Creating a Data Dictionary for Large SAS Data Sets
Steve Wilson, AMEX Life Assurance, San Rafael, Ca.

ABSTRACT
In many processing environments, SAS programs are commonly used to read data from a flat file and "load" it into a SAS data set for analysis purposes. The users of this loaded data are often provided with scant information about the data set, though.

Although PROC CONTENTS provides descriptive information of the data set, this information is generally not enough to satisfy the user's very real concerns about how and where the variables in the data set were obtained. In Version 6 of SAS, this problem is intensified because Version 6 SAS data sets do not contain the source code used to create them.

This paper will discuss methods that can be used to automatically construct a comprehensive on-line data dictionary for the "loaded" SAS data set. The resulting data dictionary can be viewed using SAS/FSP.

The SITUATION
Consider this situation:
I am the user of a SAS data set containing over 700 variables. The data set was created for me by 'someone' in the Systems group. They only supplied me with a data set name and a copy of the PROC CONTENTS.

I have to prepare an analysis, but as I begin to work with the data, a number of questions about the data set arise.

Are missing or unknown values re-coded?
Some values look odd. Were they read correctly from the input file?
Some variables are codes which make no sense to me. Can they be decoded?
How was the derived variable XYZ created?

My usual course in answering these questions is to call Systems and try to get the answers from whomever loaded the data. If I'm lucky, I know who did the load and that person is familiar enough with the load program to give me the answers immediately.

At the other extreme, I've got to find the load program myself and look through hundreds of lines of unfamiliar SAS code to find the answers myself.

Usually, though, I hear:
"I'll have to check the program and get back to you later."

Not only is this frustrating for me, but it also wastes the time of the Systems department because they have to research my every request. With enough requests, relations become strained, and they stop returning my calls.

There must be a better way!

A SOLUTION
SAS application programmers can easily avoid this situation by automatically generating a separate SAS data set containing information on the loaded data and providing the users of the loaded data with easy access to that information. This information data set is called a Data Dictionary.

There are only 3 simple steps in creating a data dictionary for end users. They are:

1) Determine what information the users want to know about the load. Talk to the users! At a minimum, all dictionaries should contain the following information:
   a) Variable names and attributes
   b) How the variables were obtained
      Either INPUT statement specifications or code used to create derived variables

2) When your "load" program is finished, a dictionary constructor program (or set of programs) is written to create the dictionary. These can be as specialized as desired.

3) Create easy access for the users. The best way to do this is to set up a command that will automatically allocate the necessary data sets, invoke SAS, and execute PROC FSBROWSE on the data dictionary, using a pre-defined SAS/FSP screen.

While not minimizing the importance of step 1, this paper will focus on steps 2 and 3.
CONSTRUCTING THE DICTIONARY

For a simple data dictionary, a SAS program must be written that reads the "load" program. At a minimum, this program will read the INPUT statement specifications for variables that were input from the flat file and the SAS source code that was used to create any derived variables.

For more comprehensive data dictionaries, other programs may be written that read associated format catalogs, ancillary programs, the contents of the loaded data set, or even the loaded data itself.

These programs are collectively known as the "Dictionary Constructor". The Dictionary Constructor can be designed to accumulate almost any type of information desired about the loaded data.

Naturally, if a standard "load" program design is used, the Dictionary Constructor can be used to construct dictionaries for numerous loaded data sets. Thus, a side benefit of this methodology is the development of a set of coding standards for "load" programs.

The Dictionary Constructor works by using SAS character functions to search the "load" program for pre-defined SAS keywords or special comments imbedded in the "load" program. These special comments are used to help the Dictionary Constructor identify SAS programming statements. The Dictionary Constructor knows what to do with the SAS statements in the "load" program once they have been identified.

SAS Keywords
Certain SAS keywords appear in all "load" programs. The use of these keywords is unambiguous. When appearing as the first word of a statement, the Dictionary Constructor knows what will follow. These keywords are:

DATA PROC RUN
These keywords are used by the Dictionary Constructor to determine the Step type.

KEEP
This keyword is used by the Dictionary Constructor to determine which variables are to be kept in the dictionary. This keyword may appear as a KEEP statement or as a KEEP= data set option.

ATTRIB or LABEL LENGTH FORMAT INFORMAT
These keywords are used by the Dictionary Constructor to determine variable attributes.

FILENAME INFIL E INPUT
These keywords are used by the Dictionary Constructor to determine where and how the data was input.

Special Comments
Special SAS comment statements are imbedded in the "load" program by the programmer who writes the "load" program. These comments are known to the Dictionary Constructor. They are used to pass information to the Dictionary Constructor or to tell the Dictionary Constructor what SAS statements follow.

My experience has shown that most "load" programs only require 4 types of special comments to capture the data that end users want to see in a data dictionary. These are:

*A varname;
This comment tells the Dictionary Constructor that the SAS statements to follow are used to assign a value to the variable VARNAME. The block of code to perform this operation is referred to as an "assignment block". The assignment block for variable VARNAME ends when the Dictionary Constructor encounters the next SAS keyword or special comment.

*D variable description;
This comment is used in an assignment block to pass a variable's description to the Dictionary Constructor. Unlike a variable label, there is no 40 character limit to this description. If no description is supplied, a variable's label will be placed in the dictionary.

*I;
This comment tells the Dictionary Constructor to ignore all the SAS statements that follow until the next SAS keyword or special comment is encountered.

*M;
This comment tells the Dictionary Constructor that the following block of code is used to re-set missing values.
LOAD PROGRAM CODING SCHEME

This is an example of a standard "load" program that a Dictionary Constructor would read:

```sas
FILENAME FLATFILE 'LMA5.FLAT.FILE.FOR.STEVE';
DATA SASUSERDATA(KEEP=ACCEDEATH ADDR_DTE ...... ZIP);
LENGTH ACCEDEATH $1 ...... ZIP $5;
FORMAT ADDR_DTE MMDDYY8. ..... CPTYPDCI SCPTYPDCI;
INFILE FLATFILE EOF=DONE;
INPUT CYCLE $ 1-4
    BLOCK $ 5-8
    CPNUMCD $PNUMCD
    CPYPCDI SCHRYPCD
    CP_DTE MMDDYY6.;

/* The Dictionary Constructor can read the above code */
/* by scanning for the previously mentioned SAS */
/* Keywords. The Special Comments are used in the */
/* code below to identify the SAS statements. */
/************************************************************************
* I Ignore statements used to subset observations input; IF PUT(STATE,$STATE.) = '00' THEN DELETE;
/************************************************************************
/* Assignment statements used to created derived vars */
/************************************************************************
* A DOL_TRV;
*D Total Travel Dollars (Lodging, Air, Car) over past year; DOL_TRV = SUM(DOL1_DG,DOL1_AIR,DOL1_CAR,
DOL2_LDG,DOL2_AIR,DOL2_CAR,
DOL3_LDG,DOL3_AIR,DOL3_CAR);

*A DEPOSIT;
*D Product prospect score for Deposit account;

/************************************************************************
* Convert Missing/out of range values */
/************************************************************************
*M;
 IF DOL_TRV = . THEN DOL_TRV = 0;
 IF GENDER NOT IN('M' 'F') THEN GENDER = 'U';
 IF CPTYPDCI = ' ' THEN CPTYPDCI = 'P';

*I Ignore the remaining program statements ;
OUTPUT;
RETURN;
DONE:
FILE LOG;
PUT @1 "Number obs input: "._N_. $;
RETURN;
RUN;

The SAS/FSP screen to view a simple dictionary created by reading the previous "load" program could look something like this:

Browse SAS data set: DATADICT.SIMPLE Screen 1
New
Command =>
Variable: Page __ of __
Type: Format: Length: 
(Printing Instructions) (Bytes for storage)
How variable obtained:

PF1=Help ... PF3=End ... PF7=Prev Page ... PF8=Next Page

As you can see, the variable name, description, and attributes obtained from the "load" program will appear in the top portion of the screen. These attributes can be compared with the contents of the loaded data set to verify that the "load" program worked as intended.

The rest of the screen contains room to display the exact SAS code that was used to obtain the variable and any statement that is used to re-set missing values.

The most important concepts to note on this screen are:

1) The screen contains "PAGE ____ of ____". This is critical because some assignment blocks will be too large to be contained in one observation in the dictionary. Allow for about 10 lines of code per observation.

2) At the bottom of the screen are descriptions of the available PF Key commands. Whenever developing screens for use by others, the screens should contain basic instructions.

3) Notice that the dictionary data set contains many long character variables that will often be blank. Thus, a big I/O and data storage savings can be obtained by compressing the dictionary data set.
Using the previous "load" program as input into the Dictionary Constructor, the following observations would appear in the data dictionary.

A variable that was simply input from the flat file would appear in the data dictionary like this:

```
Browse SAS data set: DATADICT.SIMPLE  Screen 1
Command ==>
Variable: CPTYPCD1  Page 1 of 1

Type: CHARACTER  Format: $CPTYPCD  Length: 3
(Printing Instructions)  (Bytes for storage)

How variable obtained:
INPUT @968 CPTYPCD1 $CHAR3.
```

Missing or blank values reassigned:
IF CPTYPCD1 = ' ' THEN CPTYPCD1 = 'P';

A pad character of ' ' has been defined for some of the elements on the screen to keep it looking neat.

A derived variable would appear in the dictionary as follows. Notice that two observations are needed in the data dictionary to adequately display the entire assignment block used to create the variable.

```
Browse SAS data set: DATADICT.SIMPLE  Screen 1
Command ==>
Variable: DEPOSIT  Page 1 of 2

Type: NUMERIC  Format:  Length: 3
(Printing Instructions)  (Bytes for storage)

How variable obtained:
SC5A=0;
IF TRL_NTE=1 THEN SC5A=1;
ELSE IF (2 LE TRL_NTE LE 5) THEN SC5A=2;
ELSE IF TRL_NTE GT 5 THEN SC5A=4;
IF INS_AIRF='0' THEN SC5B=1;
ELSE SC5B=0;
IF MTHS_CM <= 53 THEN SC5C=1;
ELSE IF (54 <= MTHS_CM <= 149) THEN SC5C=2;
SC5D=0;
IF HOBBYDOL GT 75 THEN SC5D=1;
DEPOSIT = SC5A + SC5B + SC5C + SC5D;
```

Missing or blank values reassigned:
PF1=Help...PF3=End...PF7=Prev Page...PF8=Next Page

Two flags appear on the first observation to alert the user that another observation contains the remaining assignment block code. These are the "Page 1 of 2" at the top of the screen, and the "Continued" that appears immediately below the last line of displayed code.

Also note that all attribute information is duplicated on each observation in the dictionary for variables requiring more than one observation in the dictionary.

Now, for a more comprehensive dictionary, the Dictionary Constructor can be programmed to analyze the loaded data to append frequency counts or summary statistics into the dictionary.

Other Special Comments can also be defined to the Dictionary Constructor to allow other information to be passed into the dictionary.

The screens on the following page display the information contained for one variable in a more complex data dictionary. This dictionary is for a SAS data set that is comprised of data loaded from four different flat files.

The first screen is used to view a separate summary data set that contains information on the load status of each of the four separate input files. This summary data set does not contain information on the variables in the loaded SAS data set, but instead it contains one summary observation on this more complex load.
The user is provided with easy movement from this summary screen to view the individual variables in the loaded SAS data set.

The SAS/FSP screen created to view the variables contained in this more comprehensive data dictionary follows. This is a three part screen.

The first screen again contains the variable attributes and any statement that is used to re-set missing values. Additionally, this screen shows simple univariate statistics that were calculated from the loaded data as well as information obtained by reading another program that was used to load variables from the SAS data set into DB2 tables.

By pressing PF11 the user can view the following screen for this variable.
BENEFITS

1) Programming standards are developed for “load” programs because “load” program coding goes hand-in-hand with the Dictionary Constructor.

2) Many users don’t like working with data created by other people. The data dictionary can relieve some of this anxiety.

3) Data integrity is enhanced because errors are more readily detected. Alert users can find errors in variables of concern by investigating the code in the dictionary. Application programmers can also find some errors via standard exception reports they generate from the dictionary.

4) “Load” programs are structured better because programmers know that users will be examining their code.

5) One Dictionary Constructor can create dictionaries for numerous loaded data sets. When non-standard “load” programs are created, only minor modifications will usually be required for the Dictionary Constructor.

6) Reversing the process can allow for the generating of “load” programs from an existing dictionary. This opens the interesting prospect of users themselves being able to define variables to be created during a load.

LIMITATIONS

1) Arrays that process data for numerous kept variables cannot be read by the Dictionary Constructor. Removing arrays from “load” programs, however, will actually enhance the performance of the program since array processing takes substantially more time than executing each line of code separately. This is because “array indices are floating point numbers and array elements can be of varying length”. (See reference 1)

2) Temporary derived variables that are used to create more than one kept variable are problematic. Since the temporary variable does not appear in the dictionary, its appearance in the kept variable’s assignment block can be confusing to users. Below are two alternatives to solving this dilemma.

   a) The code used to create the temporary variable can be copied into each assignment block where the temporary variable is needed. Thus, the code to create the temporary variable would appear in the dictionary every time the temporary variable is used.

   ```
   A VAR1;
   TEMP = ...;
   VAR1 = operation using TEMP;
   A VAR2;
   TEMP = ...;
   VAR2 = operation using TEMP;
   ```

   b) If the temporary variable is used too often for the above option, the temporary variable can simply be kept in the final data set so that the temporary variable can be located in the dictionary.

   ```
   A TEMP;
   TEMP = ...;
   A VAR1;
   VAR1 = operation using TEMP;
   A VAR2;
   VAR2 = operation using TEMP;
   ```

3) Very complex “load” programs can be difficult for a Dictionary Constructor to read. This methodology is basically for straightforward loads where one observation input from a flat file results in one observation being output into a SAS data set.

REFERENCES


SAS and SAS/FSP are registered trademarks of SAS Institute, Inc., Cary, N.C.

ACKNOWLEDGEMENTS

Thanks to Tina Hom of American Express, TRS in New York for her invaluable assistance in developing the Cardmember Information Management dictionary constructor.