Paper 028-29

# How to Implement the One-Time Methodology

Mark Tabladillo, Ph.D., markTab Consulting, Atlanta, GA


## ABSTRACT

This tutorial will demonstrate how to implement the "One-Time Methodology", a way 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 factors – whether or not the user will modify the data, and whether or not there is a "large" amount of data to store. This presentation will demonstrate an example of how to plan the data sources, and guidelines on how to integrate the resulting elements (enumerated constants, frame variables, standardized and customized SAS datasets) in a SAS/AF application.

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 1,000,000 surveys have been processed for 35 states and 100 international sites (from 60 countries).

The "One-Time Methodology" was conceptually developed 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.

This methodology framework has been previously presented (Tabladillo 2003a) along with its class structure (Tabladillo 2003b). The focus of this tutorial is to present specific coding techniques and tips on implementing this methodology.

## THE "ONE-TIME METHODOLOGY"

This 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.

The One-Time Methodology drove the basic structure of the YTSS 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 somewhat different, certain elements need to be modifiable. Thus, the YTSS 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 four categories of inputting data, and together this structure has been named the "One-Time Methodology".

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

The methodology is named "One-Time" because all information presented to any application falls into one and only one of these four categories. The developer chooses where to place the information, and the choice can be changed anytime in the future. The term "non-modifiable" is equivalent with build-time information, and the term "modifiable" is equivalent with "run-time" information.

The next four sections will define, discuss and illustrate how to implement of these four categories:

1. Enumerated Constants
2. Standardized SAS Datasets
3. Frame Variables
4. Customized SAS Datasets

## CREATING 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 at run-time (but could always be set during build-time). 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, "enumerated" means specifically listing out and declaring a specific piece of information to be a variable (with a DCL or DECLARE statement). That variable is then later set to a specific build-time value.

Base SAS often allows variables to be introduced without explicit declaration. However, declaration leads to more inherently explicit code 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.

## ENUMERATED CONSTANT EXAMPLES

This first example shows a declaration of variables used to describe the possible file transfer formats made available on the frame. The code appears in the frame SCL:

```
DCL
      char
         xferSAS8
         xferSAS6
         xferExcel2000
         xferExcel97
         xferExcel5
         xferAccess2000
         xferAccess97
         xferDBF
          xferEpiInfo2000;

* Enumerated Constants for File Formats;
xferSAS8    ='SAS Version 8';
xferSAS6    ='SAS Version 6';
xferExcel5  ='Excel 5';
xferExcel97 = 'Excel 97';
xferExcel2000  ='Excel 2000';
xferAccess97   ='Access 97 Table';
xferAccess2000 ='Access 2000 Table';
xferDBF     ='dBase File (DBF)';
xferEpiInfo2000 ='EpiInfo 2000';
```

The above example shows the information declared as variables, and then set to a specific value in a separate statement. The values are not set with the DCL statement because that would add a fractional inefficiency to processing speed.

These file transfer formats are specifically declared, because they represent information known at build-time which is not expected to be changed or altered at run-time. In this case, the list represents some possible formats for transferring data to or from SAS. While the elements could have alternatively been declared as their content, providing an enumerated label insures that the application will have a single and central place to define the information. If the possibilities should change, the specific enumerated constants would be changed too. Finally, although this group of constants were defined as character variables, they could have been implemented as an SCL list, with the enumerated names being the names of the SCL elements.

The next example defines an SCL list in a class, and the job of this list is to enumerate all possible surveys:

```
public list
      surveyList/(
         InitialValue={
            '2003 GSPS - Global School Personnel Survey',
            '2003 GYTS - Global Youth Tobacco Survey',
            '2003 GYTS EURO - Global Youth Tobacco Survey',
            '2003 YTS  - Youth Tobacco Survey',
            '2002 GSPS - Global School Personnel Survey',
            '2002 GYTS - Global Youth Tobacco Survey',
            '2002 YTS  - Youth Tobacco Survey',
            '2001 GSPS - Global School Personnel Survey',
            '2001 GYTS - Global Youth Tobacco Survey',
            '2000 GSPS - Global School Personnel Survey',
            '2000 GYTS - Global Youth Tobacco Survey',
            '2000 NYTS - National Youth Tobacco Survey',
            '2000 YTS  - Youth Tobacco Survey'
           },
         Category='Survey Variable',
         Description='Returns the master list of surveys (displayed on choice screen)',
         AutoCreate='Yes',
         Editable='No',
         ValidValues=''
          );
```

In this case, the “initialValue” is declared along with the variable because class variables are automatically made upon object instantiation. Another way to define this enumerated list is to set the initial value in the constructor method. The elements could have been assigned descriptive labels; however, this list is never broken down or indexed so there were no advantages to naming the elements.

## CREATING STANDARDIZED SAS DATASETS

For this application, some standardized SAS datasets (also known as SAS tables) were used, and during the application’s processing, these datasets were typically partially read into memory, if at all. The term “standardized” refers to being the same at build-time and therefore not needing to be changed during run-time. What distinguishes the SAS dataset method of storage as opposed to the enumerated constant examples of variable or SCL list storage is the length.

In the “One-Time Methodology”, the length is categorically described as “Long Information” and “Short Information”. There is not an absolute definitive line between the two extremes, but the categories make sense because information will be either stored in a dataset and (“Long Information”) and hard coded into a variable or SCL list (“Short Information”). The developer will therefore make a tradeoff by assessing overall memory requirements, client and/or server processing speed, hard disk space, and ease of updating. For example, if abundant resources were available, the argument would lean toward all build-time data being enumerated within the application.

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 define specific Native American populations and also certain American territories. The state list is another example of slowly-changing information which does not need to be presented to users at run-time.

Overall, when a developer can classify information as non-modifiable (build-time), the resulting user interface will be simpler and reduce the possible errors when running the application.

## STANDARDIZED SAS DATASET EXAMPLE

The example presented here is the list of international countries (analogous to the US states dataset described earlier). It would have been nice to put the results of the Proc Contents for the standardized dataset which contains the countries, but this two-column format does not lend itself to sharing that output. The dataset has 242 observations, each row representing a “country” or “member” of the World Health Organization (WHO). Technically, not all the members are recognized universally as “countries”. There are four variables in the dataset, the full country name, the two-letter abbreviation for that country, the World Health Organization Region Abbreviation, and the World Health Organization Region Name.

Different parts of the application access this list. One point of access is on the Frame, where the information is read into an SCL list inside the Frame's SCL code. The actual transfer is inside an object, which simply moves the SAS dataset information into a list connected with that object. Then the Frame code populates the list box with the names of the "states" (which could be either American states or WHO members).

```
GETSTATEINFO:
   * State Information;
   statesObj.datasetID = open(statesFile,'I');
   if statesObj.datasetID then do;
      returnCode = statesObj.populateStateLists(surveyYearObj.surveyType);
      if returnCode then frameObj.systemMessage = 'ERROR:  CANNOT ESTABLISH STATE LISTS';

      * Conditionally Populate List Box with State Names;
      if not(frameObj.systemError) then do;
         statelst.items = statesObj.stateNameList;
         stateDetailList = copylist(statesObj.stateDetailList);
      end;
      else do;
         statelst.items = makelist();
         stateDetailList = makelist();
      end;

      if statesObj then do;
         returnCode = statesObj.CloseDataset();
         if returnCode then frameObj.systemMessage = 'ERROR:  CANNOT CLOSE STATES.OBJ DATASET';
      end;

   end;
   ELSE DO;
      frameObj.SystemMessage = 'ERROR:  DATASET NOT OPENED: ' || statesFile;
   END;
RETURN;
```

The information could have been stored as an SCL list within the catalog, but because the list periodically changes, it made more sense to make it available in the Windows Explorer world (it would have been hidden inside the SAS Catalog) where someone could see it, and modify the contents with Microsoft Excel (our typical tool of choice for modifying datasets).

The main point is that there are only two possible datasets which could be considered a "states" dataset, one which has American states and the other one which has WHO members ("countries"). Any of the thirteen surveys will access the appropriate standardized dataset, and there is no need to create multiple state datasets when new surveys are added. The states information may change slowly, but these changes are best implemented at build-time. By defining (or enumerating) a specific single dataset, there is one and only one place to change this state information, and one and only one place to access this information.

## CREATING FRAME VARIABLES

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.

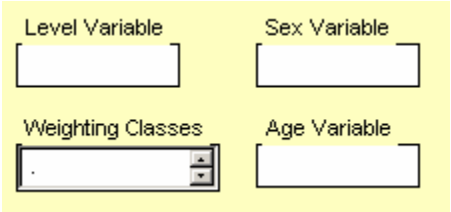
The survey processing screen (for a specific region within a state) has about 15 variables. These variables all have standard initial values dependent on survey type. In addition to presenting these 15 variables on the screen, they are also stored in a survey-specific dataset. For example, the YTS 2002 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. When the application is run again, those stored values are used as the starting defaults, which the analyst may or may not choose to change. The key requirement for the "One-Time Methodology" is having a single storage place for every category, and that ideal means storing pertinent frame variables in SAS datasets.

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 into a build-time category, 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.

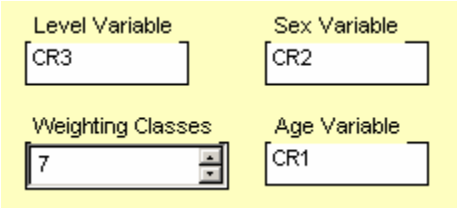
The SAS/AF documentation contains many examples of the rationale and methods of how to capture variables from a frame and integrate them into an application, as well as how to use Frame SCL to pass that information to and from SAS datasets (see SAS Institute, 2002).

## FRAME VARIABLE EXAMPLES

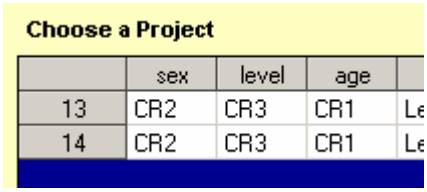
Here is a small screen shot of four variables, and how they look from the development environment:




Now, here are the same four variables as they appear when the program runs the first time for a specific survey and region:




The frame variables are all populated with default initial values, typically based on the type and year of a survey. Whether the user modifies them or not, when navigation moves away from this screen, the level, sex, and age variables are stored in a "master" dataset, which has a row for each region. This screen shot shows how these stored variables display on the SAS Data Grid object, and columns indicate sex, level and age (there are other columns too).




For the screen variable "Weighting Classes", a spin box object was chosen to visually indicate that the possible values for "weighting classes" are integers. This control will not accept character input, and is an example of validity checking inherent in design. By contrast, the sex, level, and age variables are all input in character format, and therefore the application validates those entries with the stored layout for that region. In other words, there are ways to validate character fields too.

Note that enumerated constants are never exposed to the frame because they are not intended to be easily changed. Any build-time information may display on the frame, and sometimes that information does, but it is not modifiable from the analyst's perspective.

## CREATING 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 starting collection of "master datasets" ("master" means 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 initially populated datasets.

For example, the questionnaire layout is different for each combination of survey type and year. 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 customized SAS 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.

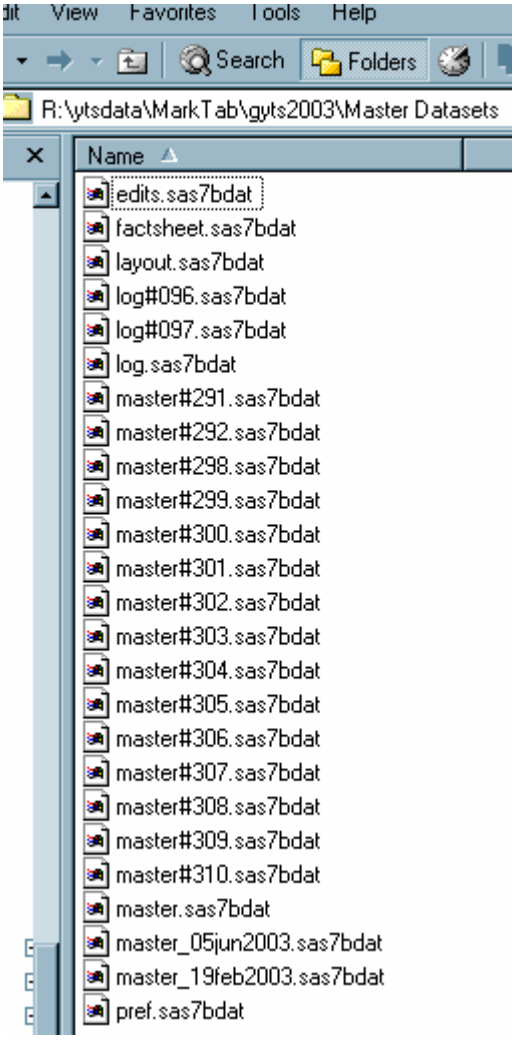
At one point, the customized layout is used to generate an INPUT statement (within a data step) to read raw data from the original ASCII file. At another point, the customized layout is used to generate a dataset which is essentially the questionnaire in printable format. A third use is when the application uses the customized layout file to generate report titles and formats for PROC TABULATE labels.

## CUSTOMIZED SAS DATASET EXAMPLE

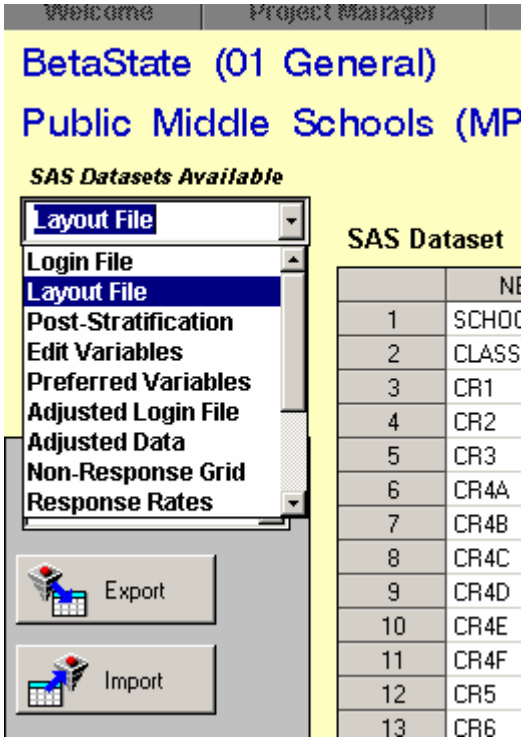
The following screen shot of Windows Explorer shows the names of SAS datasets in the "Master Datasets" folder:

When a new region is created, four SAS datasets from this "Master" folder are copied into the region's subfolder nominally called "SASData". These four files include the "edits", "factsheet", "layout", and "pref" (preferred variables). These files are standard for a specific survey type and year, but are typically customized for specific regions.

Within the Frame, there is a "Datasets" tab which makes available the option of viewing a dataset, or importing or exporting it to various file formats. In our experience, Microsoft Excel has been the tool of choice for customizing datasets because it easily allows for variable character lengths (otherwise we would have used Microsoft Access).



The next screen shot shows the name of the country ("Beta State" is the test region for software development only, a very odd region which contains all the exceptions).




In the above example, "01" means region one, and "General" is the name of that region. "Public Middle Schools" is the type of survey (there are potentially six different types for each region). The "SAS Datasets Available" lists the available datasets, and this drop-down box is linked to the SAS data grid to the right. The picture shows the "layout file" chosen, and the first 13 rows of that file display on the right. Covered up is a drop-down box which allows for choice of export formats, and the "export" and "import" button look for those files in the "SASData" subdirectory (NOT in the master datasets directory). Importing a file will overwrite a region-specific SAS dataset which may already exist. The development structure easily allows for more region-specific datasets to be created. Standardized datasets are never listed because they are not intended to be easily changed (at run-time).

In some cases the application applies checks when the data is input into SAS format from ASCII or Excel. The "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.

SCL is the central processing 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. Validity checks were all performed in SCL, as prerequisites to running processes, or in the Frame-related code.

Both during the eight 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.

## CONCLUSION

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. Information is enumerated in specific hard-coded variables or SAS datasets. The model has the advantage of having a single, unique home for all application data, and the concept flexibly allows the developer's to move elements among categories while refactoring (Fowler, 1999).

Further information is available on this application's class structure (Tabladillo, 2003a) and development (Tabladillo, 2003b).

## REFERENCES

Fowler, Martin (1999), *Refactoring: Improving the Design of Existing Code*, Reading, MA: Addison Wesley Longman, Inc.
SAS Institute Inc. (2002), *SAS OnlineDoc 9*, 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. (2003a), "Application Refactoring with Design Patterns", *Proceedings of the Twenty-Eighth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute, Inc.
Tabladillo, M. (2003b), "The One-Time Methodology: Encapsulating Application Data", *Proceedings of the Twenty- Eighth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute, Inc.

## ACKNOWLEDGMENTS

Thanks to all the great public health professionals at the Office on Smoking and Health, Center for Chronic Disease.

## TRADEMARK CITATION



## CONTACT INFORMATION

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

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