PharmaSUG 2022 - Paper AP-108

From Codelists to Format Library

Nancy Brucken, IQVIA

ABSTRACT

Codelist files in the pharmaceutical industry are designed to enumerate all possible values for a given variable. One common method for capturing and storing codelist information is to use the Microsoft® Excel® template generated by Pinnacle 21 Enterprise or Community, since that file is often used as the basis for analysis dataset specifications. From there, it's a short task to read the codelists from the file and convert them to SAS® formats and informats.

INTRODUCTION

The desire to automate programming using metadata is not limited to the pharmaceutical industry, and SAS provides many ways to drive the generation of code and variable attributes without the need to write repetitive code. This paper outlines one approach for reading the codelists from an Excel file and using them to build a catalog of formats and informats, which can then be employed to automate the creation of secondary ADaM numeric variables.

WHAT DOES A CODELIST FILE LOOK LIKE?

Pinnacle 21 Enterprise will generate a default, blank Excel specifications file as a menu option; this file can be completed with details for an individual study, and then used to generate the Define-XML file for the study. Pinnacle 21 Community cannot generate a blank specifications template, but an Excel specifications file can be produced from either a set of datasets in SAS version 5 transport format, or from an existing Define-XML. In either case, the layout of the resulting specifications template is similar. There are separate worksheet tabs for dataset metadata, variable metadata, value-level metadata and codelists.

Figure 1 shows an excerpt from a typical codelists tab:

	A	B	C	D	E	F	G	H
1	ID	Name	NCI Codelist Code	Data Type	Order	Term	NCI Term Code	Decoded Value
87	AESEV	AESEV	C66769	text		MILD	C41338	
88	AESEV	AESEV	C66769	text		MODERATE	C41339	
89	AESEV	AESEV	C66769	text		SEVERE	C41340	
90	ASEV	ASEV		text	1	Mild		
91	ASEV	ASEV		text	2	Moderate		
92	ASEV	ASEV		text	3	Severe		
93	ASEVN	ASEVN		integer		1		Mild
94	ASEVN	ASEVN		integer		2		Moderate
95	ASEVN	ASEVN		integer		3		Severe
85	TRT	TRT		text	1	Drug A		
86	TRT	TRT		text	2	Drug B		
87	TRT	TRT		text	3	Drug C		
98	TRTN	TRTN		text		1		Drug A
99	TRTN	TRTN		text		2		Drug B
00	TRTN	TRTN		text		3		Drug C

Figure 1. Sample codelist file

In this example, the codelists labeled as ASEV and TRT are associated with the primary character variables ASEV and TRT, and the codelists labeled as ASEVN and TRTN are associated with the secondary numeric variables ASEVN and TRTN. Key pieces of information are found in the following columns:

- ID: Codelist identifier used to link the codelist with its associated variable(s)
- Name: The codelist name, which can be any text string
- Term: Contains all possible values of the primary variable; for a secondary numeric variable, it contains the coded values
- Decoded Value: Populated for secondary numeric variables, and should match the values in the Term column of the corresponding primary variable.

The other two columns in the spreadsheet, NCI Codelist Code and NCI Term Code, are not used in this process, but are required to be populated if the codelist and associated terms have been pre-defined by CDISC.

READING THE CODELIST FILE

There are many different ways to read in Excel spreadsheets, depending on which SAS modules you have licensed. One of the easiest, provided you have SAS/ACCESS to PC File Formats available, is to simply use the Excel LIBNAME engine, and read the codelist file directly. With this method, each worksheet tab in the Excel file becomes a separate SAS dataset. The following code was used to read in the sample codelist file:

```
%LET INPUT_SPECS_FILE = /home/MWSUG2021/Sample ADaM Mapping specs.xlsx;

libname inspecs xlsx "&input_specs_file";
options validvarname=v7;

data codelists;
  set inspecs.codelists;
  where id in ('ASEV', 'ASEVN', 'TRT', 'TRTN'); /* For this example */
run;

libname inspecs clear;
```

SAS Program 1: Import specifications file

Note that for the purpose of this example, the codelists file is subset to only a few entries as illustration.

The system option VALIDVARNAME controls the types of characters allowed in SAS variable names. Setting that to V7 allows for variable names up to 32 alphanumeric characters long, changes any spaces or invalid characters in the Excel column headings to underscores, and appends a counter to the variable name if required for uniqueness. As a result, the spaces in the Excel column headings are converted to underscores. The Excel libref should be released after the program has finished reading from it; otherwise, it will remain open.

The resulting example CODELISTS dataset can be seen in Figure 2:

	ID	Name	NCI_Codelist_Code	Data_Type	Order	Term	NCI_Term_Code	Decoded_Value
1	ASEV	ASEV		text	1	Mild		
2	ASEV	ASEV		text	2	Moderate		
3	ASEV	ASEV		text	3	Severe		
4	ASEVN	ASEVN		integer		1		Mild
5	ASEVN	ASEVN		integer		2		Moderate
6	ASEVN	ASEVN		integer		3		Severe
7	TRT	TRT		text	1	Drug A		
8	TRT	TRT		text	2	Drug B		
9	TRT	TRT		text	3	Drug C		
10	TRTN	TRTN		text		1		Drug A
11	TRTN	TRTN		text		2		Drug B
12	TRTN	TRTN		text		3		Drug C

Figure 2. CODELISTS dataset

CONVERTING TO FORMATS AND INFORMATS

The first step in creating the format library is to create the input control dataset which will be used by PROC FORMAT to generate the actual formats and informats. There are 3 required variables in an input control dataset:

- **FMTNAME** = character variable containing the name of the format/informat to be created.
- **START** = character variable containing the starting value of the formatted range. In this case, the formatted range consists of a single value, so START points to the value to be formatted.
- **LABEL** = variable containing the decoded value.

In our example, we'll use NAME to represent the format name. For a format, TERM from the spreadsheet contains the coded value, which is used to populate START in the control dataset, and DECODED_VALUE contains the corresponding character value, which is used to populate LABEL. However, for an informat, those variables are reversed. In that case, DECODED_VALUE from the spreadsheet contains the original character value in the primary variable, which goes into START, and TERM contains the code to be stored in the secondary variable, which goes into LABEL.

In addition, the optional TYPE variable in an input control dataset indicates whether a format or an informat should be created, as well as the type (numeric or character) of the formatted result. A complete set of possible values for TYPE can be found in the SAS FORMAT procedure documentation. However, since we only need to create numeric formats and informats, we will only use TYPE='I' to generate a numeric format, and TYPE='N' to produce a numeric informat. Note that these values are not case-sensitive. In addition, since we have explicitly defined the type of format and informat by specifying TYPE, we do not need to preface character format names with a '$'.

The following code creates an input control dataset from our codelist:

```
data fmt_in (keep=fmtname type start label);
  set codelists (where=(decoded_value is not missing));

  *** Formats;
  type = 'n';
  fmtname = name;
  start = term;
  label = decoded_value;
  output;


  *** Informats;
  type = 'i';
  fmtname = name;
  start = decoded_value;
  label = term;
  output;

run;
```

SAS Program 2: Creation of input control data set

The WHERE clause further subsets to those codelist entries containing coded and decoded values, since those are the only records containing all of the information required for building formats and informats. As you can see, both formats and informats can be created in the same DATA step.

Figure 3 shows the input control dataset that is built with SAS Program 2, sorted by type and format so that it is easier to read.

	type	fmtname	start	label
1	i	ASEVN	Mild	1
2	i	ASEVN	Moderate	2
3	i	ASEVN	Severe	3
4	i	TRTN	Drug A	1
5	i	TRTN	Drug B	2
6	i	TRTN	Drug C	3
7	n	ASEVN	1	Mild
8	n	ASEVN	2	Moderate
9	n	ASEVN	3	Severe
10	n	TRTN	1	Drug A
11	n	TRTN	2	Drug B
12	n	TRTN	3	Drug C

Figure 3. Input control dataset

The next step is to actually import the input control dataset into a format catalog via PROC FORMAT. The CNTLIN= option on PROC FORMAT statement allows you to do that, and will accept a dataset that contains multiple formats and informats. However, they must be grouped together, so the dataset that we just created needs to be sorted by TYPE before it can be imported. Otherwise, only the last term for each format type and name is included. Also, overlapping ranges are not allowed in an input control dataset, so ensure that there are no duplicate records in the codelist file.

The following code builds the format library from the input control dataset:

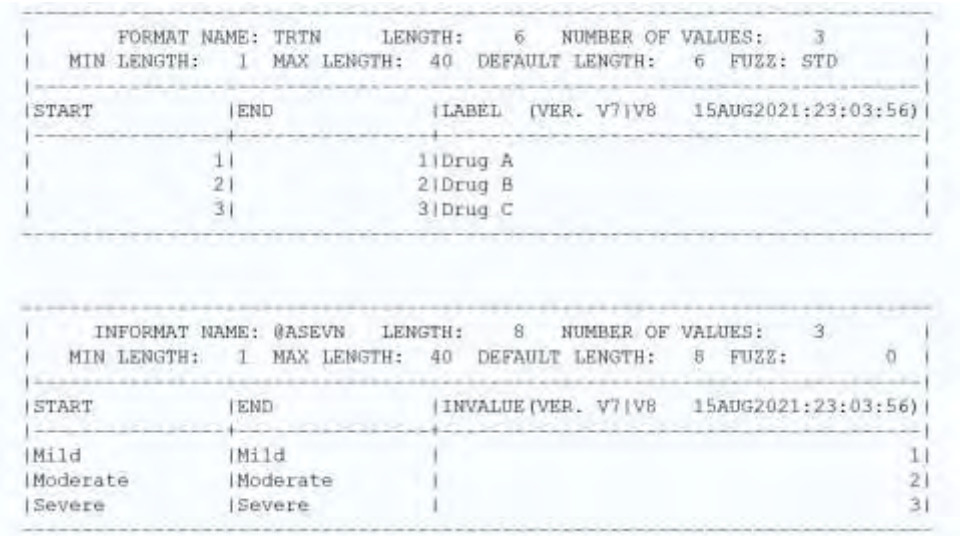
```
proc sort data=fmt_in;
  by type fmtname;
run;

proc format fmtlib library=work cntlin=fmt_in;
run;
```

SAS Program 3: Building the format library

The FMTLIB option prints out a list of all of the formats and informats contained in the work library, so that you can verify that everything was created as expected.

A partial screenshot showing one of the formats and one of the informats generated from our codelists can be seen in Figure 4.



```
FORMAT NAME: TRTN     LENGTH:    6   NUMBER OF VALUES:    3
MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH:   6  FUZZ: STD
START           END             LABEL  (VER. V7|V8   15AUG2021:23:03:56)
              1               1Drug A
              2               2Drug B
              3               3Drug C

INFORMAT NAME: @ASEVN   LENGTH:    8   NUMBER OF VALUES:    3
MIN LENGTH:   1  MAX LENGTH:  40  DEFAULT LENGTH:   8  FUZZ:        0
START           END             INVALUE(VER. V7|V8   15AUG2021:23:03:56)
Mild            Mild                                              1
Moderate        Moderate                                          2
Severe          Severe                                            3
```

Figure 4. Format library contents

CREATING SECONDARY VARIABLES

Now that the format library has been created, the informats it contains can be used to create the secondary variables in code-decode variable pairs. For example, the following statement will create the secondary numeric variable ASEVN from the primary character variable ASEV:

```
asevn = input(asev, asevn.);
```

A macro can easily be written to automatically generate secondary numeric variables from a given list of primary character variables, provided appropriate naming conventions for both variables and formats have been adopted.

CONCLUSION

Using codelists to create a catalog of formats and informats is one step along the journey to metadata-driven dataset creation. It's an easy step to take, though, using the techniques outlined in this paper, and can help minimize the impact of last-minute changes to variable values on your dataset and table, listing and figure (TLF) programs.

REFERENCES

SAS 9.4 and SAS Viya 3.5 Programming Documentation. "Results: FORMAT Procedure". Accessed August 15, 2021.
https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/proc/p0owa4ftikc2ekn1q0rmpulg86cx.htm

Wright, Wendi L. 2007. "Creating a Format from Raw Data or a SAS Data Set." *SAS Global Forum 2007*. Accessed September 21, 2021.
https://support.sas.com/resources/papers/proceedings/proceedings/forum2007/068-2007.pdf

ACKNOWLEDGMENTS

Many thanks to Richann Watson and Paul Slagle for reviewing this paper and providing input.

CONTACT INFORMATION

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

Nancy Brucken
IQVIA
nancy.brucken@iqvia.com