Creating a data dictionary using Base SAS®

Tasha Chapman, Oregon Department of Consumer and Business Services

Base SAS contains a number of procedures and options that make documentation of metadata easier than ever. This paper will discuss one method for creating a data dictionary using Base SAS, which will cover PROC CONTENTS, the Output Delivery System (including ODS EXCEL, ODS TRACE, ODS SELECT, and ODS Output), and PROC DATASETS (including Extended Attributes).

(Extended Attributes in particular are very cool and not to be missed.)

Document. Document. Document. How often are we told that we need to do a better job of documenting our work? Documentation is key for any business' sustainability and succession planning. It is also some of the most tedious and often forgotten work in every organization. In an effort to make this job easier, this paper will discuss some of the valuable tools available in Base SAS to help assist in documenting data tables.

One way to document data sets is through a data dictionary. A data dictionary provides information about the fields in a data table and their key attributes, such as field name, data type, length, etc. Luckily, Base SAS has a procedure already in place that can provide these basic elements for you – PROC CONTENTS.

PROC CONTENTS – The Variable List

PROC CONTENTS is a handy procedure in Base SAS that can display the metadata (also known as the "descriptor portion") of any SAS data set. For example, let's take one of the existing data sets that come with the SAS package, SASHelp.Heart. If we run the PROC CONTENTS code below:

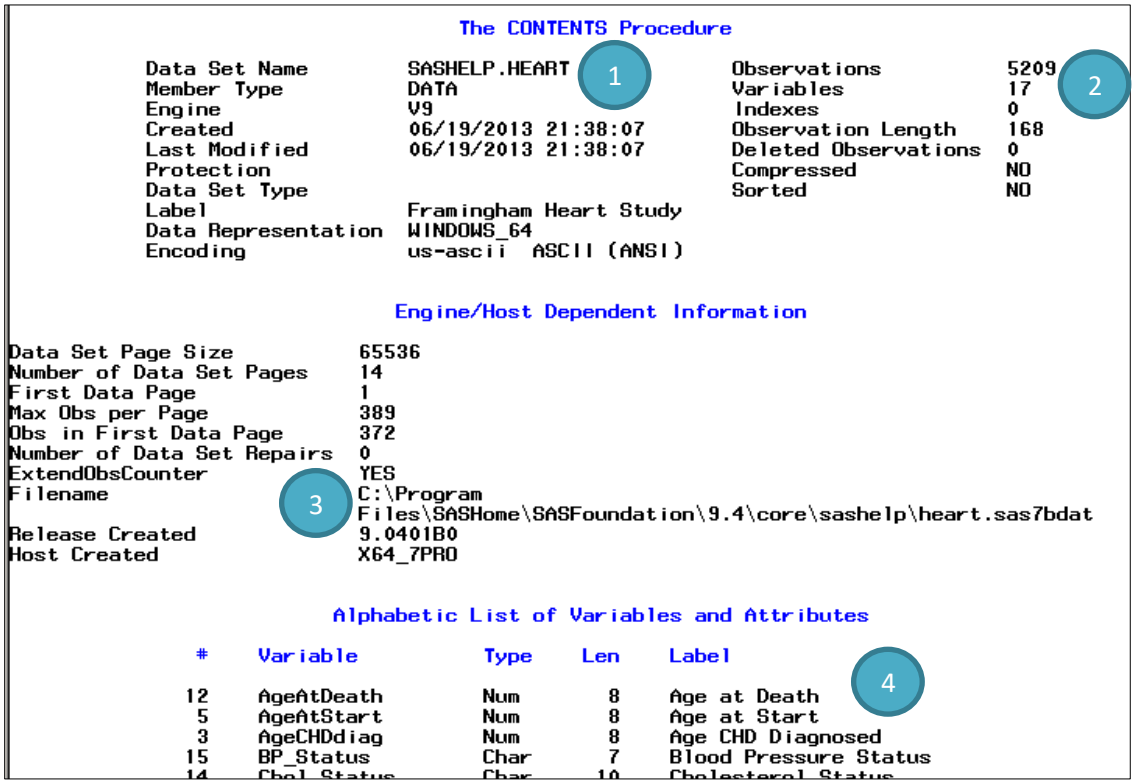
```
proc contents data=sashelp.heart; run;
```

The output we get will look like Figure 1 below. There is a lot of information here, but a few items worth highlighting:

1. The name of the data set and the library in which the data set lives. (In this case, the data set is named "Heart" and the library reference name is "SASHelp")
2. The number of observations and variables in the data set
3. The actual file location where the data set lives.
4. The variable list, which includes the list of the fields in the data set and the attributes assigned to those fields.

For the purposes of this paper, item #4 is of most importance. The variable list in PROC CONTENTS contains many of the pieces of information that we need for a data dictionary. Using the Output Delivery System (ODS) we can pull out this portion of the output, move it to an Excel document (or whatever destination we choose), and start our data dictionary.

Figure 1. PROC CONTENTS of SAShelp.Heart



The CONTENTS Procedure

Data Set Name	SASHELP.HEART	Observations	5209
Member Type	DATA	Variables	17
Engine	V9	Indexes	0
Created	06/19/2013 21:38:07	Observation Length	168
Last Modified	06/19/2013 21:38:07	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label	Framingham Heart Study		
Data Representation	WINDOWS_64		
Encoding	us-ascii ASCII (ANSI)		

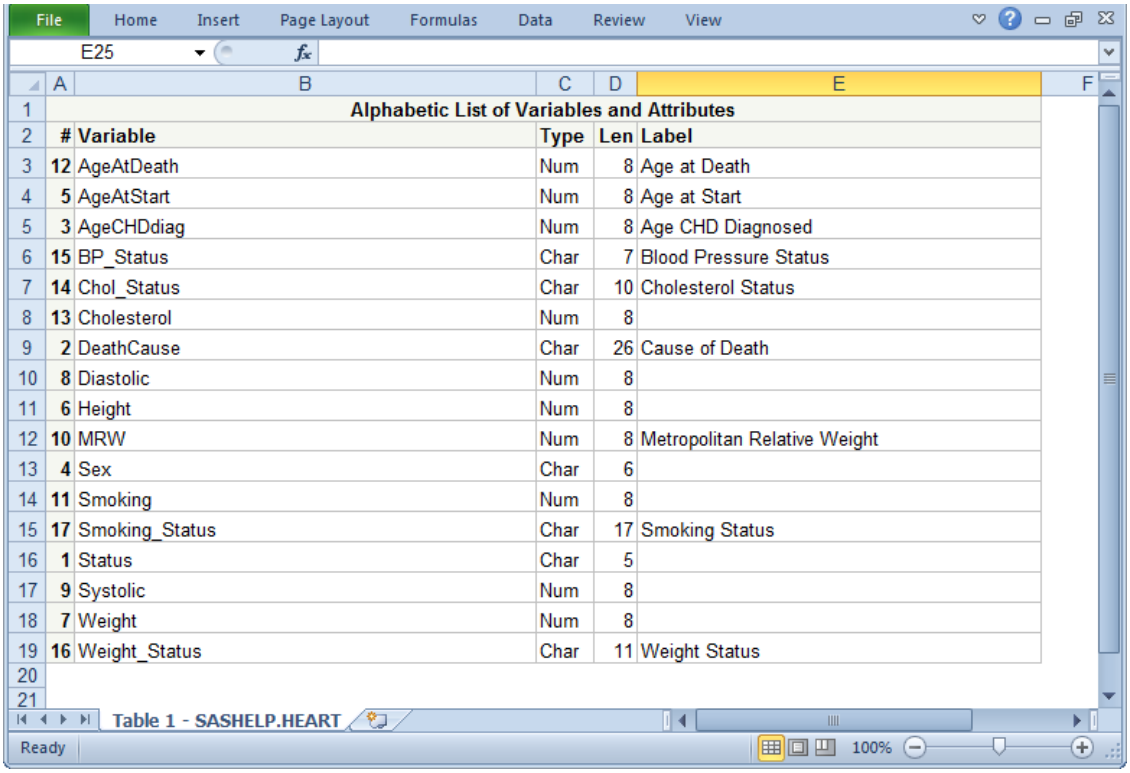
Engine/Host Dependent Information

Data Set Page Size	65536
Number of Data Set Pages	14
First Data Page	1
Max Obs per Page	389
Obs in First Data Page	372
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	C:\Program Files\SASHome\SASFoundation\9.4\core\sashelp\heart.sas7bdat
Release Created	9.0401B0
Host Created	X64_7PRO

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Label
12	AgeAtDeath	Num	8	Age at Death
5	AgeAtStart	Num	8	Age at Start
3	AgeCHDdiag	Num	8	Age CHD Diagnosed
15	BP_Status	Char	7	Blood Pressure Status
14	Chol_Status	Char	10	Cholesterol Status

Figure 2. Variable list (shown in Excel)



Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Label
12	AgeAtDeath	Num	8	Age at Death
5	AgeAtStart	Num	8	Age at Start
3	AgeCHDdiag	Num	8	Age CHD Diagnosed
15	BP_Status	Char	7	Blood Pressure Status
14	Chol_Status	Char	10	Cholesterol Status
13	Cholesterol	Num	8	
2	DeathCause	Char	26	Cause of Death
8	Diastolic	Num	8	
6	Height	Num	8	
10	MRW	Num	8	Metropolitan Relative Weight
4	Sex	Char	6	
11	Smoking	Num	8	
17	Smoking_Status	Char	17	Smoking Status
1	Status	Char	5	
9	Systolic	Num	8	
7	Weight	Num	8	
16	Weight_Status	Char	11	Weight Status

The Output Delivery System – ODS Excel

The Output Delivery System in Base SAS allows you to create report output that can be viewed by many popular software applications, including PDF, Excel, Word, and HTML. For the purposes of this paper we are going to use the ODS Excel[1] to send our report output to an Excel spreadsheet. Adding ODS to our SAS program is very simple. We simply wrap the "ODS Sandwich," as it is colloquially called, around our output producing code. The first statement opens the ODS destination, and the last statement closes it. Every bit of output produced between the open and close statements will be sent to our chosen destination. For example:

```
ods Excel file="C:\SAS\Data Dictionary.xlsx" style=statistical;

proc contents data=sashelp.heart; run;

ods Excel close;
```

The ODS open statement in the above example contains the following elements:

- `ods Excel` – The ODS destination for Excel uses Microsoft Open Office XML Format and creates Microsoft spreadsheetML XML. In other words, it creates an Excel spreadsheet that can be opened using Microsoft Excel 2010 or later.
- `file="C:\SAS\Data Dictionary.xlsx"` – Here we indicate the location and filename of the document we intend to create.
- `style=statistical` – This is an optional command that will apply a style template to our file. "Statistical" is one of the many styles that come standard with Base SAS. This style template will affect the formatting of our document, such as fonts and color selections.

The code above will create an Excel document similar to what is seen in Figure 2. However, our output will actually have three sheets, which correspond with the three parts of the PROC CONTENTS standard output[2]. The last sheet, which contains the variable list, is the part we are most concerned with. If we choose we can limit the output of PROC CONTENTS to only the portion(s) we need. We do this using a few additional tools in the Output Delivery System toolbox – ODS TRACE and ODS SELECT.

The Output Delivery System – ODS TRACE and ODS SELECT

Many procedures create multiple pieces of output. For example, PROC CONTENTS actually creates three output objects: the data set attributes, the engine/host information, and the variable list. When we used ODS Excel above, each of these three objects was output on its own spreadsheet. However, these objects can each be called and selected by name, giving us the flexibility to pick and choose only the pieces we need. To do that, we first need to know how to reference these objects.

ODS TRACE will write a record of each output object that is created to the SAS log. To use ODS TRACE, we simply need to turn it on before running any other output producing statements:

---

[1] Many platforms of SAS, such as Enterprise Guide, may automatically produce Excel output based on system settings. This is still generated using ODS Excel. However, for the purposes of this paper we are going to use explicit ODS code in our program to illustrate some of the extensive features of ODS, which are more easily controlled using code. If you are using ODS code in your program while system settings are also set-up to produce Excel output, you may see two versions of Excel output in your Results viewer.

[2] The objects produced by PROC CONTENTS may vary. For example, if your data set is sorted, PROC CONTENTS may produce a "SortedBy" object which contains information related to sorting.

```
ods trace on;

ods Excel file="C:\SAS\Data Dictionary.xlsx" style=statistical;

proc contents data=sashelp.heart; run;

ods Excel close;
```

Figure 3. Output objects of PROC CONTENTS

```
55   proc contents data=sashelp.heart; run;


Output Added:
-------------
Name:        Attributes
Label:       Attributes
Template:    Base.Contents.Attributes
Path:        Contents.DataSet.Attributes
-------------

Output Added:
-------------
Name:        EngineHost
Label:       Engine/Host Information
Template:    Base.Contents.EngineHost
Path:        Contents.DataSet.EngineHost
-------------

Output Added:
-------------
Name:        Variables
Label:       Variables
Template:    Base.Contents.Variables
Path:        Contents.DataSet.Variables
-------------
NOTE: PROCEDURE CONTENTS used (Total process time):
      real time           0.10 seconds
      cpu time            0.09 seconds
```

Now that ODS TRACE is turned on, our log contains additional information about each of the output objects produced by PROC CONTENTS. The third object – Variables – is the list of variables in our data set.

We can use the information provided by ODS TRACE to specifically choose which objects we would like to output using the ODS SELECT statement. For example[3]:

```
ods select variables;

ods Excel file="C:\SAS\Data Dictionary.xlsx" style=statistical;

proc contents data=sashelp.heart; run;

ods Excel close;
```

Using the code from the above example, only the variable list will be output to our Excel file, as shown in Figure 2. The remaining output objects will be suppressed.

[3] Now that we know which objects we want to keep, the ODS TRACE statement from the previous example is no longer relevant. It wouldn't hurt to keep it in our program, but it won't be displayed it in any further examples.

If you look at the variable list in Figure 2, you'll notice that there are a number of attributes that are listed for each variable. The attributes displayed by PROC CONTENTS may vary depending on nature of your data set. Standard attributes worth including in a data dictionary might include:

- **Variable position** – This is labeled as "#" in Figure 2 and is also sometimes referred to as "num" or "varnum". This represents the order in which the variable resides in the data set.
- **Name** – This is the name that identifies the variable. Variable names are not case sensitive, but must conform to SAS naming rules.
- **Type** – This identifies whether the variable is character or numeric. SAS data sets do not support any other variable types.
- **Length** – This is the number of bytes used to store the variable's values in the data set.
- **Label** – This is any descriptive label that has been permanently assigned to the variable.
- **Format** – This refers to any format that has been permanently assigned to the variable. Similarly, if there were any permanent informats assigned to the variable they would display in a separate "Informat" column. Figure 2 does not show these columns because no variable in the Heart data set has a permanent format or informat assigned.
- **Index** – This indicates whether the variable is part of an index. Figure 2 does not show this column because none of the variables are indexed.

All of these variable attributes can be easily modified. One way to modify them is through the DATA step, which is a common method used by many new SAS programmers. However, every DATA step creates a new data set, meaning the entire data set is read into memory and copied to a new data set. This can consume a lot of computing resources unnecessarily. If we only need to make changes to the metadata, then there is a better solution – PROC DATASETS.

PROC DATASETS – The MODIFY Statement

Among other things, PROC DATASETS allows us to create and modify data set attributes for any existing SAS data set. We can create variable labels, assign permanent formats and informats, change variable names, and much more, without running a DATA step or creating a new data set.

One important note: Changes to the data set attributes made using PROC DATASETS are permanent. However, SASHelp.Heart is a standard data set that comes with your SAS package, and as with all standard items, it is usually a good idea to *not* make permanent modifications to it. Therefore, for the purposes of the examples below, we are first going to make a temporary copy of the data set in the Work library. This step obviously is not necessary if you are working with a data set that you have full ownership of and/or explicitly intend to modify permanently.

PROC DATASETS uses run-group processing, meaning we can make multiple modifications to the same data set, or even modify multiple data sets all within the same procedure call. For example, let's say we wanted to change the name of Sex (variable #4) in the Heart data set to Gender and apply a label to Weight (variable #7). The syntax would look like so:

```
proc datasets library=work;
modify heart;
      rename sex = gender;
      label weight = 'Weight in lbs';
run;
quit;
```

The DATASETS procedure in the above example includes the following elements:

- `proc datasets library=work;` – In calling the procedure we also list which library the data set(s) we want to modify resides in.
- `modify heart;` – The MODIFY statement is where we declare which data set we want to make changes to.
- `rename` & `label` – These are just a few examples of the types of modifications we can make to the data set variables. These are sub-statements that must be placed after the MODIFY statement.
- `run;` – As mentioned above, PROC DATASETS supports run-group processing, meaning we can make multiple changes to multiple data sets within the same procedure call. We first put a RUN statement at the end of the modify run group; then if we want to make changes to another data set (within the same library), we simply start another run group with a new MODIFY[4] statement.
- `quit;` – Because PROC DATASETS supports run-group processing, the procedure must explicitly be ended with a QUIT statement.

This basic list of variables and their associated attributes listed in Figure 2 may already be more documentation than most organizations keep on hand for their data tables. Utilized to their fullest, we can store a lot of information about our data table within the data set itself. That being said, what if we wanted to keep additional documentation, such as a longer description of the variable, the formula used to calculate the variable, or the name of the source table that the variable comes from? There are no standard attributes to store this information. As it turns out, we aren't limited to standard attributes. My Friend, let me introduce you to the wonderful world of extended attributes.

PROC DATASETS – Extended Attributes

Extended attributes are user defined attributes that we can attach to a data set or a variable. They are essentially customized metadata for your data set. For this paper, we are going to focus on variable extended attributes.

We can add and modify extended attributes using PROC DATASETS. For example, let's say that for Height, Weight, and Smoking we wanted to add a new attribute known as "Unit" in which we list the unit of measurement. For other variables we want to add an attribute "ValidValues" that includes a descriptive list of what the values mean. After the MODIFY statement in PROC DATASETS, we will use the XATTR SET VAR statement to add these new attributes:

[4] Although the MODIFY statement is one of the more popular features of PROC DATASETS, and the most relevant to the current paper, it is by no means the only option available in PROC DATASETS.

```
proc datasets library=work;
modify heart;
      xattr set var
            Height (Unit = "Inches")
            Weight (Unit = "Pounds")
            Smoking (Unit = "Cigarettes per day")
            DeathCause (ValidValues = "Values are Cancer, CVD, CHD,
            Other, and Unknown. Missing if Px still alive.")
            Chol_Status (ValidValues = "<200 = Desirable; 200-239 =
            Borderline; >239 = High")
            Smoking (ValidValues = "0 = does not smoke (since last
            exam)");
run;
quit;
```

The XATTR statement above contains the following elements:

- `xattr set var` – The XATTR SET statement will add an attribute if it does not exist and update an existing attribute. Alternatively, if we want to perform one action but disallow the other (for example, we want to add a new attribute, but not accidently overwrite an existing attribute), we can use the XATTR ADD or XATTR UPDATE statements instead. XATTR SET **VAR** indicates that we are assigning attributes to variables (as opposed to XATTR SET **DS** for assigning attributes to the data set).
- `Height (Unit = "Inches")` – "Height" is the variable that the attribute is being assigned to. "Unit" is the name of the new attribute. "Inches" is the value of that attribute for the Height variable.

There is no limit to the number of attributes that can be assigned to each variable. There is also no hard limit on the length of a character value for an extended attribute.

Figure 4. Extended attributes display from PROC CONTENTS

The CONTENTS Procedure

Alphabetic List of Extended Attributes on Variables

Extended Attribute	Attribute Variable	Numeric Value	Character Value
Unit	Height	.	Inches
Unit	Smoking	.	Cigarettes per day
Unit	Weight	.	Pounds
ValidValues	Chol_Status	.	<200 = Desirable; 200-239 = Borderline; >240 = High
ValidValues	DeathCause	.	Values include Cancer, CVD, CHD, Other, and Unknown. Missing if Px still alive.
ValidValues	Smoking	.	0 = does not smoke (since last exam)

Once extended attributes have been assigned, we can use PROC CONTENTS to view this new metadata attached to our data set, as seen in Figure 4. The extended attributes of the variables are actually stored as a separate output object from the variable list; the object is named ExtendedAttributesVar[5]. It sure would be nice if we could rearrange this metadata so that these extended attributes were additional columns in our variable list instead of a separate table. Luckily, we can treat this metadata

[5] Go use ODS TRACE and see for yourself. I'll wait.

like actual data once we send it to a SAS data set, and we do this using one more tool in the Output Delivery System toolbox – ODS OUTPUT.

ODS OUTPUT and PROC TRANSPOSE

Many procedures have options to send their output to a SAS data set. However, not all procedures are alike – some require OUT=, others use an OUTPUT statement – and it can be difficult to keep all that code straight. ODS OUTPUT is an easy-to-use alternative for sending output to a SAS data set. For example:

```
ods output variables=varlist;
ods output ExtendedAttributesVar=varlist2;

proc contents data=heart; run;
```

The code above will create two SAS data sets:

- **Varlist –** This is original variable list from PROC CONTENTS with the standard attributes. This data set can be seen in Figure 5.
- **Varlist2** – This is the extended attributes from PROC CONTENTS. This data set can be seen in Figure 6.

The extended attributes table from PROC CONTENTS is set-up so that each row represents each unique variable-attribute combination. However, to make this more useful for our purposes, we want a data set where each row represents a variable from the original Heart data set and each extended attribute is in a separate column. We can use PROC TRANSPOSE to change the layout of the Varlist2 data set, like so:

```
proc sort data=varlist2; by attributevariable; run;

proc transpose data=varlist2 out=varlist2B;
by attributevariable;
id extendedattribute;
var attributecharvalue;
run;
```

The TRANSPOSE procedure will produce a data set that looks like Figure 7. As you can see, there is one row for each variable, and each of our extended attributes have their own column.

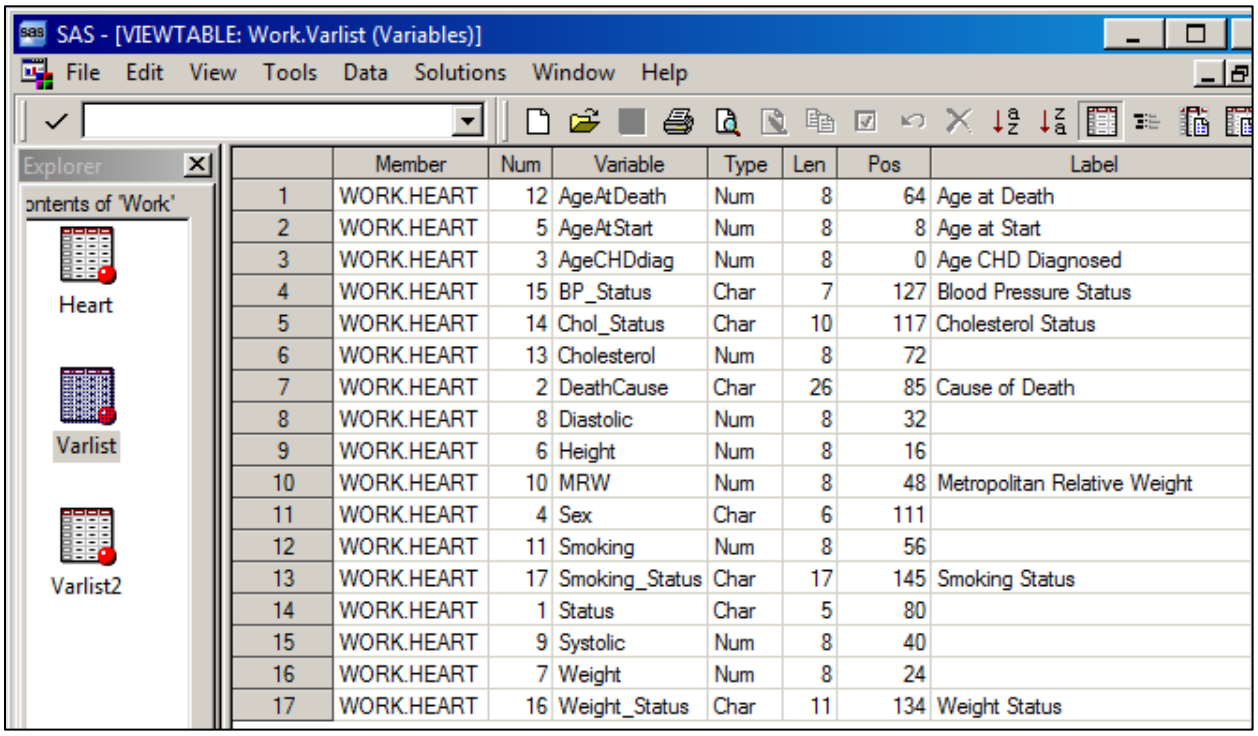
The example above works well if all of our extended attributes are character values. It is worth noting, however, that the extended attribute table from PROC CONTENTS (e.g. Varlist2) stores numeric extended attributes in a separate column. If your metadata contains a mix of numeric and character extended attributes, you will need to modify your transposition methods accordingly.

Putting it all together

Now that our metadata is officially data, we can use all of the powers of SAS to create our data dictionary however we see fit. There are many different tools available when it comes to joining and merging these data sets. Some people (such as myself) prefer PROC SQL; others prefer DATA step MERGE. Below I've provided two sets of code that can each produce the final product in Figure 8.

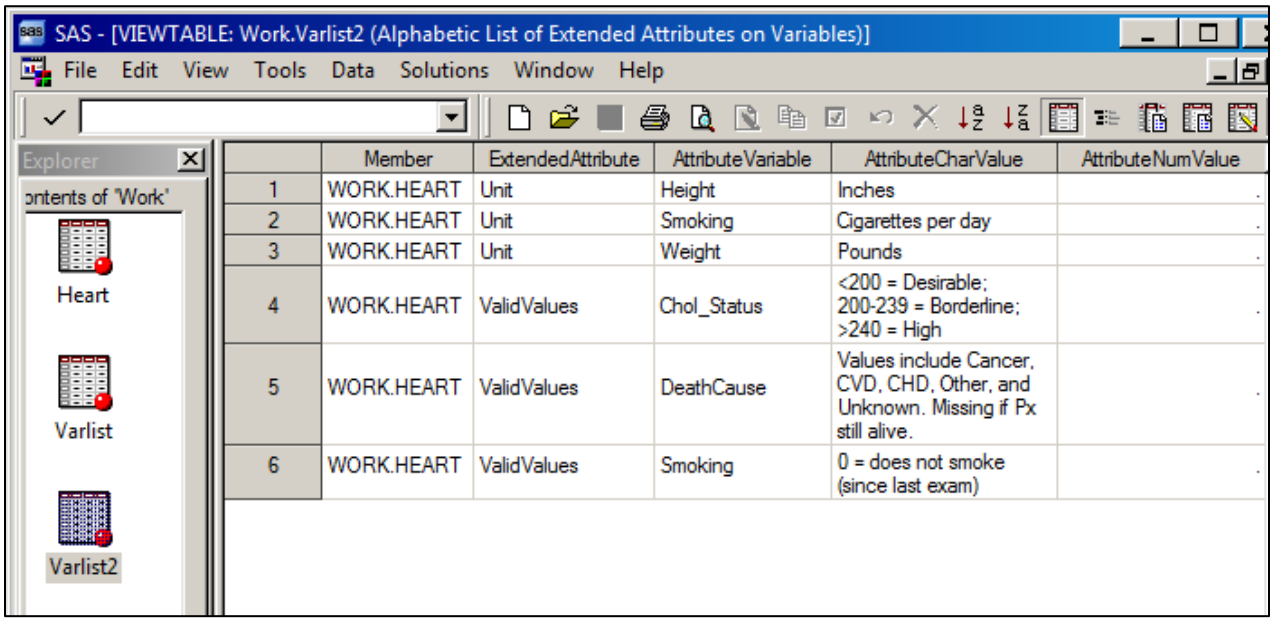
In conclusion, I hope that you have found this tutorial useful. If nothing else, you no longer have any excuse to not document your SAS data sets. Once you've built the mechanism for creating a data dictionary, you can easily modify and rebuild your data dictionary any time your data set changes. Happy documenting!

Figure 5. Varlist data set (standard attributes from PROC CONTENTS)



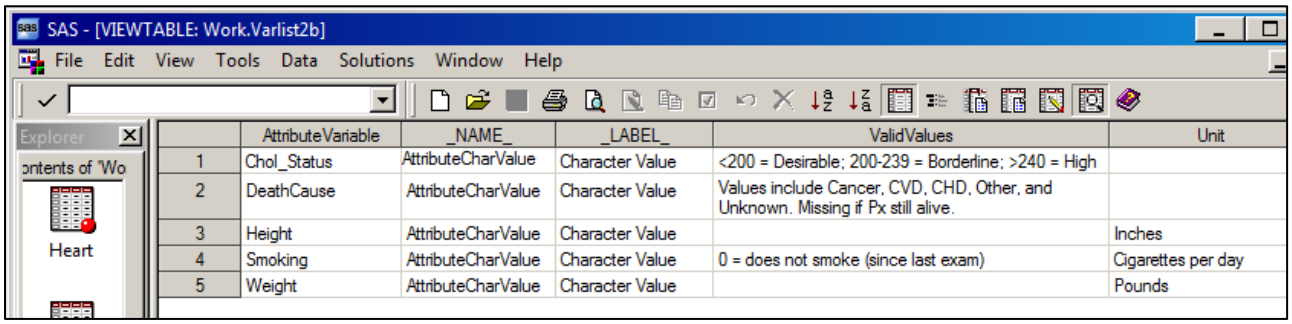
	Member	Num	Variable	Type	Len	Pos	Label
1	WORK.HEART	12	AgeAtDeath	Num	8	64	Age at Death
2	WORK.HEART	5	AgeAtStart	Num	8	8	Age at Start
3	WORK.HEART	3	AgeCHDdiag	Num	8	0	Age CHD Diagnosed
4	WORK.HEART	15	BP_Status	Char	7	127	Blood Pressure Status
5	WORK.HEART	14	Chol_Status	Char	10	117	Cholesterol Status
6	WORK.HEART	13	Cholesterol	Num	8	72	
7	WORK.HEART	2	DeathCause	Char	26	85	Cause of Death
8	WORK.HEART	8	Diastolic	Num	8	32	
9	WORK.HEART	6	Height	Num	8	16	
10	WORK.HEART	10	MRW	Num	8	48	Metropolitan Relative Weight
11	WORK.HEART	4	Sex	Char	6	111	
12	WORK.HEART	11	Smoking	Num	8	56	
13	WORK.HEART	17	Smoking_Status	Char	17	145	Smoking Status
14	WORK.HEART	1	Status	Char	5	80	
15	WORK.HEART	9	Systolic	Num	8	40	
16	WORK.HEART	7	Weight	Num	8	24	
17	WORK.HEART	16	Weight_Status	Char	11	134	Weight Status

Figure 6. Varlist2 data set (extended attributes from PROC CONTENTS)



	Member	ExtendedAttribute	AttributeVariable	AttributeCharValue	AttributeNumValue
1	WORK.HEART	Unit	Height	Inches	.
2	WORK.HEART	Unit	Smoking	Cigarettes per day	.
3	WORK.HEART	Unit	Weight	Pounds	.
4	WORK.HEART	ValidValues	Chol_Status	<200 = Desirable; 200-239 = Borderline; >240 = High	.
5	WORK.HEART	ValidValues	DeathCause	Values include Cancer, CVD, CHD, Other, and Unknown. Missing if Px still alive.	.
6	WORK.HEART	ValidValues	Smoking	0 = does not smoke (since last exam)	.

Figure 7. Varlist2B data set (transposed version of extended attributes)



SAS - [VIEWTABLE: Work.Varlist2b]

File Edit View Tools Data Solutions Window Help

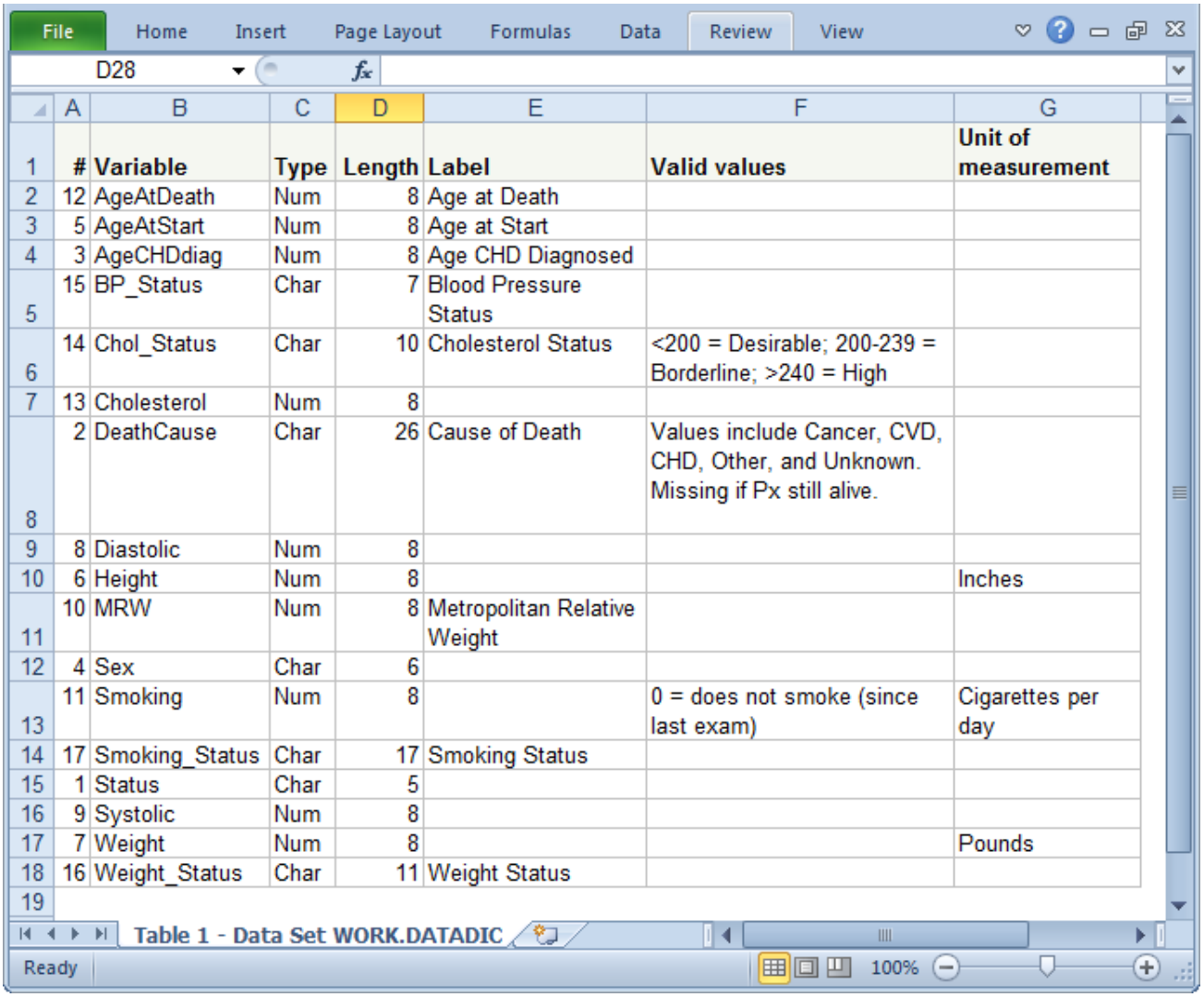
Explorer

ontents of 'Wo

Heart

	AttributeVariable	_NAME_	_LABEL_	ValidValues	Unit
1	Chol_Status	AttributeCharValue	Character Value	<200 = Desirable; 200-239 = Borderline; >240 = High	
2	DeathCause	AttributeCharValue	Character Value	Values include Cancer, CVD, CHD, Other, and Unknown. Missing if Px still alive.	
3	Height	AttributeCharValue	Character Value		Inches
4	Smoking	AttributeCharValue	Character Value	0 = does not smoke (since last exam)	Cigarettes per day
5	Weight	AttributeCharValue	Character Value		Pounds

Figure 8. Final product in Excel



File Home Insert Page Layout Formulas Data Review View

D28

	A	B	C	D	E	F	G
1	#	Variable	Type	Length	Label	Valid values	Unit of measurement
2	12	AgeAtDeath	Num	8	Age at Death		
3	5	AgeAtStart	Num	8	Age at Start		
4	3	AgeCHDdiag	Num	8	Age CHD Diagnosed		
5	15	BP_Status	Char	7	Blood Pressure Status		
6	14	Chol_Status	Char	10	Cholesterol Status	<200 = Desirable; 200-239 = Borderline; >240 = High	
7	13	Cholesterol	Num	8			
8	2	DeathCause	Char	26	Cause of Death	Values include Cancer, CVD, CHD, Other, and Unknown. Missing if Px still alive.	
9	8	Diastolic	Num	8			
10	6	Height	Num	8			Inches
11	10	MRW	Num	8	Metropolitan Relative Weight		
12	4	Sex	Char	6			
13	11	Smoking	Num	8		0 = does not smoke (since last exam)	Cigarettes per day
14	17	Smoking_Status	Char	17	Smoking Status		
15	1	Status	Char	5			
16	9	Systolic	Num	8			
17	7	Weight	Num	8			Pounds
18	16	Weight_Status	Char	11	Weight Status		
19							

Table 1 - Data Set WORK.DATADIC

Ready 100%

Code Set #1 – DATA step MERGE method[6]

```
data heart; set sashelp.heart; run;

proc datasets library=work;
modify heart;
        xattr set var
          Height (Unit = "Inches")
          Weight (Unit = "Pounds")
          Smoking (Unit = "Cigarettes per day")
          DeathCause (ValidValues = "Values include Cancer, CVD, CHD, Other, and Unknown. Missing
          if Px still alive.")
          Chol_Status (ValidValues = "<200 = Desirable; 200-239 = Borderline; >240 = High")
          Smoking (ValidValues = "0 = does not smoke (since last exam)");
run;
quit;

ods output variables=varlist;
ods output ExtendedAttributesVar=varlist2;

proc contents data=heart; run;

proc sort data=varlist2; by attributevariable; run;

proc transpose data=varlist2 out=varlist2B;
by attributevariable;
id extendedattribute;
var attributecharvalue;
run;

proc datasets library=work;
modify varlist2B;
        rename attributevariable = variable;
run;
quit;

proc sort data=varlist2B; by variable; run;
proc sort data=varlist; by variable; run;

data datadictionaryA;
merge varlist (drop= member pos) varlist2B (drop=_NAME_ _Label_);
by variable;
run;

proc datasets library=work;
modify datadictionaryA;
        label num = '#'
                variable = 'Variable'
                type = 'Type'
                len = 'Length'
                label = 'Label'
                unit = 'Unit of measurement'
                validvalues = 'Valid values';
run;
quit;

ods Excel file="C:\SAS\Data Dictionary A.xlsx" style=statistical;

proc print data=datadictionaryA noobs label; run;

ods Excel close;
```

[6] The DATA step MERGE is case sensitive. As such, variable names will need to be referred to in PROC DATASETS (and subsequently the "varlist2" data set) using the same case convention as in the original data set (i.e. the "varlist" data set).

Code set #2 – PROC SQL method[7]

```
data heart; set sashelp.heart; run;

proc datasets library=work;
modify heart;
        xattr set var
          Height (Unit = "Inches")
          Weight (Unit = "Pounds")
          Smoking (Unit = "Cigarettes per day")
          DeathCause (ValidValues = "Values include Cancer, CVD, CHD, Other, and Unknown. Missing
          if Px still alive.")
          Chol_Status (ValidValues = "<200 = Desirable; 200-239 = Borderline; >240 = High")
          Smoking (ValidValues = "0 = does not smoke (since last exam)");
run;
quit;

ods output variables=varlist;
ods output ExtendedAttributesVar=varlist2;

proc contents data=heart; run;

proc sort data=varlist2; by attributevariable; run;

proc transpose data=varlist2 out=varlist2B;
by attributevariable;
id extendedattribute;
var attributecharvalue;
run;

proc sql;
create table datadictionaryB as
select  v.num label='#',
        v.variable label='Variable',
        v.type label = 'Type',
        v.len label = 'Length',
        v.label label='Label',
        x.validvalues label='Valid values',
        x.unit label='Unit of measurement'
from varlist as v left join varlist2b as x
        on upcase(v.variable) = upcase(x.attributevariable);
quit;

ods Excel file="C:\SAS\Data Dictionary B.xlsx" style=statistical;

proc print data=datadictionaryB noobs label; run;

ods Excel close;
```

[7] The SORT and TRANPOSE procedures are case sensitive. As such, variable names will need to be referred to in PROC DATASETS (and subsequently the "varlist2" data set) using the same case convention as in the original data set (i.e. the "varlist" data set).

References

The best way to learn more about any of these topics is simply to search the web for the latest and greatest papers available. At the time of this writing, these resources were particularly useful:

PROC DATASETS; The Swiss Army Knife of SAS Procedures
SAS Global Forum paper 274-2011 by Michael A. Raithel
Found at: http://support.sas.com/resources/papers/proceedings11/274-2011.pdf

Developer Reveals: Extended Data Set Attributes
SAS Global Forum paper 135-2013 by Diane Olson
Found at: http://support.sas.com/resources/papers/proceedings13/135-2013.pdf

The Greatest Hits: ODS Essentials Every User Should Know
SAS Global Forum paper 300-2011 by Cynthia Zender
Found at: http://support.sas.com/resources/papers/proceedings11/300-2011.pdf

Learning SAS by Example: A Programmer's Guide
Book by Ron Cody