SESUG Paper 137-2025

RIGHT-SIZING CHARACTER VARIABLES: A SAS® DATA MANAGEMENT STRATEGY FOR OPTIMIZING LARGE DATASETS

Divya Gadde, Health Affairs Institute, West Virginia University

ABSTRACT

Transforming raw files into analysis-ready datasets presents many challenges. This paper describes optimization methods used to make data usable in SAS®. Without metadata or layout documentation, traditional loading methods can result in bloated and inefficient SAS datasets. This paper outlines how %RESIZE macro can identify and apply optimal variable lengths, reducing dataset size and improving usability for analysts. This method significantly streamlined performance while preserving data integrity.

INTRODUCTION

When working with huge datasets with hundreds of variables, default variable lengths can lead to excessive storage, slow processing, and limited usability. Although COMPRESS option can also reduce the space, it must be executed every time a dataset is created. To overcome that we can use simple macro to identify and apply optimal variable lengths, reducing dataset size and improving usability for analysts.

This paper outlines how to reduce the length of character variables by determining the maximum length and dynamically resize them to their actual lengths using simple %RESIZE macro. The goal of %RESIZE macro is to calculate the actual length of each character variable and resize them accordingly which thereby reduces the disk storage and improves the processing time.

DYNAMIC VARIABLE LENGTH OPTIMIZATION

What is Defined and Actual length?

Defined length is the maximum size a variable can hold, and Actual length is the actual size of data currently stored in the variable. The example below shows the difference between Defined length and Actual length.

```
Data name;
  length name $15.; /*defined length(maximum) is 15 characters*/
  name = "Alice";   /*actual length is 5 characters*/
run;
```

Defined length is always fixed and allocated size from length statement in the above example. Whereas the Actual length varies based on how many characters are stored in that variable.

Resizing the variables to Actual length could be a good data management practice helping with storage saving by reducing the file size, faster I/O resulting in better performance.

%RESIZE macro offers an automated method of calculating the maximum Actual length of each character variable in the dataset and then redefines those variables using only as much length as needed. %RESIZE macro automates the process of:

1. Scanning a dataset for character variables
2. Calculating their maximum lengths
3. Creating dynamic code with calculated length, format, and informat of character variables
4. Applying the formats and creating a new resized dataset

The next section describes the annotated code for %RESIZE macro and the impact of it when applied to the same dataset mentioned in the previous example. Note that the entire program can be found in Appendix A.

INITIALIZE AND RETRIEVE METADATA

The MACRO opens the input dataset using the function `OPEN()` and determines the number of variables using `ATTRN()`. This helps dynamically define the size of the temporary array.

```
 /* Step 0: Get number of character variables*/
%let dsid  = %sysfunc(open(&indata));
%let nvars = %sysfunc(attrn(&dsid,nvars));
%let rc    = %sysfunc(close(&dsid));
```

CALCULATE MAXIMUM LENGTHS OF CHARACTER VARIABLES

The DATA step scans through each character variable using arrays. It stores the maximum observed length of each variable across all records in a temporary array `_s[ ]`. When the last row is reached (`_eof`), it outputs a summary dataset `size` with variable names, maximum lengths, and corresponding format and informat strings.

```
data size(keep = _name_ _length_ _format_ _informat_);
  set &data end=_eof;
  array _c[*] _character_;
  array _s[104] _temporary_; /* adjust this size if needed */
  do _i_ = 1 to dim(_c);
    _s[_i_] = max(_s[_i_], length(_c[_i_]));
  end;

  if _eof then do _i_ = 1 to dim(_c);
    length _name_ $32 _format_ _informat_ $10;
    _name_      = vname(_c[_i_]);
    _length_    = _s[_i_];
    _format_    = strip(put(_length_, 4.)) || ".";
    _informat_ = strip(put(_length_, 4.)) || ".";
    output;
  end;
run;
```

GENERATE SAS CODE TO RESIZE VARIABLES

Using `DATA _NULL_` and the `FILE` statement, the macro writes the dynamic SAS code to a temporary file (`FT23F001`). This includes:

- A `RETAIN` statement to preserve variable order.
- `LENGTH`, `FORMAT`, and `INFORMAT` statements customized to each variable's Actual length.

```
data _null_;
  file FT23F001;
```

```
  if 0 then set &data; /* compile-time reference to _all_ */
  if _n_ = 1 then do;
    put 'retain ' (_all_) (=) ';' @;
    _file_ = translate(_file_, ' ', '=');
    put;
  end;

  set size;
  put 'length' _name_ ' $' _length_ ';';
  put 'format' _name_ ' $' _format_ ';';
  put 'informat' _name_ ' $' _informat_ ';';
run;
```

REBUILD THE DATASET WITH OPTIMIZED DEFINITIONS

The `%INCLUDE` statement reads and executes the dynamic code stored in `FT23F001`. A new dataset (`&outdata`) is created from the original dataset with resized character variables.

```
data resize;
  %include FT23F001 / source2;
  set &data;
run;
```

RESULTS AND IMPACT

Applying this optimization reduces the overall size of the datasets significantly. Analysts were able to access and process the data more efficiently, without performance bottlenecks or crashes. The dynamic reassignment of variable lengths not only saves storage but also reduces processing time by orders of magnitude on certain queries.

The example below highlights the impact of the %RESIZE macro in optimizing character variable lengths, resulting in a more efficient and manageable dataset.

Display 1 provides an example of a real-time SAS dataset downloaded from a SQL server and sent by a client. The data set has 107-character variables whose length is defined to be either 255 or 4000 bytes.

#	Variable	Type	Len	Format	Informat
89	AGE	Num	8		
90	BELOW_18	Num	8		
50	BMS_CNTY_CD	Char	255	$255.	$255.
51	BMS_CNTY_DESC	Char	4000	$4000.	$4000.
10	CRNT_RCRD_IND	Char	255	$255.	$255.
7	DATA_SRC_ID	Char	255	$255.	$255.
27	ENRL_AID_CTG_CD	Char	255	$255.	$255.
28	ENRL_AID_CTG_DESC	Char	4000	$4000.	$4000.
82	ENRL_BNFT_PLAN_DESC	Char	4000	$4000.	$4000.
81	ENRL_BNFT_PLAN_ID	Char	255	$255.	$255.
46	ENRL_CVRG_BGN_DT	Num	8	E8601DA10.	ANYDTDTE10.
29	ENRL_CVRG_CD_1	Char	255	$255.	$255.
31	ENRL_CVRG_CD_2	Char	255	$255.	$255.
33	ENRL_CVRG_CD_3	Char	255	$255.	$255.
35	ENRL_CVRG_CD_4	Char	255	$255.	$255.
12	ENRL_CVRG_CD_5	Char	255	$255.	$255.
58	ENRL_CVRG_CD_6	Char	255	$255.	$255.
60	ENRL_CVRG_CD_7	Char	255	$255.	$255.
62	ENRL_CVRG_CD_8	Char	255	$255.	$255.
42	ENRL_CVRG_CD_9	Char	255	$255.	$255.
44	ENRL_CVRG_CD_10	Char	255	$255.	$255.
30	ENRL_CVRG_DESC_1	Char	4000	$4000.	$4000.
32	ENRL_CVRG_DESC_2	Char	4000	$4000.	$4000.
34	ENRL_CVRG_DESC_3	Char	4000	$4000.	$4000.
11	ENRL_CVRG_DESC_4	Char	4000	$4000.	$4000.
57	ENRL_CVRG_DESC_5	Char	4000	$4000.	$4000.
59	ENRL_CVRG_DESC_6	Char	4000	$4000.	$4000.
61	ENRL_CVRG_DESC_7	Char	4000	$4000.	$4000.
13	ENRL_CVRG_DESC_8	Char	4000	$4000.	$4000.
43	ENRL_CVRG_DESC_9	Char	4000	$4000.	$4000.

Display 1. Variable lengths before applying RESIZE macro

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Informat
120	AGE	Num	8		
121	BELOW_18	Num	8		
58	BMS_CNTY_CD	Char	7	$7.	$7.
59	BMS_CNTY_DESC	Char	24	$24.	$24.
18	CRNT_RCRD_IND	Char	1	$1.	$1.
15	DATA_SRC_ID	Char	4	$4.	$4.
34	ENRL_AID_CTG_CD	Char	13	$13.	$13.
35	ENRL_AID_CTG_DESC	Char	30	$30.	$30.
114	ENRL_BNFT_PLAN_DESC	Char	37	$37.	$37.
113	ENRL_BNFT_PLAN_ID	Char	15	$15.	$15.
52	ENRL_CVRG_BGN_DT	Num	8	E8601DA10.	ANYDTDTE10.
36	ENRL_CVRG_CD_1	Char	7	$7.	$7.
38	ENRL_CVRG_CD_2	Char	7	$7.	$7.
40	ENRL_CVRG_CD_3	Char	7	$7.	$7.
42	ENRL_CVRG_CD_4	Char	7	$7.	$7.
20	ENRL_CVRG_CD_5	Char	7	$7.	$7.
81	ENRL_CVRG_CD_6	Char	7	$7.	$7.
83	ENRL_CVRG_CD_7	Char	7	$7.	$7.
85	ENRL_CVRG_CD_8	Char	7	$7.	$7.
48	ENRL_CVRG_CD_9	Char	7	$7.	$7.
50	ENRL_CVRG_CD_10	Char	7	$7.	$7.
37	ENRL_CVRG_DESC_1	Char	46	$46.	$46.
39	ENRL_CVRG_DESC_2	Char	46	$46.	$46.
41	ENRL_CVRG_DESC_3	Char	46	$46.	$46.
19	ENRL_CVRG_DESC_4	Char	46	$46.	$46.
80	ENRL_CVRG_DESC_5	Char	46	$46.	$46.
82	ENRL_CVRG_DESC_6	Char	46	$46.	$46.
84	ENRL_CVRG_DESC_7	Char	46	$46.	$46.
21	ENRL_CVRG_DESC_8	Char	46	$46.	$46.
49	ENRL_CVRG_DESC_9	Char	46	$46.	$46.

Output 1. Variable lengths after applying RESIZE macro.

File Size	611GB
File Size (bytes)	656028979200

Display 2. File size before applying RESIZE macro.

File Size	41GB
File Size (bytes)	43653160960

Display 3. File Size of after applying RESIZE macro

```
NOTE: Writing HTML5(EGHTML) Body file: EGHTML
27
28          proc freq data=raw.enroll;
29          tables crnt_rcrd_ind;
30          run;

NOTE: There were 4576093 observations read from the data set RAW.ENROLL.
NOTE: PROCEDURE FREQ used (Total process time):
      real time           51:51.93
      cpu time            13.61 seconds
```

Output 2. When the same dataset with 4M records and 118 variables set to calculated length the processing time has drastically decreased from 51 minutes to 4 minutes, and the size of the dataset was also reduced there by increasing the disk storage.

```
NOTE: Writing HTML5(EGHTML) Body file: EGHTML
27
28          proc freq data=raw.enroll_resized;
29          tables crnt_rcrd_ind;
30          run;

NOTE: There were 4576093 observations read from the data set RAW.ENROLL_RESIZED.
NOTE: PROCEDURE FREQ used (Total process time):
      real time           4:44.07
      cpu time            0.62 seconds
```

Output 3. Processing time after applying RESIZE macro.

IMPLEMENTATION

To use the %RESIZE macro, simply provide the name of your input dataset and specify the name of the output dataset you want to create. Detailed code provided in the Appendix section can be used to create a cleaner version of the dataset with improved performance and reduced storage.

CONCLUSION

Transforming huge datafiles into optimized SAS datasets requires more than basic import steps—it

demands thoughtful handling of space, structure, and usability. This paper demonstrates how SAS can be used creatively to overcome real-world ETL limitations, especially when working with large and undocumented datasets. The approach is scalable, efficient, and applicable across other high-volume data environments where system resources and performance matter.

REFERENCES

SAS Institute Inc. (2023). *SAS® 9.4 Functions and CALL Routines: Reference* (6th ed.). Cary, NC: SAS Institute Inc.
https://documentation.sas.com/doc/en/pgmsascdc/v_034/lefunctionsref/titlepage.htm

SAS Institute Inc. (2022). *SAS® 9.4 Macro Language: Reference*. Cary, NC: SAS Institute Inc.
https://documentation.sas.com/doc/en/mcrolref/9.4/titlepage.htm

SAS Institute Inc. (2022). *SAS® 9.4 DATA Step Programming: Tips and Techniques*. Cary, NC: SAS Institute Inc.

Manning, K., & Delery, C. (2018). *Reduce Dataset Size Using Character Variable Truncation Techniques in SAS*. Proceedings of the SAS Global Forum 2018. Cary, NC: SAS Institute Inc.
https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/1445-2018.pdf

ACKNOWLEDGMENTS

I would like to thank Brandon Marsh, Bryce Weaver, Gideon Devadason and Marco Mazzocchi who provided valuable feedback during the development and testing of the RESIZE macro. Special thanks to the data management and analytics team for sharing insights on real-world challenges in working with large datasets.

CONTACT INFORMATION

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

Divya Gadde
West Virginia University
Divya.gadde@hsc.wvu.edu

Appendix A

```
/*************************************************************************************************************************/
 /* Program: RESIZE macro                                                                                                 * /
 /* Author: Divya R Gadde                                                                                                 */
 /* Created: 07/02/2025                                                                                                   */
 /* Purpose: Resize Character variables to their actual length                                                           */
/* Parameters: Users must specify the following parameters:
                Indata – name of the target dataset
                outdata – name of the resized dataset
/*************************************************************************************************************************/
   %macro RESIZE(indata=, outdata=);
    %local dsid nvars rc;
   /* Step 0: Get number of character variables*/
   %let dsid  = %sysfunc(open(&indata));
   %let nvars = %sysfunc(attrn(&dsid,nvars));
   %let rc    = %sysfunc(close(&dsid));

   /* Step 1: Calculate max lengths of character variables */

   data size(keep = _name_ _length_ _format_ _informat_);
     set &data end=_eof;
     array _c[*] _character_;
     array _s[104] _temporary_; /* adjust this size if needed */
```

```
  do _i_ = 1 to dim(_c);
    _s[_i_] = max(_s[_i_], length(_c[_i_]));
  end;

  if _eof then do _i_ = 1 to dim(_c);
    length _name_ $32 _format_ _informat_ $10;
    _name_      = vname(_c[_i_]);
    _length_    = _s[_i_];
    _format_    = strip(put(_length_, 4.)) || ".";
    _informat_ = strip(put(_length_, 4.)) || ".";
    output;
  end;
run;

/* Step 2: Create dynamic code with calculated lengths/formats */
filename FT23F001 temp;
options missing = ' ';

data _null_;
  file FT23F001;
  if 0 then set &data; /* compile-time reference to _all_ */
  if _n_ = 1 then do;
    put 'retain ' (_all_) (=) ';' @;
    _file_ = translate(_file_, ' ', '=');
    put;
  end;

  set size;
  put 'length ' _name_ ' $' _length_ ';';
  put 'format ' _name_ ' $' _format_ ';';
  put 'informat ' _name_ ' $' _informat_ ';';
run;

/* Step 3: Apply formatting and resizing to a new dataset */
data resize;
  %include FT23F001 / source2;
  set &data;
run;

/* Step 4: Verify structure */
proc contents data=resize varnum;
run;
```