

Data Dictionary Approaches to Edit and Validate Survey Data

Naoko S. Stearns, Research Triangle Institute; RTP, NC

Abstract

The process of validating and editing survey data becomes crucial to an organization's reputation when the intended user is the general public. Also, these tasks require substantial resources, including experienced programmers and data analysts. Fortunately, using a data dictionary can facilitate that effort in two ways: by producing documentation about the data and by generating SAS[®] code that precisely represents the data. The implementation of a data dictionary and the associated code generators automates many of the mundane and error-prone data cleaning tasks, such as, assigning special missing codes and producing deliverable ASCII files. This paper discusses a method of using a Microsoft Excel[®] data dictionary that is dynamically accessed via Dynamic Data Exchange (DDE) for the purpose of efficiently editing and validating survey data.

Introduction

The motivation for using a data dictionary is to make the data cleaning and validation processes efficient and consistent. Figure 1 shows the functionality of a dictionary-driven system in the survey data cleaning process. A data dictionary, as a central repository of data about the data, facilitates the edit process. You can use the dictionary to produce useful documentation, enforce data consistency and define edit rules.

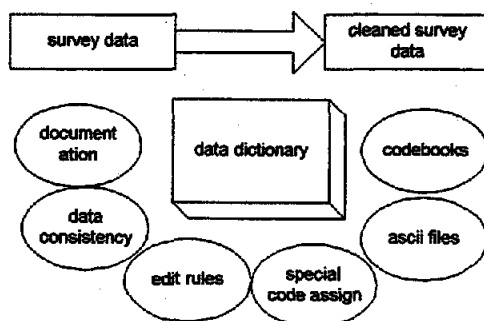


Figure 1:

As a simple example, labels are maintained in the dictionary, not hard coded in the program itself. Many data cleaning routines such as the special missing code assignment and ASCII file

productions are automated by writing code generators that use a dictionary as input, to generate programs which perform these routines. Finally, codebooks can incorporate the information defined in the dictionary.

This paper's emphasis is on the utility of data dictionary itself, whether the dictionary is in Excel or Microsoft Access[®] tables, dynamically read via DDE, ODBC or in a non-dynamic setting. Regardless of the specific method, a data dictionary provides a systematic way of documenting as well as accurately producing any deliverable file.

Dictionary Tables

You can take advantage of the SAS Dictionary Tables, a facility available in the version 6.07 or higher, to extract data attributes (e.g., variable name, data type, storage length, etc.) of existing SAS data sets to build a dictionary. The SQL Procedure allows you to extract data attributes of existing SAS data sets from the columns table. You can then import selected variable attributes into your chosen Windows applications such as Excel spreadsheet or Microsoft Access tables, to maintain and update. Once essential information has been extracted from the Dictionary Tables, the project dictionary may be developed by adding more fields to make it useful.

An alternative to using the SAS provided Dictionary Tables, in the absence of available SAS data sets, is to obtain similar variable attributes from survey collection instruments. The survey collection instruments often produce ASCII files containing variable names, labels and formats that can be used to populate the dictionary.

Dynamically Accessing Data Dictionary

The below is an example to DDE access to the Excel dictionary.

```
libname dict "c:\survey\dict";
filename xlsdict DDE
'Excel|c:\survey[dict.xls]dict!R2C1:
R540C74';
data dict.dict;
    length itemno $6 tvar $8 type $1
```

```

len $2 valwid $4 label $40
comments $200
-- More data -- ;
infile xlsdict dlm='09'x notab
      dsd missover;
input itemno tvar type len
valwid label comments
-- More data -- ;
run;

/* invoke code generator macros */

```

The FILENAME statement defines the *fileref* *xlsdict*, DDE (a special device-type keyword) and the DDE triplet. This DDE triplet consists of the server application name (e.g., Excel), the full path of the spreadsheet name (e.g., c:\survey\dict.xls\dict), and the range of the data (e.g., R2C1:R540C74), delimited by the special characters | and !. The INFILE statement indicates that the data delimiter inserted by Excel is a tab (DLM='09'x where '09'x is the ASCII hexadecimal representation of a tab character). Also, the NOTAB option prevents SAS from converting tabs into blanks so that embedded blanks in a text field are considered as blanks, not delimiters. Finally, the DSD option instructs SAS to treat two consecutive tabs as a missing value.

Once the dictionary is read into SAS from Excel upon updates to the dictionary, the subsequent invocation of code generator macros rebuilds reports and programs.

The Setup

To illustrate, consider the dictionary used for a large transportation survey at Research Triangle Institute. This survey consisted of several types of questionnaire (e.g., household, person, vehicle, trip, and etc.), and resulted in several related but separate deliverable files. In addition to the usual data attributes, we added flag fields indicating which deliverable files included which variables. Other useful fields to include in a dictionary are a notes field (a brief description of how the variable was edited and other relevant comments), the value width, and questionnaire ID number, to name a few.

The following section discusses the special missing code assignment, which can be time-consuming without a dictionary and the code generator. Other functions of the dictionary-driven process are only briefly discussed in the last section.

Special Missing Code Assignment

The special missing code assignment is one of the final edits that programmers perform on deliverable files. The purpose of assigning the special missing values, typically a variation of 9-filled values (e.g., 994,998 and 999), is to differentiate among reasons why a response was missing. For example, a missing response can be due to refusal to answer by a respondent or legitimately skipped because the question was not applicable (e.g., means of transportation to work for a respondent who was too young to drive or work).

The value width of the missing code depends on the actual size of the data element. A numeric Age variable may be given a value of 994, 998 or 999, to differentiate from a large but valid age value, such as, 98. One-byte yes/no response (coded as 1/0) may be given a value of 94, 98, or 99, or simply 4,8, or 9. The rules of the special missing code vary from a survey to survey, but they need to be consistent across multiple files produced from a single survey. This is where the dictionary-driven system comes into play.

Using the SAS special missing values (the letters A through Z) is not always practical because they are for numeric variables only and deliverable files are prepared for non-SAS system users as well as SAS system users.

The following is an example of how to generate the special missing code look-ups from the dictionary so that a programmer does not need to know the exact patterns or the size of the values as he edits.

The %sclookup Macro

First, three formats are prepared that map the value width of variables to appropriate 9-filled values shown below, as skip value (xx4), don't-know value (xx8), and the refusal value (xx9).

```

proc format;
  value $skipval
    2 = '94'
    3 = '994'
    4 = '9994'
    5 = '99994'
    6 = '999994'
  ;
  value $nkval
    2 = '98'
    3 = '998'
    4 = '9998'
    5 = '99998'

```

```

        6 = '999998'
;
        value $refsval
        2 = '99'
        3 = '999'
        4 = '9999'
        5 = '99999'
        6 = '999999'
;
run;

```

The following macro %sclookup prepares the data to build the look-up tables of all variables defined in the dictionary. In the first data step, the program reads the variable identifier, value width and the data type to create skipval, nkval, and refsval. These three variables store the correct patterns of special missing codes. Then, the data are separated into numeric variable data numvars and character variable data chavars.

```

%macro sclookup;
    data numvars chavars;
        set dict.dict (keep=tvar
                        valwid type);
        skipval=put (valwid, $skipval.);
        nkval  =put (valwid, $nkval.);
        refsval=put (valwid, $refsval.);

        if type='N' then
            output numvars;
        else if type='C' then
            output chavars;
run;

```

The %mkformat Macro

The next step creates the formats and informats that convert the variable names to the appropriate special missing values. The %mkformat macro nested within the %sclookup macro performs this task via the input control data sets. To prepare an input control data set, the required variables, START, LABEL, TYPE and FMTNAME are created. The variable identifier is renamed START. The macro parameter missvar resolves to the special missing code variable (skipval, nkval or refsval), and becomes LABEL. The numvars data set is given the format type of TYPE=I, preparing for the numeric informat. Similarly, the chavars data set is given TYPE=C, preparing for the character format. The RETAIN statement stores the TYPE and FMTNAME information. Finally, the CNTLIN= option of the FORMAT procedure converts the input control data set into the character format or the numeric informat. Since this survey file requires three special missing codes, the %mkformat macro invocations are present for formats and informats, three times

each. This process efficiently builds the required look-ups of all the variables defined in the dictionary without having to specify the VALUE and INVALUE statements in the FORMAT procedure.

```

%macro mkformat
    (indsn, fname, missvar);
    data &fname.;
        set &indsn. (rename=(tvar=start
                             &missvar.=label));
        %if "%upcase(&indsn.)" eq
            "NUMVARS" %then %do;
            %let fmttype=I;
        %end;
        %else %if "%upcase(&indsn.)" eq
            "CHAVARS" %then %do;
            %let fmttype=C;
        %end;
        retain type "&fmttype."
                 fmtname "&fname.";
run;

proc format cntlin=&fname.; run;
%mend mkformat;

%mkformat(numvars, nskip, skipval);
%mkformat(numvars, nnotasc, nkval);
%mkformat(numvars, nrefuse, refsval);

%mkformat(chavars, cskip, skipval);
%mkformat(chavars, cnotasc, nkval);
%mkformat(chavars, crefuse, refsval);

proc format fmtlib; run;
%mend sclookup;

```

How the Look-ups are Used

The previous section illustrated how the look-ups were generated from the dictionary. Below is an example of how the look-ups are used to assign the skip values.

```

%macro legskip(type, nvars, vars);
    %do i=1 %to &nvars.;
        %let var=%upcase
            (%scan(&vars., &i.));
        %if "%upcase(&type.)" eq "N"
            %then %do;
            &var.=input("&var.", nskip.);
            %end;
        %else %if "%upcase(&type.)" eq "C"
            %then %do;
            &var.=put("&var.", $cskip.);
            %end;
        %end;
    %mend legskip;

```

The %DO group processes any number of variables that are specified in the parameter nvars. The %scan function looks at the

argument &vars. which resolves to a list of variables and returns the *i*th word as the current variable to be assigned. If the variable being edited is numeric, then the INPUT function uses the numeric informat to return the numeric skip value (e.g., 94 or 994). If the edited variable is a character, the PUT function uses the character format to assign the character skip value (e.g., '94' or '994').

Thus, a programmer can invoke the %legskip macro as shown without knowing the exact pattern of the values that he edits:

```
Data new;
  Set old;
  If age lt 17 then do;
    %legskip(n,3,car1 car2 car3);
    %legskip(c,2,work1 work2);
  end;
```

The first macro invocation assigns the skip values of three numeric variables if the respondent is less than 17 years of age. The second invocation handles the assignment of character variables for the same age condition. Obviously, this is a simplified scenario that can be easily hard coded. However, in a practical situation, this macro allows the programmer to assign the skip values of hundreds of variables by simply giving the variable data type, the number of variables and the variable list. This data cleaning task also used a similar macro to make the global change of don't-know and refusal values (-1 and -2) entered by the data collection instrument into the appropriate 9-filled values.

Other Functionality of Dictionary System

Other functionality of the dictionary system includes the following:

Documentation – A data dictionary itself, aside from the deliverable survey files, is a useful document. Numerous reports about the target variables can be generated well before the data are ready, which gives analysts time to plan how to avoid conflicting names, data types, editing rules and etc.

Data Conversion and Edit Rules – Data conversion and edit rules can be defined as a SAS statement or for more involved code as macro programs in the dictionary.

Data Consistency – Label, Length and Keep statements can be generated from the dictionary so that the consistency of the deliverable files is enforced.

ASCII File Production – The ASCII file production is often a part of the deliverable requirements for those not using the SAS system. A code generator that specifically does this task of any number variables and data sets in the dictionary will save programmer's time to write numerous discrete programs.

Codebooks – You can integrate the data dictionary with the codebook to make the codebook report comprehensive as a summary of the edited data.

Conclusion

A dictionary-driven data editing and validation process described above lays out a reliable and efficient infrastructure to accomplish required data cleaning tasks. In a dictionary-driven environment, programmers would focus on writing fewer numbers of code generators that produce code and programs based on the dictionary definitions, rather than writing many discrete programs which are used only once. The special missing code assignment illustrates an efficient use of the dictionary to build look-up tables to minimize tedious and error-prone hard coding. The dictionary serves as a useful project document and facilitates the development of code generators.

SAS is a registered trademark 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.

References

Frank C. Dilorio, Nancy J. Michal (1995) "Data About Data: An Introduction to Dictionary Tables," *Proceedings of the Twenty-First Annual SAS Group International Conference*, 21, 213-219.

SAS Companion for the Microsoft Windows Environment, Version 6, Second Edition, Cary, NC: SAS Institute Inc.

Author Information

Naoko S. Stearns
Research Triangle Institute
Research Triangle Park, NC 27709-2194
919.541.7369
Stearns@rti.org