

Extended Uses of the CONTENTS Procedure

Steven Wright, Quintiles, Inc., Research Triangle Park, NC

Abstract

The CONTENTS procedure is a versatile tool containing a large amount of valuable information. When employed as the foundation of programs designed to build more complex tools, its utility becomes even more obvious. I will be considering three different variations.

In the first example, I will show that it is easy to construct a simple data dictionary by combining the CONTENTS output from several data files. Organizing information in this way makes it simpler to spot database structure problems.

The second illustration will demonstrate how you could modify existing variable formats using information from CONTENTS. Data files created by other users sometimes contain unwanted default formats. These can be removed while maintaining other valuable formats.

Finally, I will show that the information from CONTENTS can be used to control macros designed to run prints or summary statistics on all or part of the files contained in a directory.

In summary, I plan to show that the potential usefulness of this procedure is largely a function of the creativity and needs of the user.

Introduction

One of the defining beauties of the SAS® system is its prodigious wealth of tools. My goal in this paper is to explore the expanded use of one SAS tool, namely the CONTENTS procedure. I offer this example to stimulate thinking into a broader use of all the SAS tools. While there is nothing particularly profound about the code to build these tools, I want to present the ideas to illustrate that most problems can be solved readily with SAS and some imagination.

Typically, CONTENTS is viewed as a finite procedure whose usefulness is somewhat limited.

I would disagree. Other useful tools can be built from it. They include:

- 1) A data dictionary
- 2) A format modifier
- 3) A code writer

Terminology

In this paper I will be using standard database terms from the SAS system and from other languages interchangeably. I will refer to fields and variables, rows and observations, datasets and (data) tables as one and the same. The purpose of this use of terminology is not to promote confusion but to broaden experience with other database disciplines and to try to reduce boredom by the use of synonyms.

Contents of CONTENTS Output

To understand the potential usefulness of the output dataset from PROC CONTENTS, we need to look at what information SAS provides the user in this file. Like many other parts of SAS, a quick scan of the output can leave you a little breathless. Figure 1 contains the description of the output data set from PROC CONTENTS.

At first, some fields seem meaningless. LIBNAME can have only one value because this procedure takes only a single libname. Similarly, MEMNAME appears to be repeated unnecessarily unless CONTENTS is run on the entire library, using the DATA=libname.ALL_ parameter. And who labels datasets, so why waste forty characters of space on a MEMLABEL field?

Is there really anything useful in this dataset? Yes! NAME and TYPE! Finally, some useful fields. These, along with LENGTH, constitute the part of CONTENTS used most often.

When we get to formats and informats, things become interesting. With both formats and

-----Variables Ordered by Position-----						
#	Variable	Type	Len	Pos	Format	Label
1	LIBNAME	Char	8	0		Library Name
2	MEMNAME	Char	8	8		Library Member Name
3	MEMLABEL	Char	40	16		Data Set Label
4	TYPEMEM	Char	8	56		Special Data Set Type (From TYPE=)
5	NAME	Char	8	64		Variable Name
6	TYPE	Num	8	72		Variable Type
7	LENGTH	Num	8	80		Variable Length
8	VARNUM	Num	8	88		Variable Number
9	LABEL	Char	40	96		Variable Label
10	FORMAT	Char	8	136		Variable Format
11	FORMATL	Num	8	144		Format Length
12	FORMATD	Num	8	152		Number of Format Decimals
13	INFORMAT	Char	8	160		Variable Informat
14	INFORML	Num	8	168		Informat Length
15	INFORMD	Num	8	176		Number of Informat Decimals
16	JUST	Num	8	184		Justification
17	NPOS	Num	8	192		Position in Buffer
18	NOBS	Num	8	200		Observations in Data Set
19	ENGINE	Char	8	208		Engine Name
20	CRDATE	Num	8	216	DATETIME16.	Create Date
21	MODATE	Num	8	224	DATETIME16.	Last Modified Date
22	DELOBS	Num	8	232		Deleted Observations in Data Set
23	IDXUSAGE	Char	9	240		Use of Variable in Indexes
24	MEMTYPE	Char	8	249		Library Member Type
25	IDXCOUNT	Num	8	257		Number of Indexes for Data Set
26	PROTECT	Char	3	265		Password Protection (Read Write Alter)
27	COMPRESS	Char	8	268		Compression Routine
28	REUSE	Char	3	276		Reuse Space
29	SORTED	Num	8	279		Sorted and/or Validated
30	SORTEDBY	Num	8	287		Position of Variable in Sortedby Clause
31	CHARSET	Char	8	295		Host Character Set
32	COLLATE	Char	8	303		Collating Sequence
33	NODUPKEY	Char	3	311		Sort Option: No Duplicate Keys
34	NODUPREC	Char	3	314		Sort Option: No Duplicate Records

Figure 1. Description of PROC CONTENTS Output Dataset

informats we have the name, primary length, and decimal length as separate fields.

Now, how might this information be used?

Data Dictionary

One use of the CONTENTS output dataset that I wish to demonstrate is the construction of a simple data dictionary. Generally my approach to creating a data dictionary is to combine all variables from a project into a single list. This list could be used as: a guide to the overall structure of a set of data tables, as a "desk reference" to be

used when writing code, and as check for uniformity of field type and length. My discussion will focus on the last point. An example will be given based on data from clinical research dealing with adverse events.

Generally, I prefer to keep this dictionary simple, as shown in Figure 2. There are many other useful pieces of information that could easily be added. In this case I have kept the following: variable name, the dataset or member name, the format, the last modification date and the number of records or observations in each dataset. Now, what can we learn from such a list?

(Field) NAME	MEMNAME	FORMAT	MODATE	NOBS
RECDT	DETAIL	DATETIME 16.0	11MAR95:05:32	1008
	PATIENT	DATE 7.0	11MAR95:05:34	851
RELATION	DETAIL	.	11MAR95:05:32	1008
	PATIENT	.	11MAR95:05:34	851
REPTERM	SUMMARY	\$ 58	11MAR95:05:29	2279
	SYMPTOM	\$ 58	11MAR95:05:38	875
RESOLVE	DETAIL	DATE 7.0	11MAR95:05:32	1008
RESOLVEA	DETAIL	.	11MAR95:05:32	1008
	SUMMARY	\$ 1	11MAR95:05:29	2279
RESULT	DETAIL	.	11MAR95:05:32	1008
ROUTE	CM	.	11MAR95:05:31	5482
	DRUG	.	11MAR95:05:33	1131
GENDER	PATIENT	.	11MAR95:05:34	851
	DEMOG	\$ 1	11MAR95:05:36	783

Figure 2. Sorted Output From a Sample PROC CONTENTS

First, note that the variables, REPTERM and ROUTE, are used in multiple data tables with the same structure. This was done intentionally because these items are related, but are found in conjunction with different types of data. Collecting similar data in identical fields is often helpful for analysis programming where such fields might be combined or used in similar listings.

In the "real world" it is sometimes necessary to break good database design rules and collect redundant data. At other times it is highly advisable to go back and fix bad database structures. A composite list of datasets, fields, and formats highlights these areas for the database designers up front, and for the analysis programmers latter on. Let us consider two examples.

Notice that the demographic variable GENDER is collected in two tables. Perhaps this redundancy is needed, but these two fields should agree in type and format. Here they do not. These inconsistencies should be fixed so that coding can be done correctly. Think of the fact that the

code "IF GENDER='M' ..." will only generate a mild warning when running against the PATIENT table. This warning could easily be missed, and the demographics of an analysis would be altered. Usually it is much better to correct a database initially rather than building a list of exceptions to standard coding.

The discordant formats associated with Receipt Date (RECDT) stand out in this combined database description shown in Figure 2. In looking at separate contents or prints, you probably would not see this discrepancy. Another point worthy of note concerning the difference in formats of these two fields is that they contain different data. Date fields count in units of days; date-time fields count in seconds. If these variables are not handled properly, it is quite possible to miscalculate duration.

This example was constructed to show some gross problems that can be easily spotted with a visual examination of this modified PROC CONTENTS output. It might be more reliable to have the computer search for mismatched structures, and to report only those fields where mismatches occur. This way the more subtle

errors will be caught. This is especially critical in wide data bases where there are many variables to examine.

Modifying or Removing Formats

It is very common to receive a data table that has user-defined, permanently attached formats. Often we either do not receive those formats, or do not want them. A quick fix might be to write the format code:

```
FORMAT _NUMERIC_ _CHARACTER_;
```

This code works well provided there are no fields with desirable formats. However, if you want to keep specific formats, the output file from CONTENTS does allow you to construct a macro to keep them.

There are three main steps needed for such a macro. The first is to create the standard output dataset from the CONTENTS procedure.

Next, build a data step to select the observations with the desired formats and discard the others. These variable names and their formats are concatenated into a retained string. This string is written to a macro variable after the last record from the CONTENTS output has been processed.

Finally, use the DATASETS procedure to remove all formats as illustrated by the FORMAT statement shown earlier. If there are specific formats to reapply, execute another FORMAT statement containing the information from the macro variable previously constructed.

Code Writer

Often you will need a print of all data in a project library. When this happens, it would be convenient to be able to write the following code:

```
PROC PRINT DATA=LIBNAME._ALL_;
```

However, this code does not work! Here again an opportunity exists to use output from CONTENTS and our creativity to build a macro that will write code for us.

Doing just a print of all the files is trivial, and does not get you anything special for your effort. A careful review of the information provided from CONTENTS (See Fig. 1) reveals several items that could be used to construct a title or footnote in your print. The number of observations, dataset creation and modification dates, and sort order might be helpful in the title. There is also plenty of esoteric, but important, information for a footnote. Some of these include: IDXUSAGE (index usage), COMPRESS, NODUPKEY, and the FSEDIT related items, DELOBS and REUSE.

Rather than just processing all files, any one of several different criteria could be applied to select which files to process. The most obvious criterion would be the modification date.

If summary statistics of a set of data tables are needed, a smart piece of code could be written to help. Commonly PROC MEANS is run to pick up the numeric variables and PROC FREQ is run to do the character ones. Generally, FREQ should be restricted to character variables using the special name list, _CHARACTER_, in the TABLE statement. However, if the length of some character variables is more than 16, you should use PROC SUMMARY because FREQ truncates to 16 characters. Using the information from CONTENTS, a macro could intelligently build code which selects the appropriate procedure. You might also restrict the summary statistics to "short" character fields to avoid comment entries.

In the age of "Data Warehousing", the information in the field ENGINE could be significant. Did these data come from version 5, 6.04, 6.06, etc. or is it an export dataset? Is it from Oracle®, DB2®, or another relational database? This information could be invaluable, especially in a few months or years when someone wants to know the origin a particular data point. If that information is extracted and put on the page with the data, the question probably will never reoccur.

Coders' Corner

I have tried to allude to what is needed to accomplish the tasks we have discussed without actually writing the code. There are a few things that I want to mention here that I hope are useful to those who are newer to the SAS system.

In our output file some things seem to be repeated endlessly like COMPRESS, REUSE, and SORTED. Other information is constantly changing like NAME, and to a lesser degree, format fields. However, the field SORTEDBY is generally, but not always, missing. When SORTEDBY is not missing, it contains the sort order. The importance of this field should not be overlooked even though it contains mostly missing values.

Three separate fields contain the information regarding the format of a field. It is frequently preferable to make a single entity from these three fields. To create the description in a desired format, you can compress a concatenated string, constructed in part using the PUT function.

Conclusion

There are many uses of CONTENTS output. SAS provides a friendly environment for innovative, even imaginative, coding. My hope is that this brief consideration of some uses of a procedural output dataset will spur your interest in tackling similar opportunities in your coding environment.

Acknowledgments

SAS® is a registered trademark or trademark of SAS Institutes Inc. ORACLE® is a registered trademark or trademark of ORACLE Corporation. DB2® is a registered trademark or trademark of International Business Machines Corporation.

® indicates USA registration.