Paper DM09

Developing a Control Methodology for Customized Data Management and Processing

Mark Tabladillo, Ph.D., Atlanta, GA

ABSTRACT

This presentation will focus on the control methodology developed to manage, validate, and process survey data. Extensive SAS/AF® development experience is not required, though it will help to have had exposure to the basics of SAS/AF and SCL.

INTRODUCTION

To assist states and countries in developing and maintaining their comprehensive tobacco prevention and control programs, the Centers for Disease Control (CDC) developed the Youth Tobacco Surveillance System (YTSS). The YTSS includes two independent surveys, one for countries and one for American states. A SAS/AF application was developed to manage and process these surveys. During a four year period, over 600,000 surveys have been processed for 35 states and 100 international sites (from 60 countries).

This presentation will focus on the control methodology developed to manage, validate, and process survey data. Primarily, beyond the survey data, there are 12 customized SAS datasets which provide customized process management and validation for each state and region. The SAS/AF application supports the datasets and helps manage data flow by using several Windows-based procedures and one graphical control. The presentation will describe the control methodology conceptually developed, and generalize lessons learned for planning and implementing data management through customized datasets.

THE CHALLENGE

In 1998, the Office on Smoking and Health (OSH) first administered the YTSS in three states. This initial group participated in the Youth Tobacco Survey (YTS), built on identical sampling methodology to the Youth Risk Behavior Surveillance System (YRBSS). In 1999, OSH also launched the Global Youth Tobacco Survey (GYTS), sponsored by the World Health Organization (WHO). Subsequently, the Global School Personnel Surveys (GSPS) was added.

Three U.S. states participated in 1998, 10 states participated in 1999, and about 30 states participated in the 2000 school year. OSH provided customized survey support and analysis for each location (state or country). Each location administers a different survey, requiring creating a customized two-stage cluster sampling design, data collection strategy, and analysis. The survey questions are typically customized, with many states and countries choosing to add additional questions to the standard core questionnaire or change the order of questions. Additionally, the states or countries are typically divided into regions, requiring a separate set of reports for each region, and then combined data analysis for the entire state. Many states choose to survey both middle school and high school, so a state with 5 regions would require 5 reports for middle and high school (each), and total summary report, for a total of 12 processes.

Again, the survey methodology for the new YTS and GYTS was identical to what had been developed over a 10 year period for the YRBSS. Originally, that surveillance system had an analysis written in SAS/AF for the mainframe (which was the statistical computer of choice at the time). By 1999, SAS was at version 6.12, and the YRBSS team had been (and still is) undergoing a massive redevelopment to move their mainframe application to the desktop, in addition to adding some Visual Basic, SQL Server, and SAS/Intrnet functionality. Since the YRBSS software was in flux, it was not possible to simply copy the same code since the logic resided in several languages. Additionally, it was not necessary to immediately add all the functionality which the YRBSS project would gain from their multilingual approach.

With so much new development happening, the lead developer for the YRBSS SAS/AF application simply did not have the time to devote to the OSH project, though he did strongly recommend SAS/AF as the best solution. The current OSH team already had three intermediate SAS programmers and one advanced SAS developer working full-time on both doing the survey design and also the survey analysis. The advanced developer had started modifying the YRBSS SAS/AF Application, but was not done yet.

With the application development in progress, the OSH team had decided to start fresh with base SAS and SAS Macro Language. However, they were already at maximum working capacity with just 10 states to process. Each region required between 2,500 and 3,000 lines of base SAS and SAS Macro code which had to be customized and/or modified each time a new region was run. The opportunity for error was large, and the team was already overwhelmed with the current workload. Though the 1999 schedule was feasible, all the SAS programmers agreed that unless they developed an application, they would be unable to meet the promised demand for 2000. The advanced SAS developer had therefore started modifying the YRBSS SAS/AF application, but did not finish the project before he left CDC.

FINDING A STARTING POINT

At the beginning, there were three possible starting points: 1) attempt to make the first developer's modified SAS/AF application work, 2) start with the in-flux YRBSS SAS/AF application, and make that system work, or 3) start with the base SAS and SAS Macro Language code.

Choice	Advantages	Disadvantages
First SAS/AF Application	Code started to be tailored	Code did not work Structure inherited No Classes
Similar YRBSS SAS/AF Application	Code Works	No customization performed Base structure still reflected mainframe heritage No Classes
New Base SAS Code	Code Works, Customization Built-In	Not SAS/AF No Classes

The time schedule given to have an application running was nine months. The first four weeks were spent understanding the current base SAS code, and the two applications presented as possible candidates to change.

At the end of this initial analysis, the starting point chosen was the new base SAS code. The advantage of having code that completely works is good, because in a complex process (even with several thousand lines of code) it is known that the code performs and outputs what is expected. Granted, this code did not have all the functionality desired, but it did provide a complete baseline example that worked.

Additionally, choosing to modify the existing code most closely aligns the intermediate programmers (who remained with the project) to the new software application. It was apparent early that these programmers had several solid ideas of how to proceed on the project, and specifically had strong concerns about the growing burden of data management and number of

files. The first concern was about identity, and the second one was about sheer volume.

Though the creators of the two applications were not working directly with the project, their work was also important because they provided a conceptual direction of how to proceed. Also, their work provided an understanding of what the OSH team expected in terms of a final output. So, it was important to study both the software methodology (internals) and also the look and feel (externals) so that the final result could best meet expectations.

THE ONE-TIME RULE

Immediately, one of the areas of concern which could be solved with SCL came to be known as the "one-time rule". This rule states that unique information needs to only be introduced (or defined) once to the application. This basic rule is an important feature of any well-designed data management project.

Conceptually, implementing the one-time rule involves collecting similar data elements together and deciding how to store them: in datasets, in SCL lists, as variable definitions, or perhaps (as later implemented) inside classes.

In the previous base SAS code, SAS macros were used to automate many code portions. Starting with a baseline code, the macros were manually changed, and the resulting program was saved. However, the inputs to the macros were often identical, and having to hardcode these identical changes several times provided opportunities for error.

The One-Time Rule drove the basic structure of the application. . First, when possible, data were either hard coded (as enumerated SCL lists or variables), or put into standardized datasets (not intended to be modified). If all the data could be stored this way, the overall process would literally be a push of a single button.

However, because each survey is different, certain elements need to be modifiable, either at the dataset or SCL variable level. Thus, the application needed to allow for two ways for the data analyst to tell the application about region-specific information. One was through the FRAME input components (such as list boxes and text boxes), which then could be stored in a dataset or text file. The second way, for larger sets of information, was through a modifiable matrix (SCL List) or dataset.

Conceptually, the following table illustrates these four categories of inputting data.

	Short Information	**Long Information**
Non-modifiable	Enumerated Constants	Standardized SAS Datasets
Modifiable	FRAME Variables	Customized SAS Datasets

The next four sections will discuss each of these four categories.

ENUMERATED CONSTANTS

Many languages have built-in constants with a standard symbolic representation. In this paper, the word “constant” is defined as a fact which is not intended to be changeable by the user (but possibly could be changed by the code). By this definition, for example, “constant” can refer to the standardized SAS Macro variables which store information about the computer, even though the same variable may return a different value for another user and computer.

The term “enumerated” generally refers to counting, and as applied to application development, it refers to specifically listing out and declaring variables (with a DCL or DECLARE statement) as being constants. Base SAS often allows variables to be introduced without explicit declaration, but the declaration feature leads to better documentation because it requires explicit variable typing (numeric or character, for example) and defined variable sizes. SAS classes even allow more declaration options, and include, for example, adding a description and placing sets of variables into categories.

In this application, an example of the enumerated constants is the SCL list which holds the possible combination of survey types and years. Though this list will change yearly, it is typically not necessary for the application user to be concerned with what the whole list is, and therefore it is the type of information which can be stored in an SCL list.

STANDARDIZED SAS DATASETS

Following the previous example, if the application were providing support for 6 types of surveys over a 25 year period, perhaps a better way to store that information would be in a SAS dataset. There is not generally a definitive line between what should be stored in a dataset and what should be hard coded, and the tradeoff has been historically determined by memory usage, processing speed, and hard disk space.

However, as all these factors become more available, the new criteria will lean more toward programmer preference, and specifically whether a programmer would rather see several screens full of information or instead put the information into a dataset.

For this project, there was no need to share these datasets with other applications or uses, and therefore the SAS format was appropriate. However, in a general case, one could always code this standardized information to come from another format (and convert it with proc access, for example), or possibly on another platform altogether (and use, for example, SAS/CONNECT®).

For this application, one of the standardized dataset contains all the US states and territories along with their two-letter abbreviated codes. Either the unique full name or unique abbreviation are used in all the report headers, report names, directories, and file names (whether text or SAS datasets).

Overall, being able to classify information as non-modifiable means will lead to a more parsimonious user interface and reduce the possible number of errors from user input.

FRAME VARIABLES

The standard SAS/AF literature contains many examples of the rationale and methods of how to capture variables from a frame and integrate them into an application (see SAS Institute, 2000a, 2000b, 2000c). For this survey application, the screen variables all had default values and defined valid ranges (a character variable can have a range too, by limiting its size). The user (analyst) could then either accept the default or modify the screen variables based on the specific survey requirements.

To store these variables, a survey specific dataset was created, from which initial values were displayed on a screen, and to which the final values would be stored. Thus, the YTS 1999 would have one dataset and the individual rows represented regions within that combination of survey and year.

Over time, the application has been rewritten to allow, as much as possible, for the software to set or calculate variables. Again, if there is a way to derive or figure out a value, that function is put behind the scenes, and not made the variable available to the user. This technique follows the "One-Time Rule" because it allows for derivative variables or information to be automated instead of presented. Making the user interface as simple as

possible streamlines complexity, reduces the possibility of error, and simplifies the process of debugging when something goes wrong.

The SAS/AF literature contains many examples of how to implement and store screen variables, and thus, this method is not a focus for this presentation.

CUSTOMIZED SAS DATASETS

Larger sets of information are better stored as a matrix or dataset, and for this application, different regions would each have a customizable set of datasets. Conceptually, Shalloway and Trott (2002) consider this type of storage to be a form of encapsulation, and term this type of customized storage as the “Analysis Matrix” tool.

A starting collection of "master datasets" (standardized for each survey and year combination) would be used as templates for each region, and then each specific region could possibly have its own customized copy of these master datasets.

An example is the questionnaire layout, which is different for each combination of survey type and year. In this case, a “core” questionnaire represents the starting point, from which each country or state might make customized changes based on the core. Thus, many states will have different questionnaires based on the YTS 2000 core questionnaire.

SCL can then be programmed to read these datasets, and then substitute the information into the code several times. SCL can even open several datasets simultaneously, and bring in information from several datasets into what would be submitted as a customized data step.

Since each region could possibly have its own questionnaire, that layout information is used at several times. At one point, it is used to generate an INPUT statement (within a data step) to read raw data from the original file. At another point, the layout file is used to generate a dataset which is essentially the questionnaire in printable format. A third use is using the layout file to generate report titles and formats for PROC TABULATE labels. These multiple uses were reflected in the original base SAS code, where identical information was either entered as macro content several times, or outright hard coded (such as the original INPUT statements).

FROM CONTROL DATASETS TO SCL

The application uses the control datasets (either non-modifiable or modifiable) to generate both SCL and SAS code. The term “control” refers to the ability to affect or create either SCL or SAS code.

For SCL, the application reads information from the dataset into the SCL workspace. From that point, the code can be directly used in SCL, or optionally, submitted as base SAS code (which is required for many types of commands, like the data step or most procedures, both of which unfortunately have no direct SCL equivalent).

Creating customized base SAS code from SCL involves sending portions of the text to the SUBMIT buffer, and then submitting the final product (using the SUBMIT CONTINUE command). The technique is similar to SAS Macro, which is not necessary to implement this technique. What is similar is that SCL variables are substituted inside SUBMIT blocks by using a single ampersand before the name of the variable. Unlike SAS Macro variables, SCL variables are not all character variables.

Instead of providing an actual example, the following table includes the specific SCL commands repeatedly used to submit base SAS code.

Command	Use
SUBMIT	Allows sending only part of a command to the buffer, even allowing you to send half a line without the semicolon
ENDSUBMIT	Marks the end of a block of text sent to the buffer
OPEN	Opens the dataset
ATTRN, ATTRC	Obtains information about the dataset, specifically NLOBS, the number of non-deleted observations
VARNUM	Determines the variable number (order) given the variable name
FETCHOBS	Obtains one observation from the dataset
GETVARN	Obtains the numeric value
GETVARC	Obtains the character value
CLOSE	Closes the dataset (allow the dataset lock to be released and allowing the LIBNAME to be cleanly reset)
SUBMIT CONTINUE	Releases all the code in the buffer for execution

Typically, the information was read directly from the dataset and sent straight to the buffer. However, sometimes it was good to save the information in an SCL list for later use, manipulation, or execution. A cost of SCL lists is in memory, and a way to measure performance is to open an application like System Monitor (which is for Windows 95) to see what the difference is. Another possible drawback is that adding an SCL list increases complexity, possibly making code harder to maintain or modify in the future (since an SCL list will necessarily couple two sections of code together).

MONITORING CONTROL DATASETS

Since the control datasets were used to send information straight to SAS execution, there were many opportunities for the application to crash.

The original six base SAS programs were consolidated into five processes. There are five (originally six) processes because each step represents a stopping point where some amount of checking needs to be done to insure that the final output is as expected. Those types of checks include reports intentionally created to look for anomalies in the survey data, and for conditions under which the original sample design might be violated. The analysts also check that the expected number of surveys had indeed been scanned in (typically the survey is administered on bubble scantron-type forms) and match to the original school roster.

Internally, each of the five processes first checks that the datasets exist and can be exclusively opened, as tested by the SCL OPEN function. If the dataset does not exist, either the program will copy the standardized master copy, or in some cases, there is no standardized dataset, and the program will terminate normally. However, the typical situation will be that the dataset does exist and is available for exclusive use. This locking only affects the regional level datasets, and therefore two separate analysts could be working on different regions of the same survey without encountering a locking error.

The second level of checks involves the existence of certain variables within each dataset, as tested by a nonzero return code from the VARNUM command. Each type of dataset has to have standardized names for this type of check to work. The layout file, for example, looks for the variable "QUESTION" to be the character field which has the questionnaire text. Each of these control files was assigned standardized variable names and

expected variable types (numeric or character). If any expected variable is not present, the program terminates normally.

Over time, variables were sometimes added to these control datasets in two categories, either required or optional. An optional variable, if present, will trigger perhaps a block of SCL or base SAS code. One example in the layout file is the table numbers. The original code automatically created table numbers (for the TITLE statement) which started at one and increased a counter by one. However, sometimes cross-regional comparisons are more easily done when standardized table numbers represent specific variables, and those numbers may not reflect the file's (or sorted file's) variable order. Thus, the code now looks for a optional "TABLENUM" variable, which is not required to execute the code, but when present will be used instead of the standardized counter.

CONTROL DATASET INTEGRITY

Another aspect of control is control dataset integrity, referring to having valid values inside the fields.

In some cases the application applies checks when the data is input into SAS format. A datasets tab was created for importing and exporting control datasets into various formats. That tab also has a SAS data table component, which can be used to modify the dataset. However, our experience has been that regular Microsoft Excel is generally better and less prone to crash (since touching some spots around a legacy data table may crash SAS).

Excel is still best specifically because it is not an inherent database software, and therefore does not have the size and type (character or numeric) specifications (which become restrictions) that Microsoft Access or SAS would have. The Excel interface allows for easily reordering variables, renaming columns, or especially resizing character fields.

Both during the five processes and the separate data input process (converting from Excel to SAS), some datasets are checked for integrity, and specifically that certain values are valid. For example, a fields may be checked for valid code tags. Another example is that a code fragment variable is checked to see that the parentheses are balanced (named “code fragment” because the results are submitted behind an IF statement in base SAS).

During the initial design phase it's possible and prudent to build in many checks. However, exceptions and anomalies continue to arise, specifically because the field professionals have the opportunity to modify aspects of the survey questionnaire and sampling design. Their expected changes represent a process interface, and have sometimes modified the control dataset integrity criteria, and sometimes the dataset design.

MONITORING PROCESS RESULTS

To recap, there are a total of five processes which each had represented a separate base SAS program. With customized code being generated for each region, the application saves the process logs (not the original code, just the log) so that the runtime feedback could be studied. Additionally, the application saves the output screen, though sometimes it sends portions of the output to separate files. Large data management projects should require saving at least the log and perhaps all the outputs too.

The names of the output and log files were chosen to simply have the date (instead of the date and time). That way, during a specific day, multiple runs of the same process (which are common) will simply overwrite the old files. Commonly, an analyst will run a specific process repeatedly on the same day, each time applying different screen or dataset inputs.

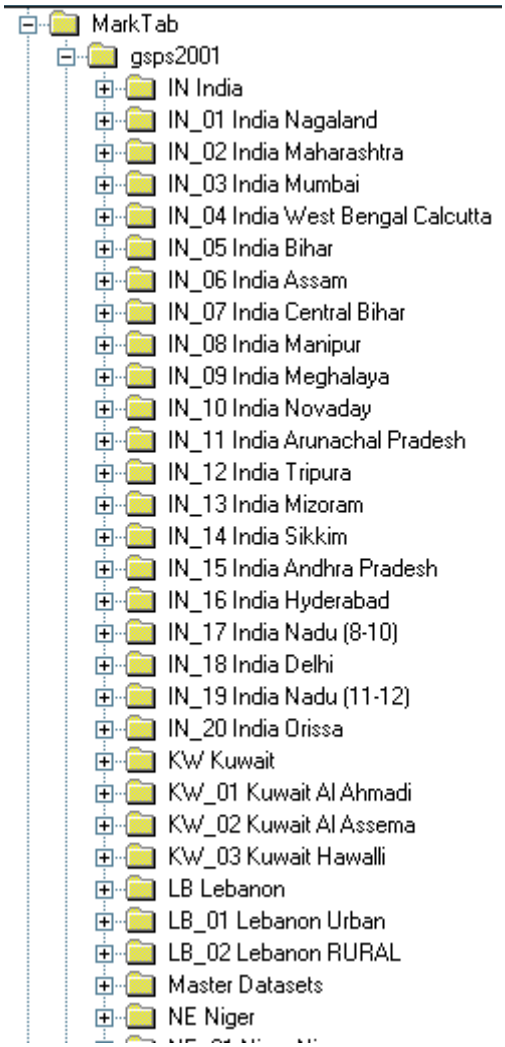
A recent addition has been a process log dataset, which keeps some of the screen variables, as well as an overall process time. Some processes take typically 15 seconds for 4,000 observations, and that same dataset could take 2.5 minutes in another process. Recently, CDC has moved the SAS 8.2 application to a Citrix server. As implemented, this server typically takes more processing time than running SAS through Novell alone.

Each of the five processes typically takes a few minutes to execute on desktop machine. However, some time is required to gather the data for the interactive FRAME and the customized control datasets. Also, there are always time gaps between each process during which the analysts will examine the integrity of the survey results, and often discover errors in the sample design datasets or missing information. Though the output screen will commonly point out flaws, sometimes the survey omissions are indicated only in the log. The intermediate reports have been designed to, as much as possible, show discrepancies on the output screen as opposed to making the user dig through an often complex log.

HIERARCHICAL DATA STRUCTURE

Perhaps one of the most helpful techniques is to use the survey/year dataset to also generate Windows subdirectory structures for managing files.

The following screen capture shows a partial directory structure for the GSPS 2001 survey. Under the "gsps2001" subdirectory, there are a total of 250 folders and 1,905 files, representing 575 MB of information.



A SAS dataset (called "Master") in the "Master Datasets" subfolder includes all the region-specific information, including the numbers and names of the specific regions, and other information used to populate the FRAME variables.

Storing this information hierarchically allows the analysts to interactively use Windows Explorer with SAS open, and perhaps move or zip certain files. Originally, there were no country or region names on the folders, but the dynamic addition of these fields (which are maintained by the "Master"), allows the analyst to have folder names which are consistent with the names stored inside the datasets and on the final reports.

Within each regional subdirectory, five standardized subdirectories store the region-specific files: 1) "Final" has the final output results, 2) "Input" has the input text files, 3) "Log" stores program logs, 4) "Output" stores program output, and 5) "SASData" stores the SAS survey data and SAS control datasets (and perhaps the control data in another format like Microsoft Excel). Each file is named according to a standardized naming convention, which necessarily requires information from the control datasets, and sometimes the date is included.

WORKING WITH MICROSOFT WINDOWS

SAS version 8 already includes many of the commands which would be used to operate Microsoft Windows. However, there were some cases where the program needed functionality beyond the available set. Extending the reach into Microsoft Windows shows that a control dataset can reach beyond the SCL and base SAS environment and into the operating system.

Fortunately, there is an advanced SAS technique available which can be used to call the Windows 32 API, and theoretically access any function available in Windows. This technique is mentioned here, for example, because it was used to move files (same as a "Rename" using Windows Explorer), and was specifically used to apply region names to the directories.

The technique requires accessing the SASCBTBL attribute table. For example, the following table provides the code for moving a file. The SASCBTBL was stored inside the catalog inside a source program element, then moved to a text file using the SCL PREVIEW function. More information about SASCBTBL is available in the *SAS Companion for the Microsoft Windows Environment, Version 8*, under the section "Accessing External DLLs from the SAS System". General information about the Windows 32 API is in Appleman (1999).

```
ROUTINE MoveFileA
        MINARG=2
        MAXARG=2
        STACKPOP=CALLED
        MODULE=KERNEL32
        RETURNS=LONG;
        ARG 1 INPUT CHAR FORMAT=$CSTR400.;
        ARG 2 INPUT CHAR FORMAT=$CSTR400.;
```

Also, there was a simple Visual Basic control included in the SAS/AF Frame which provides the ability to both move and name the input datasets into the standardized form that runs the application. This object was saved from Visual Basic as an OCX visual control, and then included in the frame using the insert OLE object.

Internally, the OLE object is set as a drag site, with several SAS text boxes being the drop site. Thus, the control works by visually presenting the Windows directory, and then being able to simply drag a file from that visual directory into a SAS text box, the result being the full original name of that file. This ability prevents file naming errors by presenting the analyst with an intuitive Windows Explorer interface. Alternatively, the SAS text boxes still work by just typing in a specific name (or perhaps cutting and pasting a name from another location, perhaps email). A command button will then copy the file from its current location to the expected location with the expected standardized naming applied.

The following table provides the code for this relatively simple Visual Basic control:

```
Option Explicit
' Path needs to be explicitly defined because of an internal
truncation error
Dim Path As String

Private Sub UserControl_Initialize()
  Drive1.Drive = "e:\"
  Dir1.Path = "e:\osh"
  File1.Path = "e:\osh"
End Sub

Private Sub Drive1_Change()
  Dir1.Path = Drive1.Drive   ' Set directory path.
End Sub

Private Sub Dir1_Change()
  Path = vbNullString
  Path = Dir1.Path    ' Set file path.
  File1.Path = Path   ' Set file path.
  Label1(0).Caption = Path
End Sub

Private Sub File1_OLEStartDrag(Data As DataObject,
AllowedEffects As Long)
   Dim i As Integer
   Data.SetData , vbCFFiles
   Data.SetData Data.Files(1), vbCFText
End Sub
```

GENERALIZED LESSONS

1) ***Understand the user objectives in terms of both functionality and also in terms of deliverable***. In this case, because SAS is such a wonderfully featured product, the functionality could all be provided in some way. The final product, too, was largely compared to previous SAS/AF work.
2) ***Start with what works***. When given the chance to use various projects or products, it's always good to start with working code, all the while, not ignoring what other helpful techniques or approaches can come from other sources.
3) ***Continually think through how to apply the "one-time rule".*** Many of the gains made in eliminating duplication came even before a class structure was added, and started with clustering related data elements together.
4) ***Standardize all data possible***. Creating standardized datasets, hard-coded SCL lists, and enumerated variables removes variation from the application user. Even variables or information which can be derived should also be moved into the application and away from the interface. Users prefer having information hidden.
5) ***Allow users to modify brief information on screen, and complex information within datasets***. Screens are good for smaller sets of information, and datasets (particularly using a spreadsheet interface) allows a better way to customize the "Analysis Matrix".
6) ***Expect the application to monitor data integrity as well as processing results***. Having standardized criteria for screens, and more importantly, for customized datasets allows for greatly reducing processing errors. As well, overall performance metrics provide a good sense of how the combination of local and network hardware are performing together.
7) ***Leverage operating system functions through SAS***. In this case, Windows was the platform of choice, but the same techniques can be used to develop operating-specific calls on other platforms too. While some platforms do not inherently exist on GUI interfaces, all of them have ways to use operating system level commands to manage files.

CONCLUSION

Managing a large file structure is very doable with SAS/AF, but requires forethought and planning. While a standard SAS/AF project is complex enough, a highly customized application, such as the one presented, presents unique challenges which can be best handled with not only the standard SAS/AF interaction, but also the intentional application of the "Analysis Matrix". The application is hard coded to control the dataset integrity and validity of these control datasets, which are primarily used to generate both customized SCL and base SAS code. Beyond SAS, the control datasets also manage the storage and output of data on the network, and can be used to run operating system commands. While Microsoft Windows was specifically illustrated, the "Analysis Matrix" could clearly be applied to control other operating systems.

Further information on the class structure and development for this application is presented in Tabladillo (2002).

REFERENCES

Appleman, D. (1999), *Dan Appleman's Visual Basic Programmer's Guide to the Win32 API*, Indianapolis, IN: Sams Publishing, Inc.

SAS Institute Inc. (2000a), *SAS/AF Software: Application Development I Course Notes*, Cary, NC: SAS Institute, Inc.

SAS Institute Inc. (2000b), *SAS/AF Software: Application Development II Course Notes*, Cary, NC: SAS Institute, Inc.

SAS Institute Inc. (2000c), *SAS/AF Software Procedure Guide, Version 8*, Cary, NC: SAS Institute, Inc.

SAS Institute Inc. (2000d), *SAS Companion for the Microsoft Windows Environment, Version 8*, Cary, NC: SAS Institute, Inc.

Shalloway, A., and Trott, J. (2002), *Design Patterns Explained: a New Perspective on Object-Oriented Design*, Boston, MA: Addison-Wesley, Inc.

Tabladillo, M. (2002), " Refactoring design patterns into a SAS application ", *SESUG Proceedings 2002*.

ACKNOWLEDGMENTS

Clifton Loo, Ph.D. provided important editing and feedback. Also, thanks to all the great public health professionals at the Office on Smoking and Health, Center for Chronic Disease.

CONTACT INFORMATION

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

Mark Tabladillo, Ph.D.
Email: marktab@marktab.com
Web: http://www.marktab.com/