

POWER SAS: A Survival Guide

Copyright ©2002 by Kirk Paul Lafler

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

Library of Congress Cataloging-in-Publication Data

Lafler, Kirk Paul, 1956-

Power SAS : a survival guide / Kirk Paul Lafler.  
p. cm.

ISBN 1-59059-066-X

1. SAS (Computer file) I. Title.

QA276.4 .L34 2002

005.3'042--dc21

2002013258

Printed and bound in the United States of  
America 12345678910

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The information in this book is distributed on an "as is" basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

Technical Reviewers: Charles Shipp,  
Mike DiGregorio, and Sunil Kumar Gupta

Editorial Directors: Dan Appleman, Gary Cornell,  
Jason Gilmore, Simon Hayes, Karen Watterson,  
John Zukowski

Managing Editor: Grace Wong

Project Manager and Development Editor:  
Tracy Brown Collins

Copy Editor: Corbin Collins

Production Manager: Kari Brooks

Compositor: Diana Van Winkle,  
Van Winkle Design

Artist: Cara Brunk, Blue Mud Productions

Indexer: Rebecca Plunkett

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Marketing Manager: Stephanie Rodriguez

Distributed to the book trade in the United States  
by Springer-Verlag New York, Inc., 175 Fifth  
Avenue, New York, NY, 10010 and outside the  
United States by Springer-Verlag GmbH & Co. KG,  
Tiergartenstr. 17, 69112 Heidelberg, Germany.  
In the United States, phone 1-800-SPRINGER,  
email [orders@springer-ny.com](mailto:orders@springer-ny.com), or visit  
<http://www.springer-ny.com>.  
Outside the United States, fax +49 6221 345229,  
email [orders@springer.de](mailto:orders@springer.de), or visit  
<http://www.springer.de>.

For information on translations, please contact  
Apress directly at 2560 9th Street, Suite 219,  
Berkeley, CA 94710. Phone 510-549-5930,  
fax 510-549-5939, email [info@apress.com](mailto:info@apress.com),  
or visit <http://www.apress.com>.









The source code for this book is available  
to readers at <http://www.apress.com> in the  
Downloads section. You will need to answer  
questions pertaining to this book in order to  
successfully download the code.





## CHAPTER 2

# Data Access

BASE SAS SOFTWARE provides a variety of tools for reading and processing files of virtually any type and format. This chapter presents an assortment of tips that will make your data input and retrieval tasks an easy one. Starting with the basics, you'll learn many useful techniques for accessing input files successfully. Once these techniques are mastered, you'll be ready to move up to the more challenging data-access requirements. You'll learn a variety of techniques that will make your data access requirements fast, easy, and fun.

In this chapter, you'll learn how to

-  Read in-stream data that resides inside the program
-  Read blank-, comma-, and tab-delimited input files
-  Use different styles of the INPUT statement
-  Specify what record to begin reading and stop processing in an input file
-  Specify column and line pointers when reading an input file
-  Handle input files that contains carriage control characters
-  Read multiple input files
-  Identify the last record in an input file

-  Read variable length and hierarchical input files
-  Read SAS data sets sequentially and directly
-  Subset observations from a SAS data set
-  Read and process data using SQL

## External Data

This section includes tips that will show you how to process all types of input files. From in-stream data to external input files, you'll learn how to read and process blank-, comma-, and tab-delimited input files, fixed- and variable-length input files, input files containing carriage control characters, multiple (concatenated) and hierarchical input files, and much more.

### 1 Exploring file types

The Base SAS product, and in particular the DATA step, contains marvelous tools for reading and processing file types of virtually any type and format. Reading raw external input data is a breeze with the SAS software. SAS can read and retrieve input files of all types including fixed- and variable-length, blank-delimited, comma- and tab-delimited, named-input, hierarchical files, hybrid files

## 19 Using column pointers with formatted-style of input

In the previous tip, you learned how formatted-style of input can be used to read an external file containing data values in the exact same columns across all records. Now we'll expand the capabilities of formatted-style of input to include the ability to control where in the input buffer data record to begin reading data. This is important because it gives you complete flexibility to choose where in the input buffer record an input value starts. The available column pointers are described in Table 2-1.

Table 2-1. Column Pointer Controls

POINTER	DESCRIPTION
@n	Positions the pointer to column <i>n</i> .
+n	Moves the pointer <i>n</i> columns to the right.

In the example, in-stream data is read using a formatted-style of input with column pointer controls. The INPUT statement begins by positioning the pointer to column 1 to read the movie title from the next 11 columns. It then moves the pointer 5 columns to the right in the input buffer record, skipping over the RATING data value, to read the RENTED date in columns 17 through 26 as follows.

Code:

```
DATA MOVIES;
  INPUT @1 TITLE $11. +5 RENTED MMDDYY10.;
  FORMAT RENTED MMDDYY10.;
  CARDS;
Brave HeartR    07/21/2001
Jaws            PG    02/03/1998
Rocky           PG    11/18/1997
```

```
Titanic        PG-1305/14/1998
;
RUN;
PROC PRINT;
RUN;
```

Results:

Obs	TITLE	RENTED
1	Brave Heart	07/21/2001
2	Jaws	02/03/1998
3	Rocky	11/18/1997
4	Titanic	05/14/1998

## 20 Specifying the number of lines available to the input pointer

SAS lets you specify how many lines are available to the input pointer on a read operation. By specifying the N= option in the INFILE statement, you can control how many input lines are available to the input pointer to assist in navigating up or down the records in an input file.



The default number of lines available to the input pointer is 1.

## 21 Using line pointers with formatted-style of input

Line pointers can also be used with formatted-style of input to specify which input line to read data from. You use line pointers as a way to move up or down the records in an input file. This is important because it gives complete flexibility to choose which record to read. Combine line pointer with column pointers (see previous tip), and you have complete control over the horizontal and

## 20 Reading SAS data sets directly

By default, every SAS data set is read sequentially. But what if the data you want resides at or near the end of a large data set? This can translate into longer and more costly runs due to the nature of a sequential read. SAS provides a more direct way to read data sets. Known as *random* or *direct access*, observations can be accessed without having to read each observation in a sequential manner.

You can access an observation directly by using the **POINT=** option of the SET statement. You'll need to know the observation number to use the POINT= option. Although it can't be used with a WHERE statement, a WHERE= data set option, or BY statement, it does allow direct access to an observation or group of observations easily. By specifying the NOBS= and POINT= options, the following example selects the odd numbered observations from the MOVIES data set using DO-loop processing.



*Because a SAS data set is being read directly, a STOP statement is needed to prevent the endless looping of the DO-group loop.*

Code:

```
DATA ALTERNATE_RECORDS;
  DO OBSNUM=1 TO TOTOB5 BY 2;
    SET PATH.MOVIES
      NOBS=TOTOB5 POINT=OBSNUM;
    OUTPUT;
  END;
  STOP;
RUN;
```

## 21 Reading a SAS data set into real memory

If you open and read a SAS data set multiple times during a session, and have sufficient real memory available, you may consider transferring data from disk (or some other storage medium) to memory. By specifying a **SASFILE** statement, the necessary buffers are automatically allocated to store the data in memory. Once data is read into real memory, the file remains open to accommodate other requests against the data until another SASFILE statement closes the file, or the program or session ends. By specifying the SASFILE statement, the following example opens and reads the MOVIES data set transferring observations from disk to memory.

Code:

```
SASFILE PATH.MOVIES OPEN;
DATA PG_MOVIES;
  SET MOVIES(WHERE=(RATING="PG"));
RUN;
SASFILE PATH.MOVIES CLOSE;
```

## 22 Specifying the number of buffers

Defining an adequate number of buffers (or holding areas) for data being sent to, or received from, disk is important when accessing and processing data. The default number of buffers for the **BUFNO=** option is 1, and the maximum is operating system-dependent. Under the Windows operating system, any number of buffers can be allocated because its value is based specifically on the amount of memory available. If uncertain about what value to assign the BUFNO= option, I recommend that users set BUFNO=MIN. This will assign a value of 0 and

Table 2-6. Dictionary Views

LIBNAME	MEMBER NAME	DESCRIPTION
SASHELP	VCATALOG	Provides catalog information.
SASHELP	VCOLUMN	Provides column names and attribute information.
SASHELP	VEXTFL	Provides the FILREF, pathname, and engine for external files.
SASHELP	VINDEX	Provides data set index information for each libref.
SASHELP	VMACRO	Provides information about macro variables, their scope, and value.
SASHELP	VMEMBER	Provides SAS library information including member name and type, indexes, and path.
SASHELP	VOPTION	Provides SAS system option names and settings for your environment.
SASHELP	VSCATLG	Provides librefs and member name for allocated SAS catalogs.
SASHELP	VSLIB	Provides allocated librefs and paths.
SASHELP	VSTABLE	Provides librefs and member names for allocated SAS data sets.
SASHELP	VSTABVW	Provides librefs, member names, and member types for allocated SAS data sets.
SASHELP	VSTYLE	Provides librefs, template names, style names, and notes.
SASHELP	VVIEW	Provides librefs and dictionary view names.
SASHELP	VTABLE	Provides librefs, member names and types, date created and modified, number of observations, observation length, number of variables, and performance tuning information.
SASHELP	VTITLE	Provides title location, title number, and title text.
SASHELP	VVIEW	Provides librefs, dictionary view names, member type, and engine name.

28 Accessing dictionary views

Accessing dictionary views stored in the SASHELP library is relatively easy. You'll need to know what kind of information you want and the name of the view (see tip #27 in this section) before you start. Once you've done this, you're ready to access the view in a DATA or PROC step. The first example accesses the VSLIB dictionary view in a DATA step and then prints the allocated librefs and paths.

Code:

```
DATA LIBREFS;  
  SET SASHELP.VSLIB;  
  WHERE UPCASE(LIBNAME)= 'PATH';  
RUN;  
PROC PRINT NOOBS;  
RUN;
```

The second example accesses the VOPTION dictionary view to print SAS System options and settings.

Table 2-8. Available Summary Functions

SUMMARY FUNCTION	DESCRIPTION
<b>AVG</b> <or> <b>MEAN</b>	Determines the average or mean of values.
<b>COUNT</b> <or> <b>FREQ</b> <or> <b>N</b>	Determines the number of nonmissing values.
<b>CSS</b>	Determines the corrected sum of squares.
<b>CV</b>	Determines the coefficient of variation (percent).
<b>MAX</b>	Determines the largest value.
<b>MIN</b>	Determines the smallest value.
<b>NMISS</b>	Determines the number of missing values.
<b>PRT</b>	Determines the probability of a greater absolute value of Student's t.
<b>RANGE</b>	Determines the difference between the maximum and minimum values.
<b>STD</b>	Determines the standard deviation.
<b>STDERR</b>	Determines the standard error of the average (or mean).
<b>SUM</b>	Determines the sum of values.
<b>SUMWGT</b>	Determines the sum of the WEIGHT variable values.
<b>T</b>	Determines the Student's t value for testing the hypothesis that the population mean is zero.
<b>USS</b>	Determines the uncorrected sum of squares.
<b>VAR</b>	Determines the variance.

## Summary

In this chapter, you learned the various ways to access data using Base SAS software. Whether your input data is blank-, comma-, or tab-delimited, fixed- or variable-length, or hierarchical, SAS has all right tool to get the job done quickly and, most importantly, correctly. You also learned the basics of the SQL procedure for accessing data from existing SAS data sets.









In the next chapter, “DATA Step Programming,” you’ll continue learning using a step-by-step approach about the DATA step programming language. Even if you’re not a programmer, you’ll find the tips and the many examples simple to understand.





## CHAPTER 3

# DATA Step Programming

THE DATA STEP programming language in the Base SAS software is a powerful and full-featured programming language. Whether you're a novice or advanced user, the tips covered in this chapter are designed to show you how to use it to your advantage. This chapter covers important keywords, structured coding techniques, date and time processing, operators and modifiers, SAS functions, program testing and debugging techniques, large file-processing techniques, and concepts such as observation loops, Program Data Vector (PDV), variable and value assignments, conditional processing, and logic branching—all to enable you to write better programs.

In this chapter, you'll learn how to

-  Direct control to the top of an observation loop
-  Retain values across observation loops
-  Assign a variable's length
-  Perform conditional logic
-  Implement best practices coding standards
-  Perform date and time processing
-  Use date formats and informats
-  Apply operators and modifiers and their order of evaluation

-  Use arithmetic, character, date and time, and array functions as part of SAS statements
-  Test and debug SAS program code
-  Process large file efficiently and quickly
-  Document and educate other users on complex processes, pieces of code, and other abstract coding constructs

## The DATA Step Programming Language

This section includes tips that will show you how to get the most from the DATA step programming language. From understanding the DATA step's many features and capabilities to visualizing how the observation loop works, you'll find ways to take command of this powerful programming language. You'll also learn valuable techniques that can be applied to your own program code quickly and easily.

### 1 Understanding the DATA step language

The DATA step language is at the heart of the SAS System. It's a full-featured programming language designed to perform complex and powerful computing operations. In contrast to other compiled languages, a SAS program is structured in a top-down fashion. What this means is that a multi-step program is grouped

## 19 Looping inside an observation loop

Looping inside the observation loop gives users a powerful way to repeat a process a number of times. By specifying a DO statement, users are provided three methods of constructing a loop inside a DATA step observation loop:

- Indexed DO loop
- DO WHILE loop
- DO UNTIL loop

## 20 Specifying an indexed DO loop

Indexed DO loops are coded inside an observation loop in a DATA step to iterate through a series of statements in a DO group. Essentially, the index value changes each time the DO group executes. Once the maximum value of the index value is reached, the DO loop stops after the last iteration.

## 21 Specifying a DO WHILE loop

Specifying a DO WHILE loop inside an observation loop permits a conditional block of code to be executed repeatedly. This allows the DO loop to continue running as long as the specified condition is true—otherwise it stops. Be advised that the condition is always evaluated at the top of the loop, and make sure that the condition is structured so that the loop can stop at some point. Otherwise, the DO WHILE loop could be an infinite loop and execute forever.

## 22 Specifying a DO UNTIL loop

Specifying a DO UNTIL loop inside an observation loop permits a conditional block of code to be executed repeatedly. The DO loop would continue running as long as the specified condition is true—otherwise it stops. Be advised that the DO UNTIL loop executes at least once because the condition is always evaluated at the bottom of the loop. Also make sure that the condition is structured so that the loop can stop at some point. Otherwise, the DO UNTIL loop could be an infinite loop and execute forever.

## 23 Best practices coding standards

Programmers and end-users have used the SAS software to develop programs and applications since the mid 1970s. Since then, many new versions and releases of the SAS software have been released. With each version and release, a variety of new products, user interfaces, windows, procedures, statements, options, format engines, output capabilities, templates, and an assortment of other tools and features far too numerous to list have been introduced. This means that many programs as well as chunks of code written in the 1970s, 1980s, and 1990s are still in operation today. It also means that best practices coding techniques (or the lack of) borrowed and used from programmers in the past have reappeared in various forms in code of the present.

Here are a number of simple suggestions and guidelines for applying best practice coding standards into existing as well as new program code. The following points have been successfully applied to bring a more manageable programming environment when using SAS software.



## 7 Using multiple percent sign (%) wildcards in a search pattern

Sometimes it is useful to be able to combine two or more wildcard characters to broaden the search capabilities. The following example uses two wildcards, one at the beginning of the search pattern consisting of the text “the” and one at the end. To simplify the search even further, an UPCASE function is specified to convert all lowercase characters to uppercase.

*Code:*

```
DATA OPERATORS;
  SET MOVIES(WHERE=
    (UPCASE(TITLE) LIKE "%THE%")
    KEEP=TITLE LENGTH RATING);
RUN;
```

*Results:*

Title	Length	Rating
Silence of the Lambs	118	R
The Hunt for Red October	135	PG
The Terminator	108	R
The Wizard of Oz	101	G

## 8 Using the underscore (\_) wildcard with the LIKE operator

Whereas the percent sign (%) matches multiple characters, the underscore ( ) wildcard character matches just a single character. For example, the following subset selects movies based on a search pattern containing text followed by three underscores to subset all “PG” and “PG-13” movies.

*Code:*

```
DATA OPERATORS;
  SET MOVIES(WHERE=
    (UPCASE(RATING) LIKE "PG__")
    KEEP=TITLE LENGTH RATING);
RUN;
```

*Results:*

Title	Length	Rating
Casablanca	103	PG
Christmas Vacation	97	PG-13
Forrest Gump	142	PG-13
Ghost	127	PG-13
Jaws	125	PG
Jurassic Park	127	PG-13
Michael	106	PG-13
National Lampoon's Vacation	98	PG-13
Poltergeist	115	PG
Rocky	120	PG
Star Wars	124	PG
The Hunt for Red October	135	PG
Titanic	194	PG-13

## 9 Truncating and comparing strings with a colon (:) modifier

When comparing strings of different lengths, the string with the shorter length generally has its value padded with trailing blanks in order to make the comparison. This process can be overridden by using the colon (:) modifier with a comparison operator to truncate the longer string. The following example compares two different length strings using a colon (:) modifier. The longer string is truncated for the comparison.

## Program Debugging

This section includes tips on how to go about debugging a SAS program to help rid those pesky syntax, data, system, and logic-related problems that often occur during the development of a program. Because debugging plays such an integral part in the creation of a well-written program, every SAS user should take the necessary time to review these simple but essential tips. You'll learn the various types of errors that can occur in a program, how to check for and correct syntax errors, data, system, and logic errors, and much more.



*Because program debugging is frequently associated with testing, users should also review the tips in the previous section, "Program Testing."*

### 1 Reading the SAS Log to aid in error detection

The SAS Log displays important information about a program and the SAS environment. It assists in solving program-related problems by providing the following information:

- The program statements that were executed
- Important notes or information about a program
- The SAS data sets that were read and created
- The number of lines read from an external input file
- The number of observations and variables that were read and written

- Syntax, warning, and data messages
- The time and memory used by each program step.

### 2 Understanding SAS software usage errors

Usage errors, as you may be already aware, cause the SAS System to stop processing, produce warnings, or produce unexpected results. There are four types of usage errors: 1) Syntax, 2) Data, 3) System-related, and 4) Programming (Logic).

### 3 Violating syntax

Syntax errors are a result of violating one of the rules of syntax (discussed in Chapter 1, "The Basics"). The Supervisor stops processing the current program step, passing control to the following step or, if one does not exist, stops processing altogether. Common causes for this type of error are misspelled keywords, missing semi-colons, data set and/or variable names that are too long, or an invalid character as the first position.

### 4 Checking for syntax errors

It is a good idea to check for syntax errors before processing data in a program. To check for syntax, add the following OPTIONS statement to the first line of your program:

*Code:*

```
OPTIONS OBS=0 NOREPLACE;
```