOUR
ABNORMAL DREAMS	ABNORMAL DREAMS
ABNORMAL FAECES	ABNORMAL FAECES
ABORTION MISSED	ABORTION MISSED
ABORTION SPONTANEOUS	ABORTION SPONTANEOUS
ABSCESS	ABSCESS
ACCIDENT	ACCIDENT
ACCOMMODATION DISORDER	ACCOMMODATION DISORDER
ACIDOSIS HYPERCHLORAEMIC	ACIDOSIS HYPERCHLORAEMIC
ACNE	ACNE
ACRODERMATITIS	ACRODERMATITIS
ACTIVATED PARTIAL THROMBOPLASTIN TIME PROLONG...	ACTIVATED PARTIAL THROMBOPLASTIN TIME PR
ACTIVITIES OF DAILY LIVING IMPAIRED	ACTIVITIES OF DAILY LIVING IMPAIRED

It generates the SELECT statement for you:

**Edit Filter**

Column: AE.AEDECOD

Operator: In a list of values

Values:
 

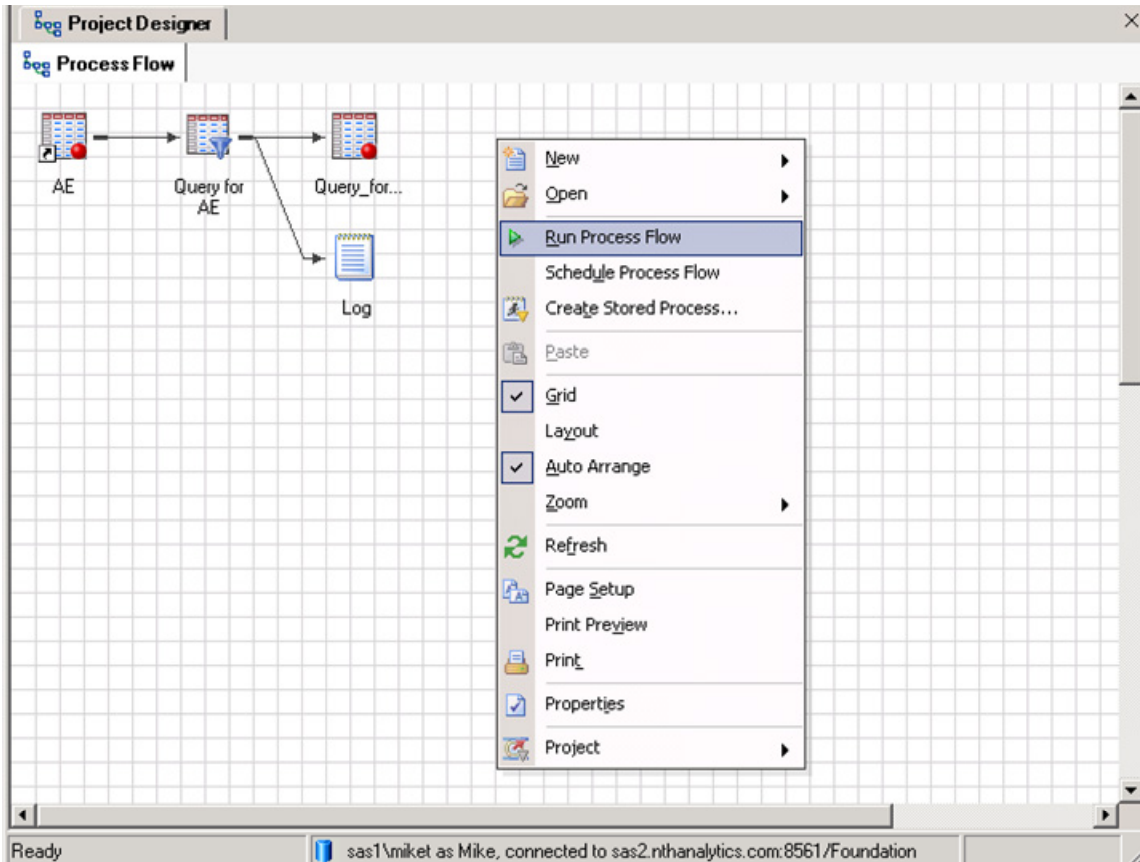
ABDOMINAL PAIN UPPER  
 ABDOMINAL TENDERNESS  
 ABNORMAL BEHAVIOUR  
 ABNORMAL DREAMS  
 ABNORMAL FAECES

Add  
Remove

Filter definition

```
AE.AEDECOD IN ('ABDOMINAL PAIN UPPER','ABDOMINAL TENDERNESS','ABNORMAL BEHAVIOUR','ABNORMAL DREAMS','ABNORMAL FAECES','ABORTION MISSED','ABORTION SPONTANEOUS','ABSCESS','ACCIDENT','ACCOMMODATION DISORDER','ACIDOSIS HYPERCHLORAEMIC','ACNE','ACRODERMATITIS','ACTIVATED PARTIAL THROMBOPLASTIN TIME PROLONGED','ACTIVITIES OF DAILY LIVING IMPAIRED','ACUTE SINUSITIS','ACUTE TONSILLITIS','ADENOIDAL HYPERTROPHY','ADENOIDITIS','ADVERSE DRUG REACTION','AEROPHAGIA','AFFECT LABILITY','AFFECTIVE DISORDER','AGEUSIA','AGGRESSION','AGITATION','AGNOSIA','AKINESIA','ALANINE AMINOTRANSFERASE','ALANINE AMINOTRANSFERASE INCREASED','ALCOHOL INTOLERANCE','ALCOHOL POISONING','ALCOHOL WITHDRAWAL SYNDROME','ALLERGIC BRONCHITIS','ALLERGIC SINUSITIS','ALLERGY TO ARTHROPOD STING','ALLERGY TO VACCINE','ALOPECIA','ALOPECIA AREATA','ALTERED VISUAL DEPTH PERCEPTION','AMBYOPIA','AMENORRHOEA','AMNESIA','ANAEMIA','ANAL INFLAMMATION','ANAL ULCER','ANGER','ANGINA PECTORIS','ANHEDONIA','ANHIDROSIS','ANIMAL BITE','ANIMAL SCRATCH','ANKLE FRACTURE','ANOREXIA','ANORGASMIA','ANOSMIA','ANTICONVULSANT DRUG LEVEL INCREASED','ANTICONVULSANT TOXICITY','ANTINUCLEAR ANTIBODY POSITIVE','ANXIETY','APATHY','APHASIA','APHONIA','APHTHOUS STOMATITIS','APPENDICITIS','APPETITE DISORDER','AREFLEXIA','ARRHYTHMIA','ARRHYTHMIA
```

Run the process flow:



The resulting query is written in PROC SQL. EG never generates dataset code. In a sense, by moving to EG, you are phasing out the dataset, except for special purpose “user interventions”.

#### STORED PROCESSES IN ENTERPRISE GUIDE

EG will automatically create a stored process from the process flow. A stored process is a macro designed to be run interactively through a SAS client: web browsers, EG, and SAS add-in for Microsoft Office being the major clients. It is not designed to be run in batch mode, because the parameters have to be passed in at run time interactively. However, you can make it run in batch by building several stored processes off the same source code with different default parameters. For example, you could have an adverse event table with a single parameter: BYGROUP. You could then create separate stored processes

- AE\_ALL: AEs with no subgroups (BYGROUP=)
- AE\_SEX: AEs by sex
- AE\_AGE: AEs by agegroup

To run them in batch, put objects for all three stored processes in the EG project. You need to create a VBScript program to batch the EG project, either from Windows Explorer or using the Windows scheduler. Both the output and the SAS log have to be directed to external files (otherwise they disappear when the project finishes).

#### STORED PROCESS SAS CODE

All stored processes have a source code file in the directory structure. It consists of EG code to make the stored process run, and normal SAS code. We want to get the normal SAS code into a standalone program.

This screenshot shows some EG-specific code:

```

SAS System Viewer - [AE.sas]
File Edit View Window Help

%STPBEGIN;

Libname NTH META ipaddr='sas2.nthanalytics.com' port=8561 protocol=bridge userid

* End EG generated code (do not edit this line);

/* --- Start of shared macro functions. --- */

/* Conditionally delete a table or view, if it exists */
/* If the member does not exist, then no action is performed */
%macro _eg_conditional_dropds(dsname);
  %IF %SYSFUNC(EXIST(&dsname)) %THEN %DO;
    PROC SQL;
      DROP TABLE &dsname;
    QUIT;
  %END;
  %IF %SYSFUNC(EXIST(&dsname,VIEW)) %THEN %DO;
    PROC SQL;
      DROP VIEW &dsname;
    QUIT;
  %END;
%mend _eg_conditional_dropds;
%macro _eg_inParam( DADM= TVDF= );

```

Here is the normal SAS code:

```

SAS System Viewer - [AE.sas]
File Edit View Window Help

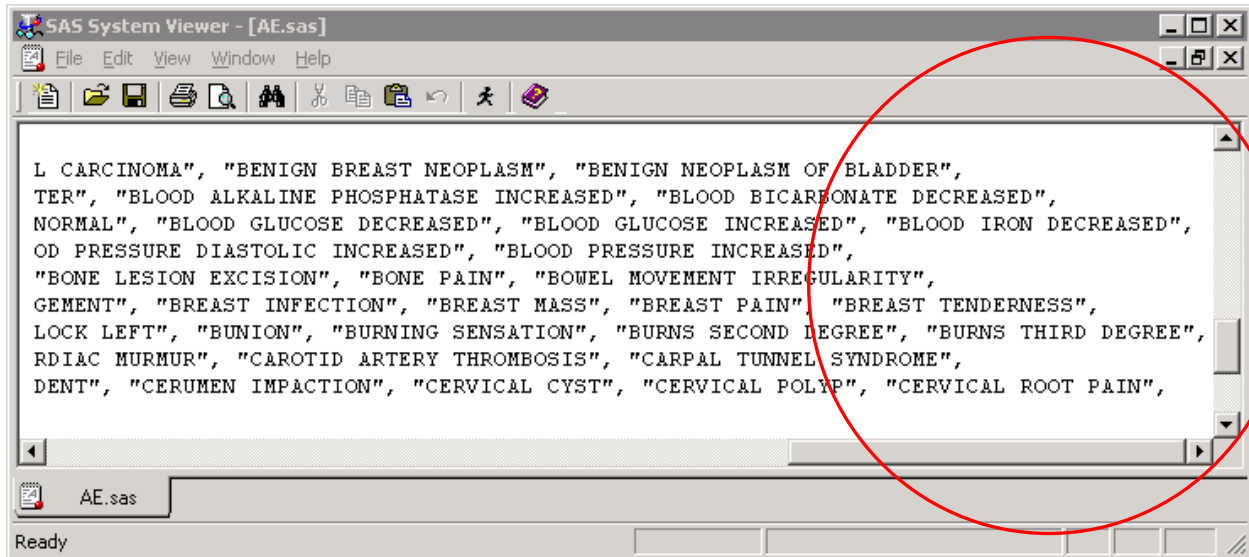
* End EG generated code (do not edit this line);

Libname NTH "\\Sas1\data\nth\data";

PROC SQL;
  CREATE TABLE WORK.Query_for_AE AS SELECT AE.STUDYID FORMAT=$15.,
    AE.USUBJID,
    AE.SEX,
    AE.AGE FORMAT=BEST22.,
    AE.SAFETY,
    AE.ARM,
    AE.AEBODSYS,
    AE.AEDECOD,
    AE.AESEV,
    AE.AESER,
    AE.AEACN,
    AE.AEOUT,
    AE.PHASE
  FROM NTH.AE AS AE
  WHERE AE.SAFETY = "YES" AND AE.AEDECOD IN ("AXILLARY PAIN", "BACK INJURY", "BACK PAIN"
" BENIGN PROSTATIC HYPERPLASIA", "BEREAVEMENT REACTION", "BLADDER DISORDER", "BLADDER SP
" BLOOD BILIRUBIN INCREASED", "BLOOD CALCIUM INCREASED", "BLOOD CHLORIDE DECREASED", "BL
" BLOOD LACTATE DEHYDROGENASE INCREASED", "BLOOD OSMOLARITY DECREASED", "BLOOD PHOSPHORU
" BLOOD TRIGLYCERIDES ABNORMAL", "BLOOD TRIGLYCERIDES INCREASED", "BLOOD UREA INCREASED"
" BOWEL SOUNDS ABNORMAL", "BRADYCARDIA", "BRADYKINESIA", "BRADYPHRENIA", "BRAIN NEOPLASM
" BREATH ODOUR", "BREATH SOUNDS ABNORMAL", "BRONCHIOLITIS", "BRONCHITIS", "BRONCHITIS AC
" BURSA DISORDER", "BURSITIS", "BUTTOCK PAIN", "CALCINOSIS", "CALCULUS URETERIC", "CALCU
" CARTILAGE INJURY", "CAT SCRATCH DISEASE", "CATARACT", "CATARACT OPERATION", "CATHETER
" CERVICOBRACHIAL SYNDROME", "CHALAZION", "CHANGE OF BOWEL HABIT", "CHEST DISCOMFORT");
  QUIT;

```

Notice that the stored process mechanism did do intelligent text splitting (as shown below), saving a tremendous amount of grief in the post-processor step.



If the search string in quotes is split across two lines in the program, it puts blank spaces in the middle of the string, and the search will not work. "BLOOD IRON DECREASED" becomes "BLOOD IRON DECREASED", for example. Of course, there are workarounds, but they are complicated. EG will do it automatically, and that is a tremendous advantage.

#### POST-PROCESSOR

For either method, you need to clean up the generated code with a post-processor. Jemiolo (2002) describes the basic concept for the MFILE method:

"The text file produced is all left aligned with a lack of hard returns between PROCs and DATA steps. To resolve these issues, a series of key words (i.e., PROC, DATA, RUN, and QUIT) are searched for. If these key words are encountered at the beginning of a line of text (not beginning with an "\*"), a blank line is placed either before or after the line, as appropriate. Conversely, if these key words are not encountered in a line of text (not beginning with an "\*"), then the line is indented by three spaces. It should be noted that each company has its own standards and tendencies and as a result, the list of key words should be changed appropriately to reflect these."

For EG, the concept is similar; but the problem is simplified because the code coming out of the stored process is well structured to start with. However, there is a lot EG-specific code that has to be stripped out to make it look like normal SAS program, as shown above.

#### LIMITATIONS OF ENTERPRISE GUIDE

For all advantages of EG, there are some significant limitations and disadvantages, aside from the learning curve, which can be fairly steep:

- Enterprise Guide has some unique features as a SAS programming environment. Therefore, SAS code from EG will not necessarily run in a SAS outside of EG. A testing step is required to verify that it does in fact run in batch or in the display manager. The major difference is that in EG, variable names can have spaces. For example, the following code is valid in EG:

```
PROC SQL;
CREATE TABLE WORK.Query_for_AE AS
SELECT AE.STUDYID FORMAT=$15.,
AE.USUBJID,
('test') AS 'variable name with blanks'n
FROM NTH.AE AS AE;
```

- Although EG has 80+ built-in tasks, it does not have a PROC REPORT task (and therefore does not have a built-in way to generate PROC REPORT code). You can write your own task as an add-in, but that's not easy and defeats the purpose of using an off-the-shelf product.
- There is no data step code generator in EG. Essentially, by going to EG, you are abandoning the datastep. I don't consider this a disadvantage. Datastep code can always be incorporated through code objects.

- There is no “set difference” task (i.e., using the EXCEPT operator in PROC SQL). This is one of the handiest features in PROC SQL, and the absence is a big limitation. It has to be done through a code object.
- Complex process flows can freeze up, and are unrecoverable in some instances.
- In a complicated process flow, tasks sometimes fail, forcing you to create an “ordered list”, which usually solves the problem.

## CONCLUSION

Despite some limitations, I think Enterprise Guide is the best way to generate standalone programs. It automatically creates acceptable SAS code, which, because it is standardized, simplifies the post-processing task necessary to create the final, polished version for submission to regulatory authorities. Enterprise Guide has some significant advantages over the alternative method of using the MFILE/MPRINT options to create standalone versions of production macro code. In particular, Enterprise Guide performs intelligent text splitting (i.e., keep code in quotes on the same line) when generating SAS code, while the MFILE/MPRINT method does not, greatly complicating the post-processing step.

## REFERENCES

FDA Reviewers as Ultimate End Users: Using the SAS System® to Construct e-Submissions that Actually Facilitate the NDA / BLA Review Process, Rich Vachal, Industry Dynamics Associates, Somerset, NJ, PharmaSug 2001

Post-Processing MPRINT Outputs to Generate Macro-Free Code, Jim Edgington and Jay Zhou, Quintiles, PharmaSUG, 2002.

Boost Your Programming Productivity with SAS® Enterprise Guide Chris Hemedinger, SAS Institute Inc., Cary, NC SUGI 30, 2005.

SAS Enterprise Guide in Pharmaceutical Applications: Automated Analysis and Reporting, Alex Dmitrienko, Ph.D., Eli Lilly and Company, Indianapolis, IN, MWSUG, 2005.

A Modular Approach to Portable Programming, Litzsinger & Riddle, Quintiles, SUGI 27, 2002

## CONTACT INFORMATION

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

Mike Todd  
 Nth Analytics  
 12 Crimson King Trail  
 Flemington NJ 08822  
 908 672 5649  
 Fax: 253.595.7413  
 E-mail: [info@nthanalytics.com](mailto:info@nthanalytics.com)  
 Web: [www.nthanalytics.com](http://www.nthanalytics.com)

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.