



Infrastructure Designed to Maximize
Workflow

PhUSE 2018

All clinical statistical programmers work within a common set of constraints:

- ❖ we must be prepared to support the production of Clinical Study Reports after a trial is complete
- ❖ we must be prepared to support filings to various regulatory agencies; we must operate under the aegis of CFR 21 Part 11
- ❖ we must contend with frequent changes in specifications and contracting timelines
- ❖ most of us use SASTM software in our work.

This paper presents an architecture purporting to maximize programmer workflow, minimize manual steps, and optimize programmer efficiency. Guiding principles include:

- ❖ single source of metadata, stored locally
- ❖ all processes driven by metadata and automated through software
- ❖ hands-off production of finished publishable products
- ❖ minimizing human-only tasks wherever possible.



- ❖ SAS server running under Windows Server
- ❖ Coders VPN into the server, and typically develop code in SAS Display Manager System (version 9.4M5)
- ❖ All SAS resources (code, datasets, format libraries, template stores, style templates, user-defined functions, and macro libraries) are stored under folders in the server's C:\ drive
- ❖ Having contractors use their own Windows or Mac/OS machine means they can onboard as quickly as IT can set up an account and enable access to the system
- ❖ Having one copy of SAS to maintain minimizes redundancy and simplifies continual efforts such as applying SETINIT or Hot Fixes.

There are three layers to customizing the SAS System:

- ❖ the main configuration file
(!!SASroot \SASFoundation\9.4\nls\en\sasv9.cfg)
- ❖ autoexec file(s)
- ❖ Windows shortcuts
- ❖ We have not had occasion to modify the Configuration file as provided by SAS Institute during the install process
- ❖ all localization takes place through using autoexec and shortcut icons pointing to appropriate folders.



Folder Structure

Within study folder structure is standard

- ❖ `\Tables` `\Listings` `\Figures` are obvious, each containing `\code` and `\output` folders
- ❖ study-level folders `\docs` (Protocol, SAP, Table shells, any supporting documents for the study)
- ❖ `\macros` (project level macros; rarely used)
- ❖ `\projmeta` for project level metadata (much more about this later)
- ❖ `\explore` folder holds any reports or data transmissions requested during or after the study but not supported by the SAP
- ❖ Project-critical e-mails are printed into PDF documents in the `\docs` folder.

Project Initiation

- ❖ All of these folders are created in the DEV environment (then copied over to PROD just before database lock and unblinding)
- ❖ A tool to automatically create the study folders would be helpful, but as it only takes a few minutes to set up (and rarely), automating this process is of low priority
- ❖ Inside each `\code` folder, the first file created is `[autoexec.sas]`, always named exactly that

Project Initiation

- ❖ Many workplaces require the first line of executable code in any project file to be something like

```
%include "\study\folder\stage\code\setup.sas";
```

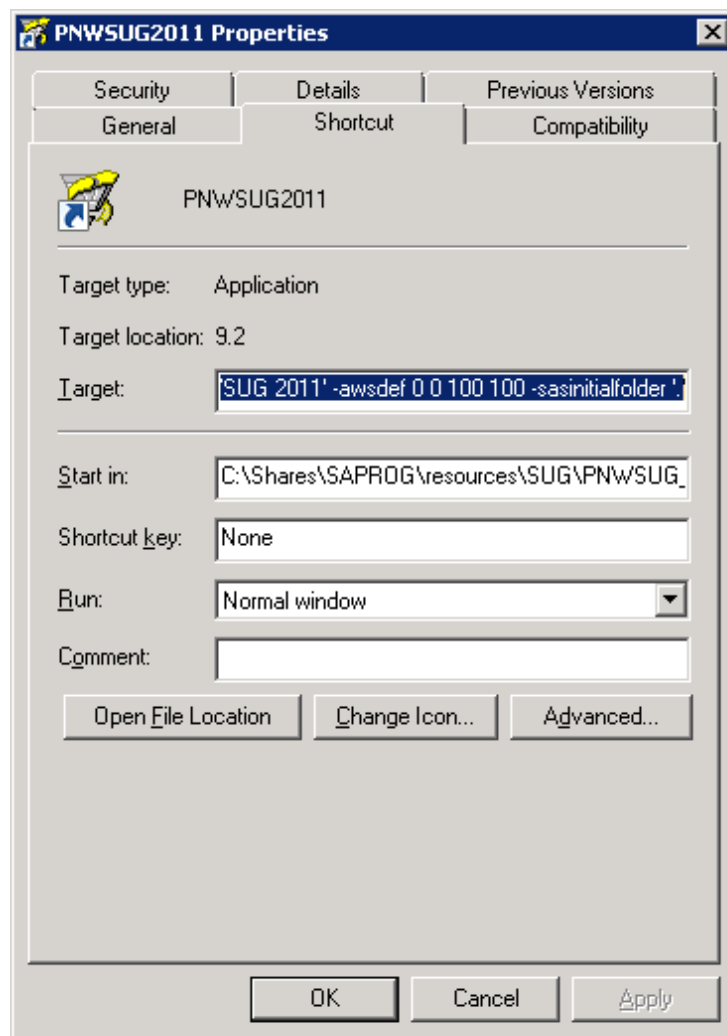
- ❖ The SAS system supports the concept of a “magic file” containing codes that one wants to have automatically executed upon initiation of a SAS session (either interactive or batch)
- ❖ When switching modes from Development to Production, copy over a handful (SDTM, ADaM, T, L, F) of autoexec files from DEV to PROD, and make the changes in those files



Windows Considerations

- ❖ This section assumes that you are coding SAS using the old-fashioned Interactive Development Environment (IDE)
 - ❖ Display Manager System (DMS)
- ❖ If instead you use
 - ❖ external editor to code, and run SAS in batch mode
 - ❖ use SAS Enterprise Guide or SAS/BI, or other modern products to access the power of SAS

then this section is irrelevant.



- ❖ In the [Start in] box, place the folder name where the code will be stored
- ❖ In conjunction with the [-sasinitialfolder `.`] in the previous [Target] box, sets the [Start in] folder as the current location for the SAS session
 - Inside SAS, **File** -> **open** will start in this folder
 - Build a special file called [autoexec.sas]. In here place any LIBNAME, FILENAME, OPTION etc. statements
 - When SAS is booting, will auto-detect the autoexec file and execute it
 - When the SAS session finishes startup you are all ready to go

- If you have a small # of projects, keep all in one folder with each project name on a separate shortcut
- If you have a larger # of projects, or if projects have sub-projects with multiple places to store related but separate code, create overall Project folders and as many shortcuts inside as each one needs
- Creating the first shortcut correctly is the hard part. Then, just copy and paste them, rename the new copy, and change stuff inside (usually just the Title in [Target] line and folder location in [Start In] line)



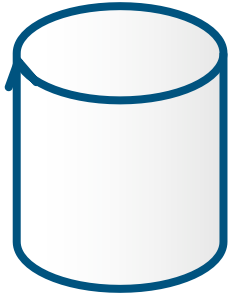
Project Metadata

We employ the typical 3-level system for SAS resources (format catalogs, macros, styles etc.) needed in a project

- ❖ Generic resources are located high in the folder hierarchy
- ❖ Project level resources are stored in the local project folder
- ❖ Individual program code can override these with entries created in the [work] library

There have been myriad papers written over the last three decades about how to set this up, and our setup is completely ordinary in this respect.

Project Metadata: Exception



Most shops store project-related metadata (Table #, Patient Populations, Titles / Footnotes etc.) in a project-level data store

- ❖ Most commonly, an Excel spreadsheet
- ❖ Chose instead to store in simple SAS data set
 - ❖ No easy way to read/write Excel
 - ❖ No data conversion / validation concerns
- ❖ Locally designated as `[projmeta.projman]`

Production code starts with:

- ❖ Standard comment section
- ❖ ~~[%include ...]~~ statement
- ❖ Call to standard enterprise-wide macro [%getPgmName ()]
- ❖ This macro combines the functionality of two macros from the SAS web site [www.support.sas.com]
- ❖ Creates a global macro variable in the current (interactive or batch) session containing the name of the program being executed
- ❖ Thus every program is self-aware of “who it is” without the programmer burden of hard coding [%let pgmname=XXX;]
 - ❖ avoids a potential disaster if a program is copy-paste-rename-tweaked while forgetting to re-tweak the hard coded name



Table Architecture

We view each Table (and most Listing) report shells through an imposed structure of (up to) three “meta” variables that define the table structure:

Page #	<code>pagenum</code>
Panel #	<code>panel</code>
Row	<code>row</code>

Table Architecture

Parameter Category/ Statistics	Group A (N=XX)	Group B (N=XX)	Total (N=XX)	meta
Age (Year) N	xx	xx	xx	Page 1
Mean (SD) Median	xx.x (xx.xx) xx.x	xx.x (xx.xx) xx.x	xx.x (xx.xx) xx.x	Panel 1
Min, Max	xx, xx	xx, xx	xx, xx	Rows 1-3
Gender	xx (xx.x%)	xx (xx.x%)	xx (xx.x%)	Page 1
-- Male	xx (xx.x%)	xx (xx.x%)	xx (xx.x%)	Panel 2
-- Female				Rows 1-3
Ethnicity	xx (xx.x%)	xx (xx.x%)	xx (xx.x%)	Page 1
-- Hispanic or Latino	xx (xx.x%)	xx (xx.x%)	xx (xx.x%)	Panel 3
-- Not Hispanic or Latino				Rows 1-3
Race	xx (xx.x%)	xx (xx.x%)	xx (xx.x%)	Page 1
-- American Indian or Alaskan Native	xx (xx.x%)	xx (xx.x%)	xx (xx.x%)	Panel 4
-- Asian	xx (xx.x%)	xx (xx.x%)	xx (xx.x%)	Rows 1-7
-- Black or African American	xx (xx.x%)	xx (xx.x%)	xx (xx.x%)	
-- Native Hawaiian or Other Pacific Islanders	xx (xx.x%)	xx (xx.x%)	xx (xx.x%)	
-- White	xx (xx.x%)	xx (xx.x%)	xx (xx.x%)	
-- Other				
Overall Interpretation of Electrocardiogram (ECG)	xx (xx.x%)	xx (xx.x%)	xx (xx.x%)	Page 1
-- Normal	xx (xx.x%)	xx (xx.x%)	xx (xx.x%)	Panel 5
-- Abnormal/Not Clinically Significant ¹	xx (xx.x%)	xx (xx.x%)	xx (xx.x%)	Rows 1-4
-- Abnormal/Clinically Significant ¹				

- ❖ Because each category of analysis is either completely specified or has an “Other” option, all subjects will fit into just these breakdowns, regardless of the number of subjects in the study
- ❖ Therefore this table may be considered as “well formed” or “completely specification driven”, as opposed to more data-driven reports
 - ❖ any Adverse Event table which is driven by the number of events occurring in each Panel=System Organ Class, or Panel=Preferred Term
 - ❖ Medical History findings

Table Architecture Meta Fields

Adopting this 3-level of organization requires a small adjustment period but reaps myriad benefits

- ❖ Code organization
- ❖ Code naming conventions
- ❖ Pre-data real table shells
- ❖ Possibility of multiple programmers collaborating on a large complex table
- ❖ Validation / Reconciliation process



Naming Conventions

Many production SAS shops have some sort of preferred naming convention

- ❖ Using the Page / Panel / Row convention this becomes easy and intuitive
- ❖ Each separate panel of output can be built separately, and all of the intermediate datasets produced during the derivation (not source datasets that are used by multiple panels) can be safely erased at the end of each panel section of code
- ❖ If specifications change or the validation process reveals problems, it becomes trivial to find the section of code responsible for a particular report section.

Table Architecture

- ❖ Once the [projmeta.projman] table is propagated with project-level metadata (from the Table Shells), one can easily write a short code segment that creates an actual output object (*demog.rtf*) without a single byte of data in the clinical database
- ❖ The table will be unpopulated with any results, but a SAS-created version of the shell can be created very early on to QC against the statistician's mock-up version
- ❖ This applies only to well-defined table types as above, not purely data driven tables like Adverse Events or Medical History.

Consider the typical workflow in a Table program:

- ❖ Extract from data source (raw, SDTM, ADaM as appropriate) the desired data and subset for analysis
- ❖ Call appropriate SAS procedures to summarize or derive required output (e.g. Transpose, Summary, GLM, Lifetest, etc.)
- ❖ Combine with various text labels and user-define formats to produce report-ready SAS dataset (in the work library)
- ❖ Combination of macro calls and Proc Report to create publication ready final output (RTF, PDF...)

In our hypothetically enormous complex program, one might assign

- ❖ Coder 1 to code for Panels 1-3
- ❖ Coder 2 for 4-10
- ❖ Coder 3 11-27

In this case one can produce four separate codes; #1, 2, and 3 are independent from one another

Advantage # 1

In this case one can produce four separate codes; #1, 2, and 3 are independent from one another, following the work flow

- ❖ Extract from data source (raw, SDTM, ADaM as appropriate) the desired data and subset for analysis for your panels
- ❖ Call appropriate SAS procedures to summarize or derive required output (e.g. Transpose, Summary, GLM, Lifetest, etc.)
- ❖ Combine with various text labels and user-define formats to produce report-ready SAS dataset (in the work library)
- ❖ Store resulting “publication ready” series of results into a permanent SAS data set within the project library. Codes # 1, 2, 3 will produce no publishable output on their own.
- ❖ Codes section # 4 simply takes the data file of results as produced by executing section (1, 2, 3) and writes out the final (large) publishable report object

Publishing Principles

- ❖ Create a ready-to-use RTF output file (no manual post-processing required)
- ❖ Use ODS (Output Delivery System) and Proc Report to create the file (standard macro calls to automate)
- ❖ Within the RTF file, primary programmer is responsible for pagination, fully specifying all attendant column headers, labels etc. Titles and footnotes are automatically supplied by extracting from `[projmeta.projman]` based on program name supplied by `[%getPgmName()]`
- ❖ Since SAS output is directed out to the RTF destination, the usual SAS distinction of numeric vs. text variables becomes moot. All output is stored in text fields, even if the content of most cells consists only of numeric values

Publishing Principles

- ❖ Where possible, allow Word to perform formatting functions; a combination of SAS styles, style overrides, and judicious use of RTF in-line codes can easily accomplish myriad tasks
- ❖ Aside from the structural fields [PPR], column names are generic: e.g., the leftmost column (usually containing explanation text) is by convention COL0, while subsequent dose-specific columns are merely COL1, COL2... as specified by the Table Shells template.

Publishing Empty Shell

Before the original source data extraction from the appropriate, build a “scaffold” or mold to hold the data.

```
data final(drop = rows);  
  format pagenum panel row 4.;  
  length col0 $ 200  
         col1 - col3 $ 20;  
retain pagenum 1;  
  if      (panel eq 4) then rows = 7;  
  else if (panel eq 5) then rows = 4;  
  else      rows = 3;  
  do panel = 1 to 5;  
    do row = 1 to rows;  
      call missing(of col:);  
      output;  
    end;  
  end;  
run;
```

Building Table Results

- ❖ The code above builds a SAS data set ready to hold the contents as subsequent code section build the report
- ❖ Note that fields [pagenum panel row] are now keys to the results
- ❖ Note that within a study, most reports will have the same generic structure with respect to number and labelling of column headers. After each separate section has completed a panel of results, add them to the growing results by very simple code:

```
data final;  
  update  
    final  
    panel1;  
  by pagenum panel row;  
run;
```



Publishing Table Results

At the very end of the Table code, now that the [work.final] dataset is fully populated, write to output object:

```
%rpt() /* standard macro uses output from      */  
      /* %getPgmName() to derive output file*/  
proc report data = final nowindows...;  
  cols (pagenum panel row col0 col1 col2 col3);  
  define ...; /* Labelling and style directives */  
run;  
%rpt() /* Automatically closes output object */  
      /* and restores LISTING destination    */
```

- ❖ Lastly, consider how building tables with meta-structure fields [PPR] can aid and indeed transform the least automated, most human-intensive part of our overall workflow: Validation
- ❖ Since our industry is so strictly regulated, it is insufficient to merely perform perfect work; we must also provide evidence that we have validated our work
- ❖ Unfortunately this nearly doubles our workload; and still does not guarantee “perfect” results (both Primary and QC programmers are quite capable of programming different codes to the same misunderstood specifications)

Typically, this workflow looks something like:

Validation (Common)

- ❖ First (Prime) coder writes code to produce a particular output object (e.g. demog.rtf). This SAS code completes with all results in the SAS work library which is naturally discarded upon program completion. The only “bread crumbs” left behind are the SASLOG, LISTING (perhaps), and whatever output object(s) are created outside the SAS system
- ❖ Secondary (QC) coder writes independent code, based on the same specifications (SAP, Tables Shells). The coder creates the same results in parallel, but instead relies on a simple Proc Print to the LISTING destination
- ❖ QC coder now compares the results from the QC code output to the nicely formatted results from the Prime program execution. This is often accomplished using what the U.S. Military refers to as “Mark-I eyeball”, meaning by pure (simple but tedious) physical inspection.

Validation (Common)

There are many ordinary circumstances where this workflow makes perfect sense and is as efficient as possible:

- ❖ The P-QC coding takes place on an already locked database
- ❖ Any discrepancies between Prime and QC output can usually be quickly tracked to unexpected data, insufficiently explicit specifications, or simple programmer error (on either side)

However, consider the workflow for a large Phase IIB or lengthy, Phase III registration trial planned to form the foundation of a Regulatory Submission for approval:

- ❖ Potentially long patient enrollment times and/or follow-up times
- ❖ Specifications (SAP, Table Shells) available for extended periods, although subject to change before Database Lock
- ❖ New and newly corrected data arriving throughout the trial.

Validation (Issues)

Under these circumstances, the above workflow shows its weakness: Every time a single data point changes:

- ❖ Re-execute Prime program (trivial)
- ❖ Re-execute QC program (trivial)
- ❖ Re-apply “Mark-I eyeballs” to manually compare Prime to QC results (decidedly **not** trivial).

Validation (Issues)

- ❖ As the QC process is often left to more junior staff, perhaps this inefficiency is considered acceptable
- ❖ Aside from the possibly abysmal effect on the morale of the QC coder, it is also extremely easy for the human eye to overlook real discrepancies while perusing potentially thousands of pages of output
- ❖ The SAS user community has produced some clever and innovative solutions over the years, resulting in many interesting papers
- ❖ The solution described here takes a slightly different approach.

Validation (Variation)

- ❖ In studies where automated QC process is practical, the study includes an extra libref, [QcData]
- ❖ Near the end of each analysis program, the Prime coder adds a simple data step, at the point in the flow where all results have been computed and are ready to be sent to a report object:

```
data qcdata.&pgmname;  
  set final;  
run;
```

Validation (Variation)

- ❖ Here `&pgmname` is a global macro symbol containing the name of the code being executed
- ❖ Note that `[qcddata.&pgmname]` has the generic structure `[pagenum panel row col0 col1... colN]`
- ❖ If examined by a SAS table viewer (e.g. ViewTable or FSView), the visual results would be identical to the nicely formatted publishable version stored in the RTF output file.

Validation (Variation)

The QC coder now begins the QC process by creating a temporary copy of the metadata

```
data qc;  
  if 0 then set qcddata.&pgmname;  
  /* more code to build the scaffold */  
  /* for the desired table structure */  
run;
```

/* OR */

```
proc sql;  
  create table qc  
    like qcddata.&pgmname;  
quit;
```



Validation (Variation)

- ❖ The QC coder then proceeds to program the rest of the table according to the same specifications as used by the Prime coder
- ❖ As each panel is derived and ready for comparison, it is added to the [work.qc] table until the process is complete
- ❖ At this point, the QC coder does not [Proc Print] the results table, but rather calls an enterprise level comparison macro, which performs a [Proc Compare] of the temporary QC table to the permanent QC DATA table, using [pagenum panel row] as keys



Validation (Variation)

- ❖ The macro extracts from `[Proc Compare]` the number of differences encountered (hopefully zero!) and prints out a discrepancy report
- ❖ Furthermore, it uses the program name to update `[projmeta.projman]` with the results of the comparison and the QC code execution date
- ❖ This allows the Project Lead to scan `[projmeta.projman]` to determine which Tables have been through the QC process, how many differences were detected, and when.



Validation (Variation)

- ❖ This is a crude measurement, as a difference of one can be much more complex and troubling a problem than 1,000 differences detected
- ❖ However, as the project work progresses, more Tables are completed and more QC checks result in success (e.g. zero differences), the Project Lead can track progress and identify tables that might require more attention or resources
- ❖ Other project utilities highlight
 - ❖ which entries in `[projmeta.projman]` have no code started
 - ❖ the age of the various report objects.

Next Steps

- ❖ Everything described in this paper has been tested and proved in the crucible of experience, including the Sponsor's and study teams first Application Filing
- ❖ Following the spirit of *kaizen*, it is a work in progress with enormous room for growth and improvement
- ❖ The strategy of having in-the-trenches coders create and polish their own tools to optimize the workflow means:
 - ❖ that progress is slow and sometimes fitful (study timelines take precedence!)
 - ❖ but it also ensures that only useful and helpful infrastructure is developed, tested, and implemented.

- ❖ The work life of the modern Clinical Statistical Programmer is quite complex
 - ❖ knowledge of pharmaceutical and regulatory requirements
 - ❖ constantly evolving and expanding CDISC standards, constantly improving software from SAS Institute
 - ❖ and the myriad details and responsibilities of supporting the clinical trial and safeguarding human health is daunting
- ❖ Given this environment, it makes sense to build as supportive and automated work environment as possible.

Imai, Masaaki (1986). *Kaizen: The Key to Japan's Competitive Success*. New York: Random House.

Hamilton, Paul. “ODS to RTF: Tips and Tricks”.
<http://www2.sas.com/proceedings/sugi28/024-28.pdf>

Rohini Rao, Omeros Corporation. “Gilding the Lily: Boutique Programming in TAGSETS.RTF”.
<http://www.pharmasug.org/proceedings/2012/DG/PharmaSUG-2012-DG11.pdf>