The Dataset Attribute Family of Classes

Mark Tabladillo, Ph.D., Atlanta, GA

ABSTRACT

This presentation will specifically present the dataset attribute family, an abstract parent and its twenty-five children. This family of classes provides information on and from SAS® datasets, and the structure is based on the Analysis Matrix design pattern (Shalloway and Trott, 2002). The structure is of generic utility for any application which accesses SAS datasets.

Extensive experience with classes or objects or design patterns (Gamma, et. al., 1995) is not necessary for this talk, but the presentation assumes knowledge of how to code a class and instantiate an object with SCL. The core techniques apply to any operating system and any object-oriented language.

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).

This application has been previously documented, both in terms of overall development (Tabladillo, 2003b) and class structure (Tabladillo, 2003a). In summary, during its lifetime, the application grew from zero to forty-four classes. This presentation will specifically define and describe the dataset attribute family, an abstract parent and its twenty-four children, a family developed to automate many common dataset handling functions and sequences within SCL.

DATASET ATTRIBUTE OVERVIEW

Twenty-five of 44 classes in the current application are part of the dataset attribute family. The single parent class is called DATASET_ATTR and given a dataset ID (or number), it will determine the core dataset characteristics from the SCL ATTRC and ATTRN commands, and store the results in object variables. The five methods are 1) create a new dataset, 2) open an existing dataset, 3) update an existing dataset, 4) close a dataset, and 5) delete a dataset.

The subclasses inherit the parent's functionality using the optional EXTENDS command on the CLASS statement. Each of the 24 subclasses refers to a different type of dataset. For example, the class DATASET_ATTR_LAYOUT refers to the questionnaire, and DATASET_ATTR_DATA refers to the initial survey data. Inside these subclasses, the variable numbers are determined for the standardized list of expected variable names. Some variables are required to run processes, and other variables are optional (if included, they will trigger certain processes). These subclasses retain the results from the VARNUM command in object variables.

For example, putting the survey questionnaire layout in its own class allows the developer to define two objects for the same type of dataset (a layout) and possibly do something with these two layouts together. The application compares the standardized master layout with the region-specific (typically customized) layout for inconsistencies during customization.

One subclass of general use is called DATASET_ATTR_PROCFREQ and was created specifically to read the standard output dataset generated by proc freq. Both the count and the percent are kept as object variables when the object is presented with a valid dataset ID.

Though there are 23 different declared types of datasets (adding the generic type makes 24 classes), future subclasses could easily be added by subclassing the parent class, and then customizing the subclass attributes and functions to reflect the structure of the new dataset. The family relationship most closely matches the Strategy design pattern (Gamma, et. al., 1995), with the common overridden method being the one described by the datasetID attribute.

Gamma, et. al., 1995 summarize the Strategy design pattern as follows:

> Strategy (page 315) Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it. (inside cover)

The dataset attribute family describes the different strategies available, with each subclass having a different set of methods (a different "algorithm") unique to that dataset.

BASE FUNCTIONS: ATTRN AND ATTRC

The dataset attribute family started with the desire to systematically include the information made possible by the SCL commands ATTRN and ATTRC. These two commands return attributes for a SAS table (or dataset). ATTRN returns numeric attributes and ATTRC returns character attributes.

The following table summarizes the information available through these commands (for version 8); all the attributes are listed because all the attributes were coded into the dataset attribute base abstract class:

Attributes for the ATTRC Function

'CHARSET' -- returns a string indicating the character set of the machine that created the SAS table

'ENCRYPT' -- returns 'YES' or 'NO' depending on whether the SAS table is encrypted.

'ENGINE' -- returns the name of the engine used to access the SAS table.

'LABEL' -- returns the label assigned to the SAS table.

'LIB' -- returns the libref of the SAS data library in which the SAS table resides.

'MEM' -- returns the name of the SAS data library member.

'MODE' -- returns the mode in which the SAS table was opened

'MTYPE' -- returns the type of the SAS data library member.

'SORTEDBY' -- returns an empty string if the SAS table is not sorted. Otherwise, returns the names of the BY columns in the standard BY statement format.

'SORTLVL' -- returns an empty string if the SAS table is not sorted. Otherwise, returns a code indicating sort.

'SORTSEQ' -- returns an empty string if the SAS table is sorted on the native machine or if the sort collating sequence is the default for the operating system. Otherwise, returns the name of the alternate collating sequence that is used to sort the file.

'TYPE' --is the SAS table type.

Attributes for the ATTRN Function

'ANY' -- specifies whether the table has rows or columns.

'ALTERPW' -- indicates whether a password is required in order to alter the SAS table.

'ANOBS' -- indicates whether the engine knows the number of

rows.
'ARAND' -- indicates whether the engine supports random access.
'ARWU' -- indicates whether the engine can manipulate files.
'CRDTE' -- returns the SAS table creation date. The value returned is the internal SAS DATETIME value for the creation date. Use the DATETIME format to display this value.
'GENMAX' -- returns the maximum number of generations.
'GENNEXT' -- returns the next generation number to generate.
'ICONST' -- returns information on the existence of integrity constraints for a SAS table.
'INDEX' -- indicates whether the SAS table supports indexing.
'ISINDEX' -- indicates whether the SAS table is indexed.
'ISSUBSET' -- indicates whether the SAS table is a subset.
'LRECL' -- returns the logical record length.
'LRID' -- returns the length of the record ID.
'MODTE' -- returns the last date and time the SAS table was modified. Use the DATETIME format to display this value.
'NDEL' -- returns the number of deleted rows in the SAS table.
'NLOBS' -- returns the number of logical rows (those not marked for deletion). An active WHERE clause does not affect this number.
'NLOBSF' -- returns the number of logical rows (those not marked for deletion) that match the active WHERE clause.
'NOBS' -- returns the number of physical rows (including those marked for deletion). An active WHERE clause does not affect this number.
'NVARS' -- returns the number of columns in the SAS table.
'PW' -- indicates whether a password is required in order to access the SAS table.
'RADIX' -- indicates whether access by row number is allowed.
'READPW' -- indicates whether a password is required in order to read the SAS table.
'TAPE' -- indicates whether the SAS table is a sequential tape file.
'WHSTMT' -- returns information about active WHERE clauses.
'WRITEPW' -- indicates whether a password is required in order to write to the SAS table.

In practical use, most of the above table is not accessed. However, coding all the options into the parent class allows future possible use.

FIVE CORE METHODS

There are five core methods in the dataset family attribute classes. The five methods are 1) create a new dataset, 2) open an existing dataset, 3) update an existing dataset, 4) close a dataset, and 5) delete a dataset. The first three methods are based on the OPEN command, the fourth based on the CLOSE command, and the last based on the DELETE command.

Probably the most used method is the OPEN method, which opens a dataset in the default "input" mode (meaning that values can be read but not modified). Here is the code for that specific method.

```
OPENDATASET:public method
      inputLibnameDataset:INPUT:CHAR
      return=num
      /(
         Description='Opens dataset in Input
Mode'
      );
      DCL
         num
            returnCode
            ;
      * INPUT mode -- values can be read but
cannot be modified. (This is the default.);
      returnCode = 0;
      if exist(inputLibnameDataset,'DATA') then
do;
         datasetID =
open(inputLibnameDataset,'i');
         if datasetID then libnameDataset =
inputLibnameDataset;
         else
            systemMessage = '*** NETWORK ERROR:
DATASET CANNOT BE OPENED IN INPUT MODE -- ' ||
inputLibnameDataset;
      end;
      else do;
         systemMessage = '*** NETWORK ERROR:
SAS DATASET DOES NOT EXIST -- ' ||
inputLibnameDataset;
      end;
      return(systemError);
ENDMETHOD;
```

Here are some comments about this openDataset method:

1. The command uses the EXIST test first to be able to send an error message specifically based on a dataset not existing. The 'DATA' option on the EXIST command insures that the method is attempting to open a dataset (as opposed to, for example, a catalog).
2. The OPEN command has the "I" flag, indicating input mode.
3. Upon a successful open, the class variable libnameDataset is set to the value of inputLibnameDataset. The main use of storing that name is to allow a DELETE command in the future.
4. Any text stored in "systemMessage" is stored in an SCL list; the variable "systemError" is the length of the stored error list, and the number of items in this list is what the method returns. SystemError can thus be used as a test condition for an if/then statement.

The second method, UpdateDataset, is similar, but based on the UPDATE command.

```
UPDATEDATASET:public method
      inputLibnameDataset:INPUT:CHAR
      return=num
      /(
         Description='Opens dataset in Update
Mode'
      );
      DCL
         num
            returnCode
            ;
      * UPDATE mode -- values in the table can
be modified and rows can be read in random
order;
      returnCode = 0;
      if exist(inputLibnameDataset,'DATA') then
do;
         datasetID =
open(inputLibnameDataset,'U');
         if datasetID then libnameDataset =
inputLibnameDataset;
         else
```

```
            systemMessage = '*** NETWORK ERROR:
DATASET CANNOT BE OPENED IN UPDATE MODE -- ' ||
inputLibnameDataset;
      end;
      else do;
         systemMessage = '*** NETWORK ERROR:
SAS DATASET DOES NOT EXIST -- ' ||
inputLibnameDataset;
      end;
      return(systemError);
ENDMETHOD;
```

The third NewDataset method does not have an EXIST test.

```
NEWDATASET:public method
      inputLibnameDataset:INPUT:CHAR
      return=num
      /(
         Description='Opens dataset in New Mode'
      );
      DCL
         num
            returnCode
            ;
      * NEW mode -- creates a new table. If
table-name already exists, the table is replaced
without warning;
      returnCode = 0;
      datasetID = open(inputLibnameDataset,'N');
      if datasetID then libnameDataset =
inputLibnameDataset;
      else do;
         systemMessage = '*** NETWORK ERROR:
DATASET CANNOT BE OPENED IN NEW MODE -- ' ||
inputLibnameDataset;
      end;
      return(systemError);
ENDMETHOD;
```

The fourth CloseDataset method is based on the datasetID numeric attribute, which would have been stored only if a successful open was completed:

```
CLOSEDATASET:public method
      return=num
      /(
         Description='Closes dataset and resets
datasetID'
      );
      DCL
         num
            returnCode
            ;
      * Close all SAS tables as soon as they are
no longer needed by an application;
      returnCode = 0;
      if datasetID > 0 then do;
         returnCode = close(datasetID);
         if returnCode = 0 then datasetID = 0;
         else do;
            systemMessage = '*** NETWORK ERROR
IN CLOSING DATASET ' ||
putn(datasetID,'best7.');
         end;
      end;
      return(systemError);
ENDMETHOD;
```

As in the openDataset method, the CloseDataset method also returns the number of error messages as its return code.

Finally, the last core method is the DeleteDataset method, which is based on the class variable libnameDataset (set only upon a successful open).

```
DELETEDATASET:public method
      return=num
      /(
         Description='Deletion of dataset'
      );
      DCL
         num
            returnCode
            ;
      * Deletes a member of a SAS data library -
- the EXIST requirement restricts to SAS tables
only;
      returnCode = 0;
      if datasetID then returnCode =
closedataset();

      if not(exist(libnameDataset,'DATA')) then
         logMessage = 'Attempted to delete ' ||
libnameDataset || ', which does not exist as a
Dataset';

      if not(systemError) and
exist(libnameDataset,'DATA') then do;
         returnCode = delete(libnameDataset);
         if returnCode = 0 then datasetID = 0;
         else do;
            systemMessage = '*** NETWORK ERROR
IN DELETING DATASET ' || libnameDataset || ' '
|| putn(datasetID,'best7.');
         end;
      end;
      return(systemError);
ENDMETHOD;
```

In this fifth and last DeleteDataset method, the code first attempts to close the open dataset. This check is useful because this SAS/AF application allows the analyst to run any ad-hoc code during processing. The EXIST command is used to verify that indeed a dataset is attempting to be deleted. As with the other documented methods, the return code is the number of errors.

These five core methods define the dataset attribute family of classes by providing a datasetID (numeric variable) and libnameDataset (character variable). The datasetID is the central value which indicates that a dataset is OPEN (in one of the three possible modes). There is a fourth mode of OPEN, called UTILITY (the “V” option), but operationally the software instead uses proc datasets for any of those types of modifications, since we desire to document those types of changes in the stored logs.

ATTRC AND ATTRN ATTRIBUTES

Once a datasetID is available, the ATTRC and ATTRN can be optionally available to a calling program. Each attribute has its own class-level variable, linked to a GetCAM method. By definition, the GetCAM option on an attribute specifies the method triggered when someone wants to obtain the attribute value (SAS Institute, 2002).

The following example is for NLOBS, one of the most common attributes used. First, the numeric variable NLOBS is defined for the class:

```
public num
      NLOBS/(
         InitialValue=_BLANK_,
         Category='Dataset ATTRN Attribute',
         Description='Returns the number of
logical rows (those not marked for deletion). An
active WHERE clause does not affect this
number',
         GetCAM='NLOBS',
         AutoCreate='Yes',
         Editable='No',
         ValidValues=''
         );
```

Notes on the above method:

1. The variable is public, meaning it can (and typically is) called from the outside.
2. The variable is linked to an internal method called "NLOBS" through the GetCAM feature.
3. The value is not editable from the outside.

The code below is the internal NLOBS protected method, triggered by the GetCAM feature:

```
NLOBS:protected method NLOBS:update:num
       /(
         Description='Obtain updated NLOBS based
on valid numeric dataset ID'
      );
      IF datasetID > 0 then NLOBS=
ATTRN(datasetID,'NLOBS');
      ELSE NLOBS= .;
ENDMETHOD;
```

Because the ATTRC and ATTRN commands return valid values contingent on a valid datasetID, the above code reflects that same requirement, and will only attempt to use the ATTRN command when a valid datasetID exists. Otherwise, the value returned is missing (for character values, a blank space would be returned). There are no error codes or log messages incorporated in the GetCAM methods, but there could be. As ATTRN or ATTRC attributes are added to future versions of SAS, the code would need to be appended by adding class-level variables and GetCAM methods.

These variables and methods describe how the abstract parent class makes dataset attributes available. Any code which accesses a subclass of this parent only needs to refer to *objectName.datasetAttribute* to obtain the appropriate value. If there is no datasetID, then the value returned is missing. The value of *objectName.datasetAttribute* cannot, by definition (Editable='No'), be set from any code outside the dataset attribute family.

VARIABLE INFORMATION

Variable names are particular to specific types of datasets. This application creates a series of subdirectories for specific regions of countries. Each subdirectory may have the same types of datasets, but a different structure and/or content. For example, the survey questionnaire layout is typically customized, and each subdirectory will therefore have a slightly different layout. The column names (variables) will usually be the same, but the rows (observations) will be different.

The abstract parent class variable datasetID has a SetCAM method named "DatasetID", which is triggered when the value is set. By definition, the SetCAM option on an attribute specifies the method triggered when someone wants to set the attribute value (SAS Institute, 2002). There are no statements in the abstract parent's datasetID method, but the subclasses override this parent method providing the ability to extract variable numbers with the SCL VARNUM command.

The following example shows an overridden datasetID method for the edits file, a simple file with two expected column variables, CRITERIA and EDITVAR:

```
* Override the datasetID SETCAM to provide
VARNUMs;
datasetID:protected method datasetID:num
       /(
         Description='Obtain dataset attributes
based on valid numeric dataset ID',
         State='O'
      );

      _super(datasetID);

      * Populate VARNUMs;
      if datasetID > 0 then do;
         varnum_criteria   =
varnum(datasetID,'criteria');
         varnum_editvar    =
varnum(datasetID,'editvar');
      end;
      else do;
         varnum_criteria   = .;
         varnum_editvar    = .;
      end;

endmethod;
```

Notes:

1. The method overrides the datasetID method, linked to the SetCAM event in the abstract parent class.
2. Because this is an overridden method, the _SUPER command is issued to allow for future universal code in the abstract parent method datasetID.
3. VARNUM_CRITERIA and VARNUM_EDITVAR are two numeric class-level variables. These variables are defined in the subclass named DATASET_ATTR_EDITS.
4. Appropriately, the variable numbers are only valid during the times the dataset is open, a statement true only when datasetID has a value.

Other dataset attribute subclasses typically have more variables than this simple example, which makes the overridden datasetID method longer. In some classes, the overridden datasetID method also obtains variable types, variable lengths, or actual data from the dataset. This information is stored by child classes in class-level variables or SCL lists.

One specific subclass is called DATASET_ATTR_PROCFREQ, and here is the overridden (State='O') datasetID method and constructor for that class:

```
* Override the datasetID SETCAM to provide
VARNUMs;
datasetID:protected method datasetID:num
       /(
         Description='Obtain dataset attributes
based on valid numeric dataset ID',
         State='O'
      );

      _super(datasetID);

      * Populate VARNUMs;
      if datasetID > 0 then do;
         varnum_freqvar=
varnum(datasetID,freqvar);
         varnum_count =
varnum(datasetID,'count');
         varnum_percent =
varnum(datasetID,'percent');
      end;
      else do;
         varnum_freqvar = .;
         varnum_count = .;
         varnum_percent = .;
      end;

endmethod;


* CONSTRUCTOR;
dataset_attr_procfreq:public method invar:char
      /(
         Description='Constructor passing
FREQVAR parameter'
      );

      freqvar = invar;
endmethod;
```

The above class is constructed by passing the character variable INVAR to the method. That value is saved in the class-level attribute FREQVAR, and becomes the attribute which determines VARNUM_FREQVAR. The other two numeric variables obtained are the count and the percent. This subclass takes the output from PROC FREQ and makes it available to the SCL environment.

CUSTOMIZED METHODS

Refactoring will continually take place, and it will involve thinking about what the application knows and does, and how to best express that functionality behind the scenes. Fowler (1999) provides many specific and concrete examples of the types of things which can be done to refactor.

During the refactoring process, it was logical to encapsulate dataset-specific methods in the dataset attribute class. Instead of providing specific code, several examples are described here which could form the creative base for creating your own customized methods.

One example is obtaining an SCL list which has all the unique values from a particular column.

A second example is the layout file, which conditionally allows for up to 26 possible answers on a particular question (the input data is alphabetic, so 26 comes from the length of the alphabet). Typically, there are only eight possible answers, but in many customized surveys, the total possible goes beyond the standard eight. The DATASET_ATTR_LAYOUT subclass has the ability to determine how many of those optional columns are on the customized layout dataset.

Another third example is where the layout subclass determines how many valid answers there are for a specific question. Though there may be eight possible answers, the actual question may only have two possible responses ("yes" and "no") and the layout subclass has a method to determine how many valid answers a particular question has. This type of information is inherent to the layout dataset definition, and encapsulating the functionality in the layout subclass centralizes the information in a single known location. The alternative location for this code would be in the calling code (whether in the frame or another class), a strategy which works but could easily lead to repetitive code throughout the application. Encapsulation is therefore the preferred approach, and since the dataset attribute family has a specific function and definition, the developer more easily can judge what should be included as a customized method.

SUBCLASSING SUBCLASSES

In some structures, it may make sense to create a second (or higher) level of subclasses. This extension helps when a core set of dataset definition applies to a group of datasets, each of which may have particular differences.

In this application, the DATASET_ATTR_LAYOUT_DATED subclass was built on the layout subclass. The class functionally compared two layout files together. Operationally, that same functionality could have been also placed in a layout customized method. Whether to expand methods into another class or collapse them into a parent class Is purely a function of refactoring and developer's choice (Fowler, 1999), and the decision need not remain static for all time.

CONCLUSION

This paper has discussed the dataset attribute family, an abstract parent and its twenty-five children. This family of classes provides information on and from SAS® datasets, and the structure is based on the Analysis Matrix design pattern (Shalloway and Trott, 2002).

The abstract parent class holds responsibility for assigning values to datasetID and libnameDataset, and make ATTRN and ATTRC attributes available at the class level. The subclasses allow each type of dataset to have variable-level information, including variable numbers, variable types, variable lengths and actual data. Subclasses also can have customized methods.

Refactoring the abstract parent allows for new ATTRN and ATTRC attributes to be added with subsequent versions of SAS. Refactoring the subclasses allows for adding variable information or customized classes. The whole family structure and relationships can also be changed.

The structure is of generic utility for any application which accesses SAS datasets. 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.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995), *Design Patterns: Elements of Reusable Object-Oriented Software*, 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/