Paper BI09-2012

BI-09 Using Enterprise Guide® Effectively

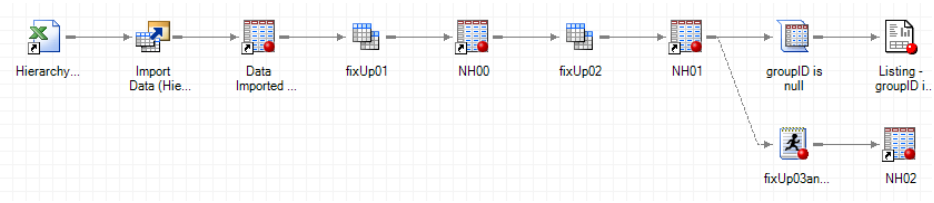
Tom Miron, Systems Seminar Consultants, Madison, WI

ABSTRACT

Enterprise Guide is not just a fancy program editor! EG offers a whole new window onto the power of SAS software. Enterprise Guide requires a different perspective on system design than the use of straight SAS coding. This paper discusses use of the Enterprise Guide environment and features of Enterprise Guide that can make your applications more user friendly and maintainable. We will look at client-server architecture issues, visual v. text layout, and EG best practices.

WHAT IS SAS ENTERPRISE GUIDE?

Enterprise Guide is not SAS. Enterprise Guide is a user interface for SAS. EG allows you to assemble graphical objects that represent data processing tasks. The tasks are connected in a sequential chain. The tasks usually create data objects such SAS tables, spreadsheets, and reports. Tasks and data are represented by distinct icons. Below is a typical chain of EG objects:



Linked EG objects that make up a process flow

Here, the first task, labeled “Import Data…”, imports a spreadsheet named Hierarchy and creates a SAS table. That table serves as input to a query object named “fixUp01” and so forth.

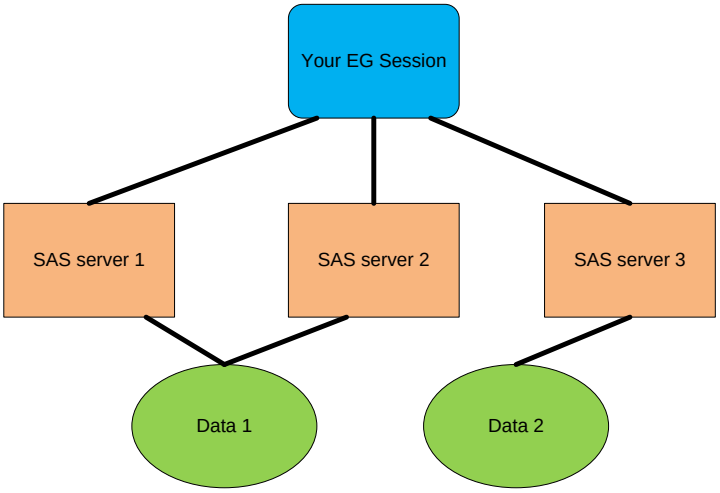
In EG, chains of tasks and data are grouped into “process flows.” And one or more process flows are stored as an EG project.

The tasks represented by icons in a process flow are not executed by or within EG. The data represented by process flow icons is not stored by EG. Enterprise Guide acts only to “surface” these tasks and data to you, the user, in a manner that allows you to create and manipulate them as graphical objects rather than through procedural program statements.

SAS ENTERPRISE GUIDE ARCHITECTURE

If programs are not run in EG where are they run? Enterprise Guide works in conjunction with servers. Servers are instances of SAS running somewhere on the network accessible to your Enterprise Guide session. EG generates SAS program statements based on the objects in your project and passes this code to a server for execution.

Usually, data represented in your EG project exists in the server’s domain. For example, the SAS table “NH01” in the process flow shown above exists in a directory accessible and known to the server that is actually running the underlying SAS code. This may be, but usually is not, on the PC on which you are running Enterprise Guide.

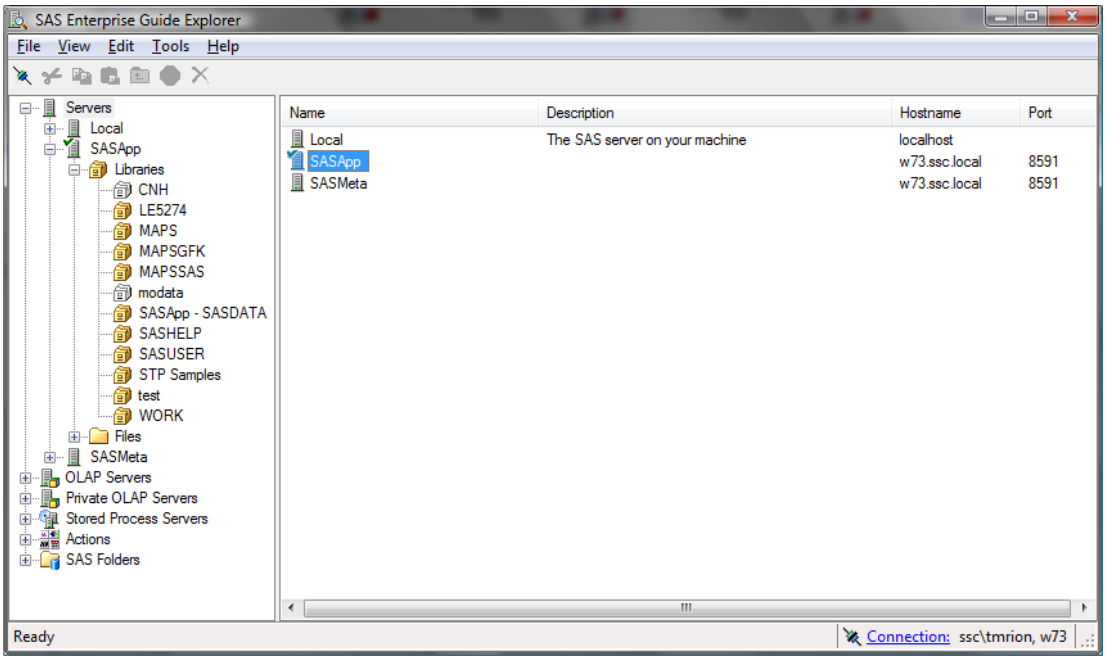


Your Enterprise Guide session relies on servers to run tasks and access data.

CLIENTS AND SERVERS

We will not cover the SAS client-server architecture here, but keep in mind that to run Enterprise Guide you must be connected to a server. In most environments this server structure will be set up for you with pre-defined folders and SAS libraries, including connections to external databases such as Oracle or DB2.

You may encounter the terms “workspace server”, “metadata server”, and “object spawner.” These refer to components of the client-server architecture that must be in place to run EG.



Connected servers and known libraries listed in the EG Explorer window.

CLIENT SERVER CONSIDERATIONS

Your EG session is running on your local Windows PC but the server that is executing the SAS programs generated by your session may be running under a different operating system. A common configuration is UNIX SAS servers connected to Windows EG sessions.

Data you refer to in your EG process flows may reside on your local PC or on a network drive under an operating system that differs from the SAS server OS. Again, a mix of Windows and UNIX is common.

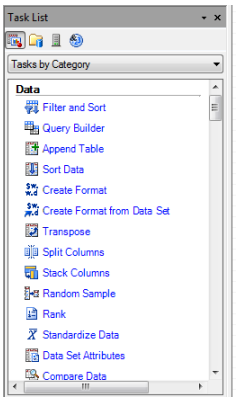
You need to be aware of differences in operating system naming conventions and data representations. For example:

Names are case sensitive on UNIX. So if you refer to a file that resides on the server's file system it will be case sensitive.

Text file lines are terminated by carriage return + line feed on Windows and line feed alone on UNIX. If you attempt to read a text file from the PC where you are running EG and your server is UNIX, it won't work correctly. *(The SAS option TERMSTR= allows you get around this particular problem. See Usage Note 14178.)*

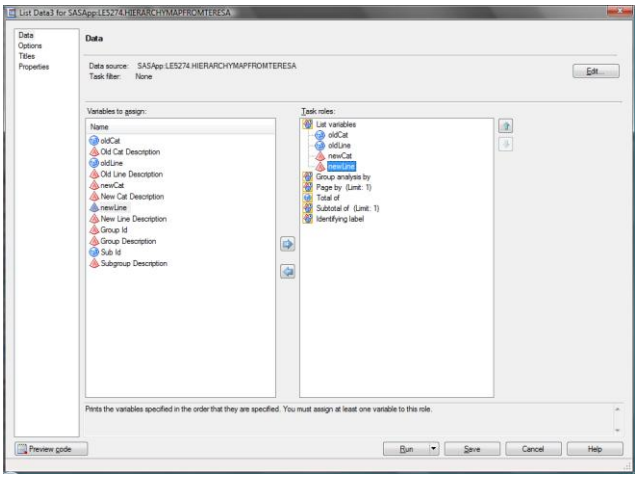
EG PROJECTS VS. SAS PROGRAMS

In Enterprise Guide you work with "projects." An EG project is a single file with extension .egp. When you open a project you see a workspace grid. On this grid you assemble and connect discrete data processing tasks selected from the EG task list.



Task List

Tasks represent defined processes. Most tasks run a specific SAS PROC. For example the List Data task runs PROC PRINT. The task interface or wizard presents valid options and settings.



The "Data" properties window for the List Data task

You control the task through a point-and-click, drag-and-drop interface, not by writing syntactically correct statements. In the task interface only valid options and parameters are presented and minimum requirements are enforced before attempting to run the task.

Context sensitive Help is available within the interface as you build the task.

CIRCLE-SLASH PROGRAMMING

Below is the SAS code generated by EG for a List Data task. This task is based on PROC PRINT. You can see the generated PROC PRINT step plus other start up and clean up code. Instead of using the PROC PRINT VAR statement to select columns, a separate SQL step creates a view with only the columns to print. This view is deleted at the end.

```
/* ---------------------------------------------------------------------
   Code generated by SAS Task

   Generated on: Thursday, August 30, 2012 at 5:23:55 PM
   By task: List Data3

   Input Data: LE5274.HIERARCHYMAPFROMTERESA
   Server:  SASApp
   --------------------------------------------------------------------- */

%_eg_conditional_dropds(WORK.SORTTempTableSorted);
/* ---------------------------------------------------------------------
   Sort data set LE5274.HIERARCHYMAPFROMTERESA
   --------------------------------------------------------------------- */

PROC SQL;
    CREATE VIEW WORK.SORTTempTableSorted AS
          SELECT T.oldCat, T.oldLine, T.newCat, T.newLine
    FROM LE5274.HIERARCHYMAPFROMTERESA as T
;
QUIT;
TITLE;
TITLE1 "Report Listing";
FOOTNOTE;
FOOTNOTE1 "Generated by the SAS System (&_SASSERVERNAME, &SYSSCPL) on
%TRIM(%QSYSFUNC(DATE(), NLDATE20.)) at %TRIM(%SYSFUNC(TIME(), TIMEAMPM12.))";

PROC PRINT DATA=WORK.SORTTempTableSorted
    OBS="Row number"
    LABEL
    ;
    VAR oldCat oldLine newCat newLine;
RUN;
/* ---------------------------------------------------------------------
   End of task code.
   --------------------------------------------------------------------- */
RUN; QUIT;
%_eg_conditional_dropds(WORK.SORTTempTableSorted);
TITLE; FOOTNOTE;
```

If you were writing a SAS step to print this table your program would be a lot shorter and simpler than the code generated by EG above.

Two points apply to almost any EG task:

1. The code you can write will always be more compact and efficient than that generated by EG.
2. It doesn't matter.

Life-cycle efficiency is highly dependent on development and maintenance time and, except in extreme cases, has little to do with the number of statements being executed or the absolute machine efficiency of the coding. In other words it doesn't matter if Enterprise Guide generated 100 statements where you could have done it in 10 because you didn't have to write any statements at all. And, you don't have to debug them.

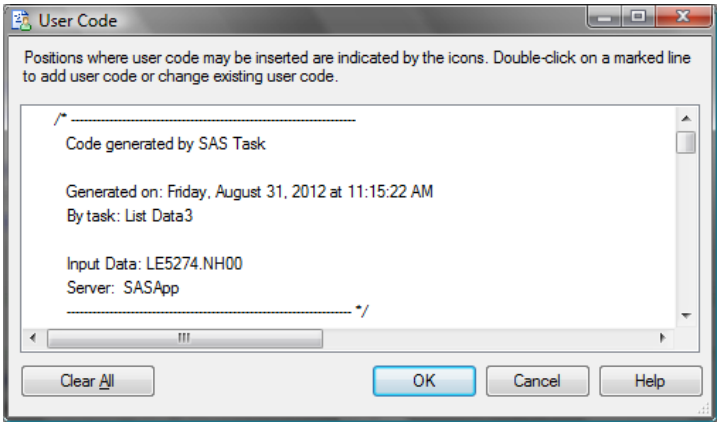
New EG users that are experienced SAS programmers are often tempted to open an EG program object and key in statements just as they did in the SAS Display Manager program editor. Effective use of EG rests on not writing SAS code. In fact, the rule for EG is:

Avoid programs and custom code like the plague.

WHY NOT PROGRAM?

One great advantage of using Enterprise Guide for system development lies in the fact that EG tasks are not open ended. They cannot be arbitrarily modified or extended. When you use an EG task you must work with the options presented in the interface. This is an advantage because the task is self-documenting and consistent wherever it's used. A List Data task is the same in any EG project, whereas, a PROC PRINT step can be written with different coding styles and options.

The downside of working within the constraints of an EG task is that there may be something you need to do that is not available as a task option. This situation makes it tempting to write a program or insert user code into the generated task code.



Insert user code window

Once you insert user code (or write a program) you leave the standardized, self-documenting world of EG tasks and enter the black box world of procedural code.

- To maintain an EG project that contains no user code or programs you have to know Enterprise Guide.
- To maintain an EG project that contains user code you have to know Enterprise Guide and SAS and understand how the SAS code works and know where the code is located in the project.

User code and programs require specific documentation. With EG tasks you may need documentation describing why tasks are strung together as they are but you do not need to explain how the tasks work. That information is built into the task interface and context Help.

THE ANSWER TO TASK LIMITATIONS: MORE TASKS

Against our intuition to pare things down, the way to get around the limitations of a single EG task is to add more tasks to get the job done.

Example:

On two tables with a common key you need to find rows that are on both tables and rows that are unique to one or the other table.

Let's look at a table of customers sent a promotional email and a table of customers that purchased a product. You need three result tables:

1. Customers that received the promo email and didn't make a purchase. (On promo table, not on purchase table)
2. Customers that made a purchase but didn't receive the email. (On purchase table, not on promo table)
3. Customers received the email and made a purchase. (On both promo and purchase tables)

Using a DATA step with a MERGE statement, the IN= option, and OUTPUT statements, you can create all three of these tables in a single DATA step with only one read of the tables. There is no single EG task that can do this. But you can do it with three Query tasks without writing code.

"But wait" you say. "That involves three passes of the data vs. one with a DATA step." That's right. But as long as that doesn't put you up against a practical execution time constraint and the marginal cost of the resource use, i.e., accessing the tables, is around zero, who cares? You have a modular, self-documenting system in which everything it does is presented through a nice graphical interface.

Every bit of custom code introduces a black box that needs to be analyzed before maintenance work can be done on the system. The "no code, more tasks" method can almost always be used to avoid custom coding and reduce the learning curve for anyone that has to maintain the system in the future.

USE PROCESS FLOWS (LOTS OF 'EM)

New Enterprise Guide users often build entire projects within a single process flow. Sometimes these process flows contain dozens of tasks with multiple inputs and outputs. Process flows with lots of tasks become hard to read because of the number of task link lines and the scrolling required to view everything in the process flow. Consider this rule:

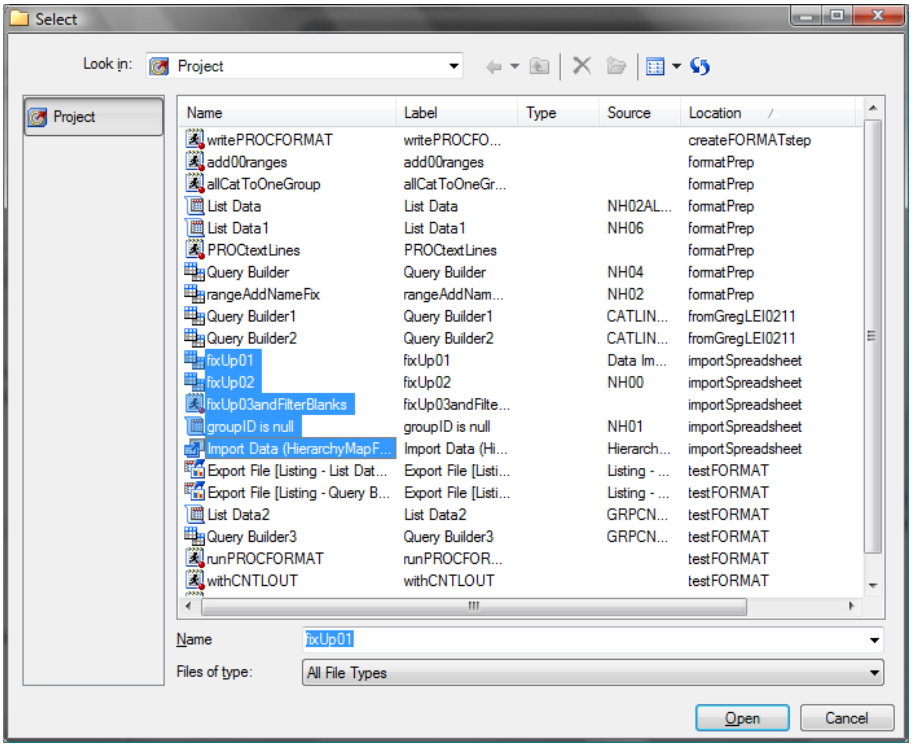
One function = one process flow

Each discrete function of your system should have its own process flow. Examples:

- Set SAS options
- Establish SAS libraries or external file FILEREFs
- Import data from a spreadsheet or database
- Validate or clean data
- Create a report
- Export results to PDF's or spreadsheets

There are several benefits of the one function = one process flow method.

1. Each process flow workspace has a minimal number of objects so it's easier to grasp what's going on.
2. The EG project is modular and can be developed, tested, and run step-by-step.
3. Using consistent, meaningful process flow names the project tree acts as a high level flow chart of your project and helps minimize written documentation.
4. Using consistent, meaningful process flow names you can quickly navigate through a complex project with the EG project tree.
5. You can create ordered lists of functionally related tasks based on the process flow location.



Select tasks for an ordered list by process flow location.

COMMON PROCESS FLOWS

As production level systems are developed with EG, it's useful to establish standardized process flows with consistent names. Following are some suggestions:

Process Flow Name	Function
Documentation	Use named Note objects for text documentation of your project. Notes might be: "Project Description", "Change History", "Input and Output", etc.
Start Up	Project initialization code. Possibly setting SAS options, timing or other project-wide macro variables, etc.
Wrap Up	End of project code: clear temporary tables, reset options, clear macro variables, clear filerefs and librefs, show timings, etc.
SAS Libraries	Establish SAS librefs not automatically available through your server.
External Files	Establish filerefs
Utility	Various tasks used during development: clear tables and reset macro variables, set OBS= or other testing options, etc.
Junk Drawer	A catchall for tasks under development, ad hoc tasks, etc.

Unfortunately, in EG 4.3 at least, you cannot apply an execution condition test to a process flow, only to a task node. This means that development process flows such as "Utility" and "Junk Drawer" suggested above cannot be bypassed by simply setting a macro variable to say "dev" or "prod." You may need to remove them in your production candidate system or create ordered lists that exclude non-essential tasks.

ONE TABLE = ONE PROCESS FLOW

In addition to the common process flows suggested above, try to use one process flow for each bit of output. If you are creating lots of intermediate tables this may not be practical or necessary. But it's useful to have a separate process flow for each table you create, even intermediate tables. This allows you to start up anywhere in the project during development and testing. It also makes maintenance easier because changes can be isolated and tested in a single process flow.

Create a separate process flow for each final output: reports, tables, listing, spreadsheet, etc. Again, this allows you to quickly find and modify your output without impacting parts of the EG project.

INTERMEDIATE TABLES

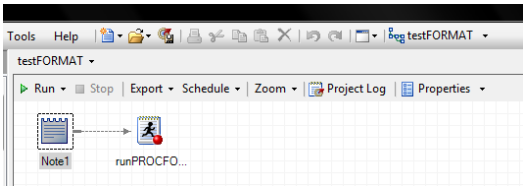
A consequence of avoiding custom code, discussed above, is that you will end up with more intermediate tables. Each functional component of your project results in a table passed to another component for further processing. Again, modularity and clarity are the benefits. You will have more pieces, i.e., process flows, but with consistency and good naming this will not be a problem.

Disk space is relatively inexpensive. It's usually practical to save intermediate tables. When you retain these tables it makes it much easier to debug and test changes. Unless you have a good reason not to save intermediate tables, do it.

It can be useful to establish a separate SAS library (folder) for intermediate tables, as opposed to final output tables. This allows you to clean out intermediate data if necessary and makes it clear where the "official" output of your EG project is without the confusion of other files in the folder. Your final output folder may be shared with other users while the intermediate table folder is not. If required or desired to work within a single folder, use a naming convention to distinguish working tables from final output.

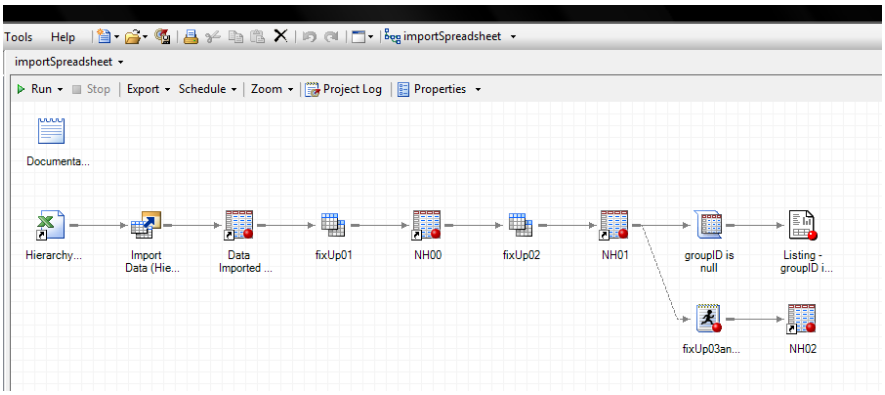
USE OF NOTE OBJECTS

The Note object is a great way to document processes and embedded SAS code (if you must). Right click on a Note and use Link Note To… to draw an arrow to the object documented.



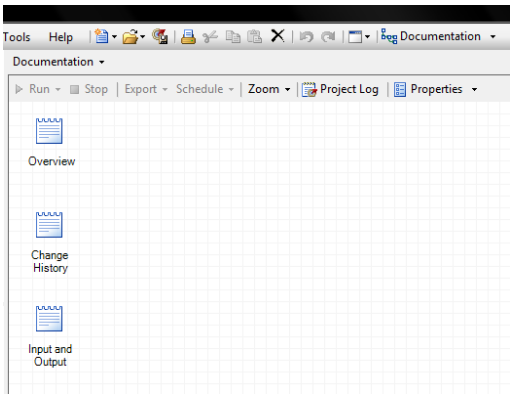
A Note object documenting an embedded SAS program

Place a "Documentation" Note at the top of each process flow to describe what it does.



Note object as process flow documentation

Add a “Documentation” process flow at the top of your project with Notes for project description, changes, etc.



“Documentation” process flow

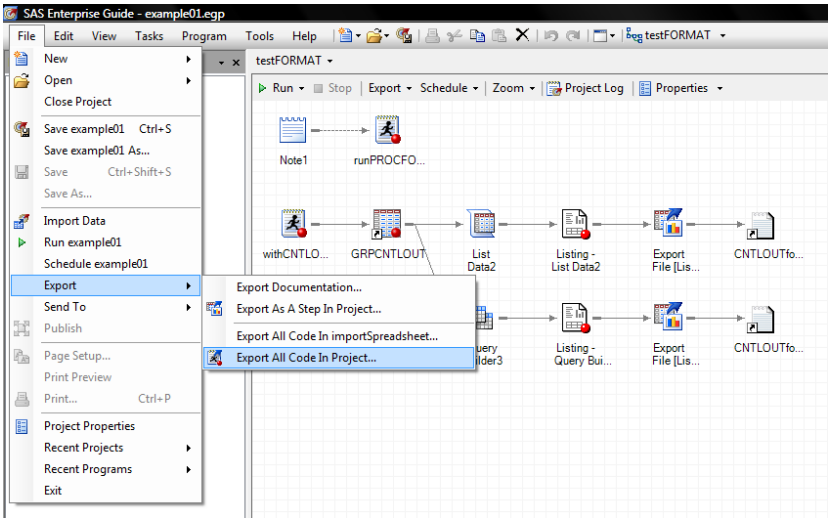
You can paste into Notes and print them from within EG. Be aware that Note text is not saved when you Export All Code… as discussed below. You can export individual Notes to text files.

PLAIN TEXT BACKUP

Your entire Enterprise Guide project is stored in a single file. Unlike SAS programs an EG project file is not stored as plain text. If your EG project file is corrupted you cannot recover by simply opening the .egp file in a text editor and cut and paste the code.

1. Always backup your project file (.egp) to a safe and accessible location. Thumb drive, network drive, etc.
2. Always backup your project as plain text with Export All Code…

Item 2. means use the Export function as shown below. This creates a .sas file. While you can’t recover your project directly from this file at least you will have a plain text file that can be used, with some work, to recover. Better than the disaster of losing the entire project.



Export All Code In Project saves generated SAS code as plain text

WRAP UP

- Effective use of SAS Enterprise Guide requires that you think visually rather than in terms of efficient SAS statement coding. A good EG project is organized around a clear and logical visual display of linked components that requires a minimum of additional written documentation.
- Avoid the use of custom SAS code, either as program objects or user modifications of EG tasks. When you use custom code you introduce a black box process that doesn't benefit from the built-in task dialogs or context Help.
- Don't be afraid of using multiple EG tasks to accomplish something that could be hand coded a single SAS step.
- Use intermediate result tables to stage data through your project.
- Use multiple process flows to break your project into logical components.
- Use standard names for common process flows ("Documentation", "Start Up", etc.)
- Make use of Note objects linked to their subject to document specific processes.
- Back up your project as both a .egp file and as plain text SAS code.

CONTACT INFORMATION

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

Tom Miron
Systems Seminar Consultants
2997 Yarmouth Greenway Dr.
Madison, WI 53711
608 625-4541
tmiron@sys-seminar.com
www.sys-seminar.com