

## Paper 28-28

**The One-Time Methodology: Encapsulating Application Data**

Mark Tabladillo, Ph.D., Atlanta, GA

**ABSTRACT**

This presentation will introduce a simple but powerful concept called the "One-Time Methodology" for encapsulating data in applications. Extensive SAS/AF® development experience is not required, though it will help to have had exposure to the basics of SAS/AF and SCL (SAS Component Language). The methodology can be applied in other application languages.

**INTRODUCTION**

This presentation will focus on the "One-Time Methodology" applied to manage, validate, and process survey data with a SAS/AF application. This methodology allows developers to plan an application's data sources based on two concepts – whether or not the user will modify the data, and whether or not there is a "large" amount of data to store.

Beyond the primary survey data, the presented SAS/AF application uses enumerated constants, frame variables, standardized and customized SAS datasets to hold data. The SAS/AF application supports information integrity and manages data flow by using standard SAS functions (presented by a frame), Windows native procedures, and one graphical control. The presentation will describe how to apply this simple but powerful methodology, and generalize lessons learned for planning and implementing application data management.

**BACKGROUND**

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 similar but independent surveys, one for countries and one for American states. A SAS/AF application was developed to manage and process these surveys. During a five year period, over 1,000,000 surveys have been processed for 35 states and 100 international sites (from 60 countries).

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 Survey (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 did not have ample time to devote to the OSH project, though he did 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.

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 METHODOLOGY”

Immediately, one of the areas of concern which could be solved with SCL came to be known as the One-Time Methodology. This nominal methodology states that unique information needs to only be introduced (or defined) once to the application.

Conceptually, implementing the One-Time Methodology involves collecting similar data elements together and deciding how to store them: in datasets, in SCL lists, as variable definitions, or inside objects.

In the original base SAS code, SAS macros were used to automate many code portions. Starting with a baseline code, the macros were manually customized each time a new survey was processed, and the resulting program was saved. However, the changes to the macros were often identical, and the need to keep changes identical provided many opportunities for errors.

The One-Time Methodology 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 routinely changed, 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, as well as discuss the relative terms “short” and “long”.

### 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 recompiling the application). 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 determining a count, and as applied to application development, it refers to specifically declaring variables (with a DCL or DECLARE statement) as constants. Base SAS often allows variables to be introduced without explicit declaration. However, declaration 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, placing sets of variables into categories, and defining classes within classes.

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 not information which needs to be modified by application users on a daily basis. Slowly changing or unchanging variables are prime candidates for enumerated constants. Enumerated information can be stored as declared character or numeric variables, within SCL lists, or within specific classes.

### STANDARDIZED SAS DATASETS

If the application were provided support for 6 types of surveys annually over a 25 year period, perhaps a better way to store that list of 150 combinations would be in a SAS dataset. There is not an absolute definitive line between what would be best stored in a dataset (“Long Information”) as opposed to being hard coded (“Short Information”). The developer will therefore make a tradeoff by assessing overall memory requirements, client and/or server processing speed, and hard disk space. For example, if abundant resources were available, the argument would lean toward all standardized data being enumerated within the application.

For this application, some standardized SAS datasets were used, and during the application’s processing, these datasets were typically partially (selected rows and/or columns) read into memory, if at all.

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

In 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). Additionally, there was a need to customize the list of US states to include Native American populations and also specific American territories. The state list is another example of slowly-changing information which does not need to be presented to users on a routine basis.

Overall, when a developer can classify information as non-modifiable, the resulting user interface will be simpler and reduce the possible errors during data entry.

### FRAME VARIABLES

The SAS/AF documentation 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.

The survey processing screen (for a specific region within a state) has about 15 variables. These variables all have standard initial values dependent on survey type. In addition to presenting these 15 variables on the screen, they are also stored in a survey-specific dataset. For example, the YTS 2002 survey would have one dataset and the individual rows represented regions within that combination of survey and year. For each region, there are about 15 variables which users might change on the screen, and each variable is stored within the dataset.

Over time, the application has been rewritten to allow, as much as possible, for the software to set or calculate variables on its own. 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. Derivative variables or information should 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.

The SAS/AF documentation contains examples of how to implement and store screen variables, as well as how to use Frame SCL to pass that information to and from SAS datasets.

## CUSTOMIZED SAS DATASETS

Larger sets of customized 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 name this type of customized storage as the "Analysis Matrix" tool (a specific design pattern).

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

For example, the questionnaire layout 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 2002 core questionnaire.

SCL can then read these datasets, and then substitute the information into submitted SAS code repeatedly. SCL can even open several datasets simultaneously, and merge information from several datasets which would produce a customized submit block.

For example, the questionnaire layout is central to survey processing, and therefore the application uses the layout repeatedly. At one point, the layout is used to generate an INPUT statement (within a data step) to read raw data from the original ASCII file. At another point, the layout is used to generate a dataset which is essentially the questionnaire in printable format. A third use is when the application uses 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).

## THE SCL GATEWAY

SCL is the common gateway for applying the "One-Time Methodology". Whether applied in classes or in Frame-related

code, the SCL language provides the power to access and manipulate data. The enumerated constants are implemented by defining variables, SCL lists, and objects with expected values. The Frame is used for interactive variables, and details on how to pass information from the Frame to the SCL environment as well as validate data integrity are in the SAS documentation.

The next sections will focus on moving data from datasets to the SCL environment, and outline: 1) how to access datasets, 2) how to check dataset integrity, 3) how to prepare to submit, and 4) how to monitor process submission.

## HOW TO ACCESS DATASETS

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

The application uses SCL to read information from the dataset into memory. From that point, the data can be directly used in SCL, or optionally, submitted with base SAS code. Code submission 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 preview buffer, and then releasing the preview buffer using the SUBMIT CONTINUE command. The technique is similar to running a SAS macro; however, SCL distinguishes between numeric and character substitution, while macro variables are all assumed to be a character type.

Instead of providing an example, the following table lists key SCL commands repeatedly used to read SAS datasets and submit customized blocks of code.

Command	Use
SUBMIT	Allows sending only part of a command to the preview buffer, even allowing you to send part of a line (without the semicolon)
ENDSUBMIT	Marks the end of a block of text sent to the preview 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 preview buffer for execution

Typically, the information was read (by a non-visual object) directly from the dataset and sent straight to the preview buffer. However, sometimes it was efficient to save the information in an SCL list for repeated use throughout the SCL code.

## HOW TO CHECK DATASET INTEGRITY

Another aspect of control is control dataset integrity, referring to having valid values inside the fields. Validity checks were all performed in SCL, as prerequisites to running processes, or in the Frame-related code.

In some cases the application applies checks when the data is input into SAS format from ASCII or Excel. A datasets tab (on the Frame) was created for importing and exporting control datasets. That tab also has a visual SAS data table component, which can be used to modify the dataset. However, our experience has been that Microsoft Excel is generally less prone to crash (since touching some spots around a legacy data table may crash SAS). Excel is also more useful 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 easily 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 field may be checked for valid codes. Another example is that a code fragment variable is checked to see that the parentheses are balanced (named "code fragment" because the information is 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 each survey questionnaire and analysis could potentially be different. These expected customizations represent different processes, and can sometimes modify the control dataset integrity criteria, and sometimes the dataset design.

## HOW TO PREPARE TO SUBMIT

Since the control datasets were used to send information straight to SAS execution, there are many opportunities for the application to crash. Given that the dataset integrity has already been addressed, there are some important checks to make before sending anything to a preview buffer.

The original six base SAS programs were consolidated into five processes (a "process" is defined here as SCL code combined with run-time submitted SAS code). There are five 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 checks include reports intentionally created to look for potential anomalies in the survey data, and for conditions under which the original sample design might be violated. For example, the users 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; this type of check needs to be done manually and not automated because there are numerous ways to verify whether the expected or scanned data numbers are wrong, including possibly recounting original scantron sheets.

Internally, each of the five processes has a similar SCL checking structure for possible early termination. First, the process checks that all accessed datasets exist and can be exclusively opened, as tested by the SCL OPEN function. If the dataset does not exist, sometimes the program will copy the standardized master copy; in other cases, there is no standardized dataset, and the program will terminate early with an error which indicates what datasets are either missing or unavailable. However, the expected situation will be that the dataset does exist and is available for exclusive use, and therefore can be locked. 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. Second, the program checks for 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. For example, the

questionnaire layout file needs to have the variable "QUESTION" as a character type. The program looks for assigned standardized variable names and expected variable types (numeric or character). If any expected variable is not present or is of the wrong type, the program terminates early with an error.

Over time, variables were sometimes added to these control datasets in two categories, either required or optional. New required variables cause early termination if not present. A new 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 an optional "TABLENUM" variable, which is not required to execute the code, but when present will be used instead of the standardized counter.

In summary, before any code is sent to the preview buffer, SCL checks the existence and availability of datasets, and the existence and correct variable types of variables.

## HOW TO MONITOR PROCESSES

To recap, there are 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 unique process 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, and we also have good process results on which network servers are more efficient.

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.

## 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 similar to previous SAS/AF work.

- 2) **Start with what works (encapsulate success).** When given the chance to use various projects or products, it's good to start with working code, all the while, not ignoring what other helpful techniques or approaches can come from other sources.
- 3) **As the application changes, continually apply the "one-time methodology".** 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) **The methodology can be used outside SAS/AF.**

## 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 strategic application of the "Analysis Matrix". The One-Time Methodology provides a way to model data encapsulation within an application, while the application language (in this case, SCL) is used as the gateway for monitoring data integrity and processing.

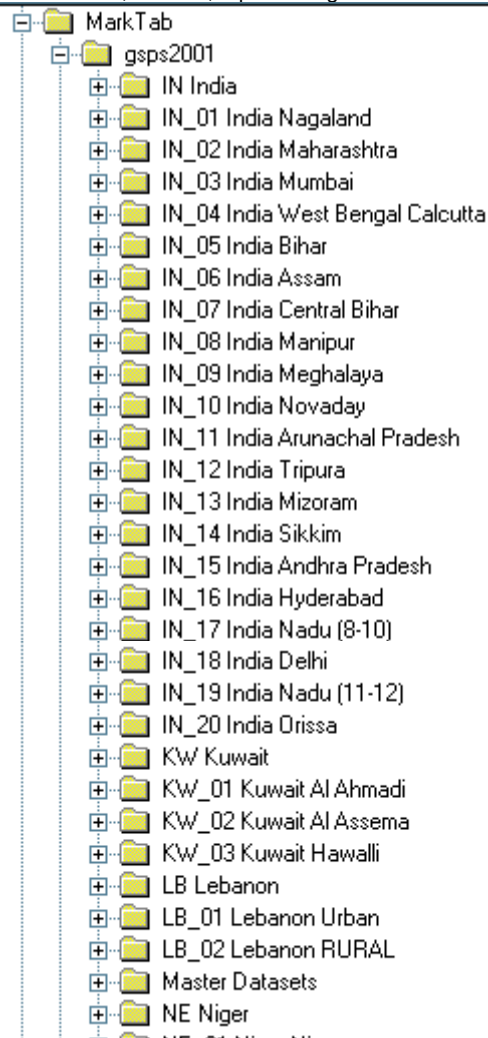
Further information on the class structure and development for this application is presented in Tabladillo (2003).

## 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. (2003), "Application Refactoring with Design Patterns", *SUGI Proceedings 2003*.

## APPENDIX A. HIERARCHICAL DATA STRUCTURE

This appendix presents an example of how to apply a customized dataset. 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 "gspgs2001" 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 survey regions, and other information used to populate the FRAME variables, as well as determine titles, file input/output names and LIBNAMEs. These uniquely identifying fields are also stored within the final cleaned survey dataset. Thus, the fields are defined one time and applied in many important areas.

Storing this information hierarchically allows the analysts to interactively use Windows Explorer (with perhaps the application open), and move or zip certain output files. The country and region names (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.

## APPENDIX B. WORKING WITH MICROSOFT WINDOWS

This appendix presents an example of how to extend SAS/AF beyond SCL to access standardized or customized datasets. 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.

SAS can 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.;
```

A simple Visual Basic control was developed for 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
```

Though this application uses Microsoft Windows, the point of this example is to illustrate that SAS can extend beyond SCL into the native operating system and issue system-level commands.

## 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.

## TRADEMARK CITATION

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 registered trademarks or trademarks of their respective companies.

## CONTACT INFORMATION

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

Mark Tabladillo  
 Email: [marktab@marktab.com](mailto:marktab@marktab.com)  
 Web: <http://www.marktab.com/>