CHAPTER

21

DATA Step Processing

Definition

Using the DATA step is the primary method for creating a SAS data set with base SAS software. There are two kinds of SAS data sets: SAS data files and SAS data views. A SAS data file contains both a data portion and a data descriptor portion. A

SAS data view uses descriptor information and data from other files. A *DATA step* is a group of SAS language statements that begin with a DATA statement. The DATA statement is followed by other programming statements that manipulate existing SAS data sets or create SAS data sets from raw data files.

You can use the DATA step for

- creating SAS data sets (SAS data files or SAS data views)
- creating SAS data sets from input files that contain raw data (external files)
- creating new SAS data sets from existing ones by subsetting, merging, modifying, and updating existing SAS data sets
- analyzing, manipulating, or presenting your data
- computing the values for new variables
- report writing, or writing files to disk or tape
- retrieving information
- file management.

Note: A DATA step creates a *SAS data set*. This data set can be a SAS data file or a SAS data view. A SAS data file stores data values while a SAS data view stores instructions for retrieving and processing data. When you can use a SAS data view as a SAS data file, as is true in most cases, this book uses the broader term SAS data set. △

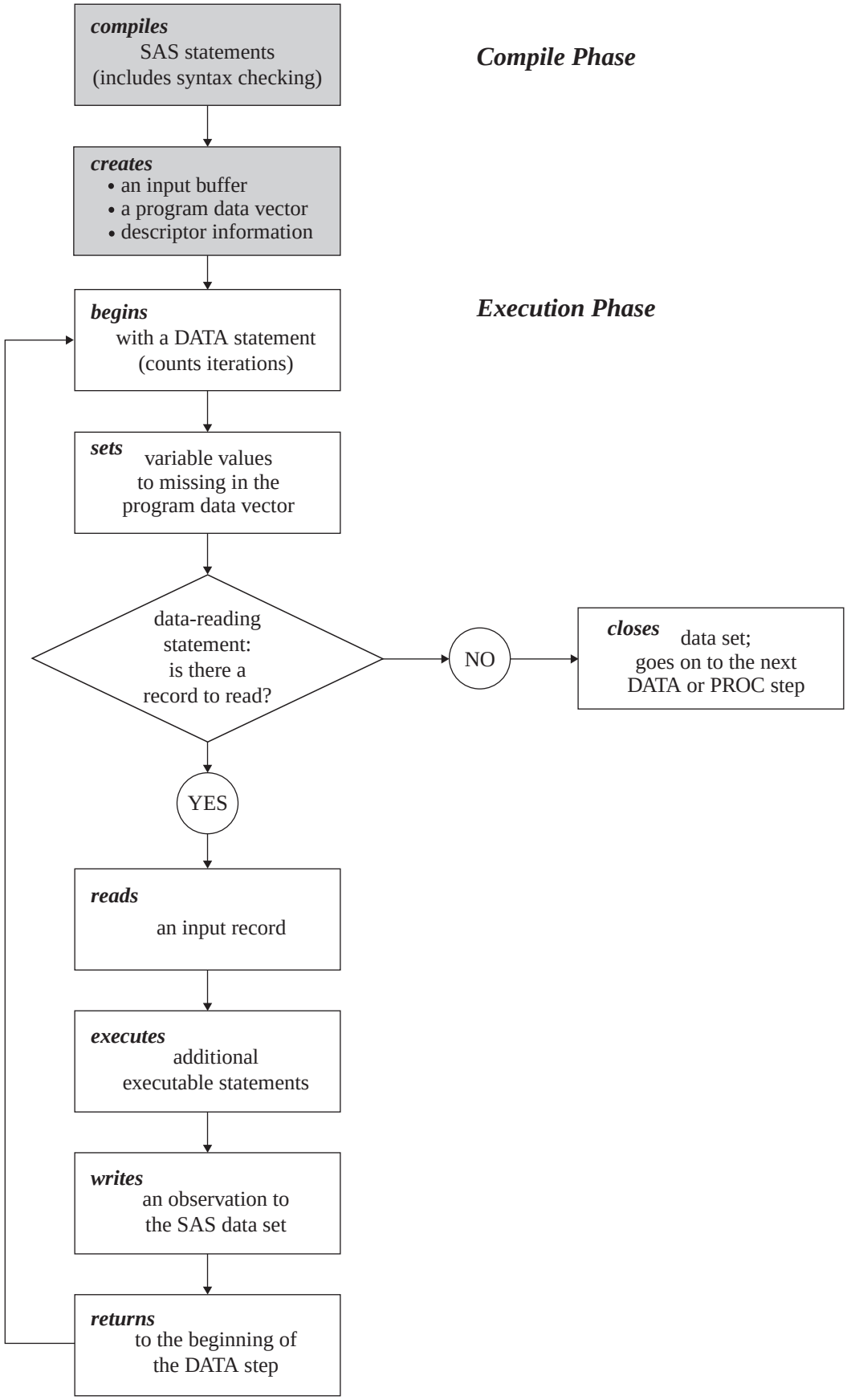
Note: In addition to the DATA step, several procedures in base SAS software create a SAS data set as part of their output. You can also use the FSEDIT and FSVIEW procedures in SAS/FSP to create and edit SAS data sets. △

Overview of DATA Step Processing

Flow of Action

When you submit a DATA step for execution, it is first compiled and then executed. The following figure shows the flow of action for a typical SAS DATA step.

Figure 21.1 Flow of Action in the DATA Step



The Compilation Phase

When you submit a DATA step for execution, SAS checks the syntax of the SAS statements and compiles them, that is, automatically translates the statements into machine code. In this phase, SAS identifies the type and length of each new variable, and determines whether a type conversion is necessary for each subsequent reference to a variable. During the compile phase, SAS creates the following three items:

input buffer — is a logical area in memory into which SAS reads each record of raw data when SAS executes an INPUT statement. Note that this buffer is created only when the DATA step reads raw data. (When the DATA step reads a SAS data set, SAS reads the data directly into the program data vector.)

program data vector (PDV) — is a logical area in memory where SAS builds a data set, one observation at a time. When a program executes, SAS reads data values from the input buffer or creates them by executing SAS language statements. The data values are assigned to the appropriate variables in the program data vector. From here, SAS writes the values to a SAS data set as a single observation.

Along with data set variables and computed variables, the PDV contains two automatic variables, _N_ and _ERROR_. The _N_ variable counts the number of times the DATA step begins to iterate. The _ERROR_ variable signals the occurrence of an error caused by the data during execution. The value of _ERROR_ is either 0 (indicating no errors exist), or 1 (indicating that one or more errors have occurred). SAS does not write these variables to the output data set.

descriptor information — is information that SAS creates and maintains about each SAS data set, including data set attributes and variable attributes. It contains, for example, the name of the data set and its member type, the date and time that the data set was created, and the number, names and data types (character or numeric) of the variables.

The Execution Phase

By default, a simple DATA step iterates once for each observation that is being created. The flow of action in the Execution Phase of a simple DATA step is described as follows:

1. The DATA step begins with a DATA statement. Each time the DATA statement executes, a new iteration of the DATA step begins, and the _N_ automatic variable is incremented by 1.
2. SAS sets the newly created program variables to missing in the program data vector (PDV).
3. SAS reads a data record from a raw data file into the input buffer, or it reads an observation from a SAS data set directly into the program data vector. You can use an INPUT, MERGE, SET, MODIFY, or UPDATE statement to read a record.
4. SAS executes any subsequent programming statements for the current record.
5. At the end of the statements, an output, return, and reset occur automatically. SAS writes an observation to the SAS data set, the system automatically returns to the top of the DATA step, and the values of variables created by INPUT and

assignment statements are reset to missing in the program data vector. Note that variables that you read with a SET, MERGE, MODIFY, or UPDATE statement are not reset to missing here.

6 SAS counts another iteration, reads the next record or observation, and executes the subsequent programming statements for the current observation.

7 The DATA step terminates when SAS encounters the end-of-file in a SAS data set or a raw data file.

Note: The figure shows the default processing of the DATA step. You can code data-reading statements (such as INPUT or SET), or data-writing statements (such as OUTPUT), in any order in your program. △

Processing a DATA Step: A Walkthrough

Sample DATA Step

The following statements provide an example of a DATA step that reads raw data, calculates totals, and creates a data set:

```
data total_points (drop=TeamName);  ❶
   input TeamName $ ParticipantName $ Event1 Event2 Event3;  ❷
   TeamTotal + (Event1 + Event2 + Event3);   ❸
   datalines;
Knights Sue    6  8  8
Cardinals Jane 9  7  8
Knights John   7  7  7
Knights Lisa   8  9  9
Knights Fran   7  6  6
Knights Walter 9  8 10
;
```

❶ The DROP= data set option prevents the variable TeamName from being written to the output SAS data set called TOTAL_POINTS.

❷ The INPUT statement describes the data by giving a name to each variable, identifying its data type (character or numeric), and identifying its relative location in the data record.

❸ The Sum statement accumulates the scores for three events in the variable TeamTotal.

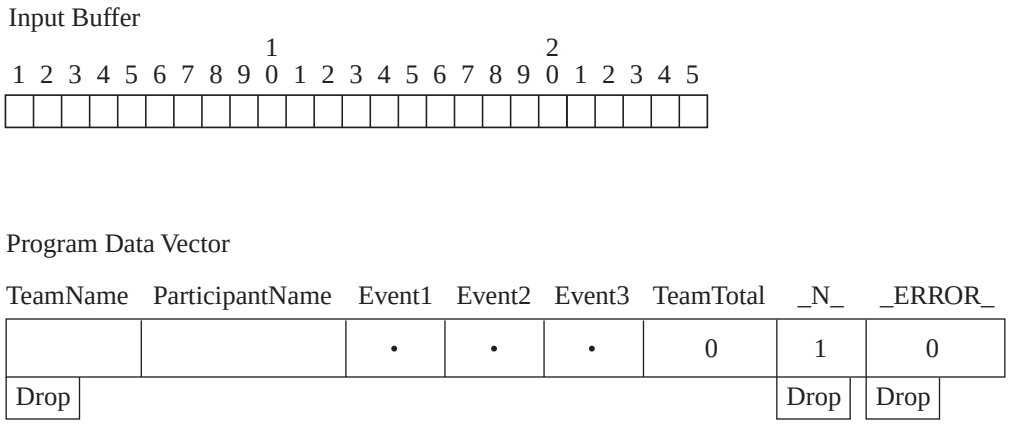
Creating the Input Buffer and the Program Data Vector

When DATA step statements are compiled, SAS determines whether to create an input buffer. If the input file contains raw data (as in the example above), SAS creates an input buffer to hold the data before moving the data to the program data vector (PDV). (If the input file is a SAS data set, however, SAS does not create an input buffer. SAS writes the input data directly to the PDV.)

The PDV contains all the variables in the input data set, the variables created in DATA step statements, and the two variables, _N_ and _ERROR_, that are automatically generated for every DATA step. The _N_ variable represents the number of times the DATA step has iterated. The _ERROR_ variable acts like a binary switch

whose value is 0 if no errors exist in the DATA step, or 1 if one or more errors exist. The following figure shows the Input Buffer and the program data vector after DATA step compilation.

Figure 21.2 Input Buffer and Program Data Vector



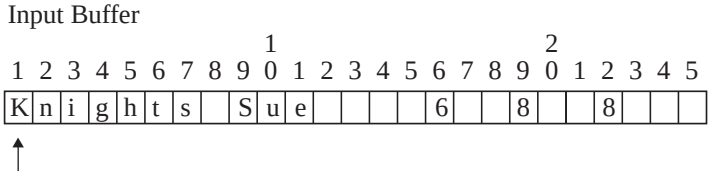
Variables that are created by the INPUT and the Sum statements (TeamName, ParticipantName, Event1, Event2, Event3, and TeamTotal) are set to missing initially. Note that in this representation, numeric variables are initialized with a period and character variables are initialized with blanks. The automatic variable _N_ is set to 1; the automatic variable _ERROR_ is set to 0.

The variable TeamName is marked Drop in the PDV because of the DROP= data set option in the DATA statement. Dropped variables are not written to the SAS data set. The _N_ and _ERROR_ variables are dropped because automatic variables created by the DATA step are not written to a SAS data set. See Chapter 10, “SAS Variables,” on page 99 for details about automatic variables.

Reading a Record

SAS reads the first data line into the input buffer. The input *pointer*, which SAS uses to keep its place as it reads data from the input buffer, is positioned at the beginning of the buffer, ready to read the data record. The following figure shows the position of the input pointer in the input buffer before SAS reads the data.

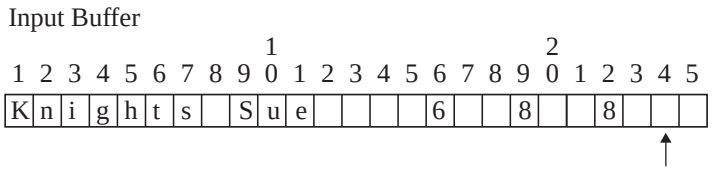
Figure 21.3 Position of the Pointer in the Input Buffer Before SAS Reads Data



The INPUT statement then reads *data values* from the record in the input buffer and writes them to the PDV where they become *variable values*. The following figure shows

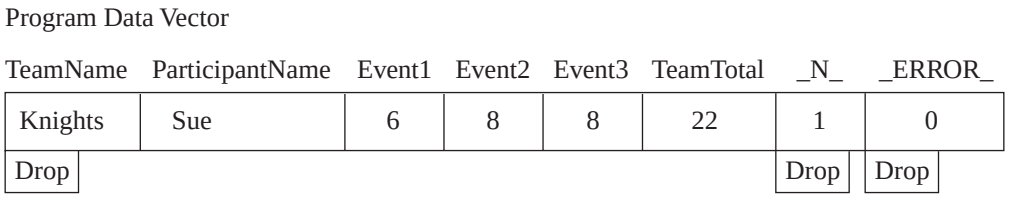
both the position of the pointer in the input buffer, and the values in the PDV after SAS reads the first record.

Figure 21.4 Values from the First Record are Read into the Program Data Vector



After the INPUT statement reads a value for each variable, SAS executes the Sum statement. SAS computes a value for the variable TeamTotal and writes it to the PDV. The following figure shows the PDV with all of its values before SAS writes the observation to the data set.

Figure 21.5 Program Data Vector with Computed Value of the Sum Statement



Writing an Observation to the SAS Data Set

When SAS executes the last statement in the DATA step, all values in the PDV, except those marked to be dropped, are written as a single observation to the data set TOTAL_POINTS. The following figure shows the first observation in the TOTAL_POINTS data set.

Figure 21.6 The First Observation in Data Set TOTAL_POINTS

Output SAS Data Set TOTAL_POINTS: 1st observation

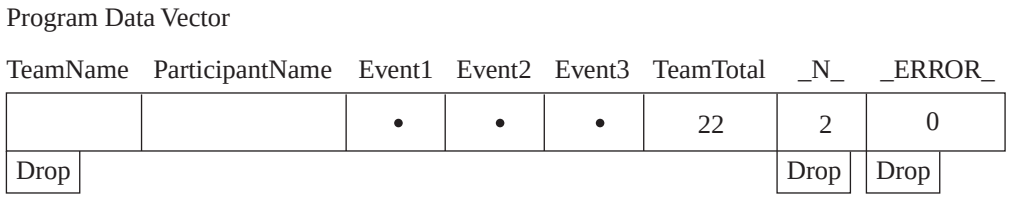
ParticipantName	Event1	Event2	Event3	TeamTotal
Sue	6	8	8	22

SAS then returns to the DATA statement to begin the next iteration. SAS resets the values in the PDV in the following way:

- The values of variables created by the INPUT statement are set to missing.
- The value created by the Sum statement is automatically retained.
- The value of the automatic variable _N_ is incremented by 1, and the value of _ERROR_ is reset to 0.

The following figure shows the current values in the PDV.

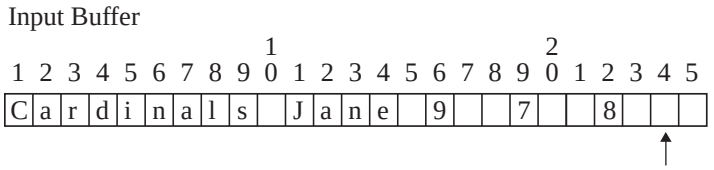
Figure 21.7 Current Values in the Program Data Vector



Reading the Next Record

SAS reads the next record into the input buffer. The INPUT statement reads the data values from the input buffer and writes them to the PDV. The Sum statement adds the values of Event1, Event2, and Event3 to TeamTotal. The value of 2 for variable _N_ indicates that SAS is beginning the second iteration of the DATA step. The following figure shows the input buffer, the PDV for the second record, and the SAS data set with the first two observations.

Figure 21.8 Input Buffer, Program Data Vector, and First Two Observations



Output SAS Data Set TOTAL_POINTS: 1st and 2nd observations

ParticipantName	Event1	Event2	Event3	TeamTotal
Sue	6	8	8	22
Jane	9	7	8	46

As SAS continues to read records, the value in TeamTotal grows larger as more participant scores are added to the variable. _N_ is incremented at the beginning of each iteration of the DATA step. This process continues until SAS reaches the end of the input file.

When the DATA Step Finishes Executing

The DATA step stops executing after it processes the last input record. You can use PROC PRINT to print the output in the TOTAL_POINTS data set:

Output 21.1 Output from the Walkthrough DATA Step

Total Team Scores 1

Obs	Participant Name	Event1	Event2	Event3	Team Total
1	Sue	6	8	8	22
2	Jane	9	7	8	46
3	John	7	7	7	67
4	Lisa	8	9	9	93
5	Fran	7	6	6	112
6	Walter	9	8	10	139

DATA Step Execution

The Default Sequence of Execution in the DATA Step

The following table outlines the default sequence of execution for statements in a DATA step. The DATA statement begins the step and identifies usually one or more SAS data sets that the step will create. (You can use the keyword _NULL_ as the data set name if you do not want to create an output data set.) Optional programming statements process your data. SAS then performs the default actions at the end of processing an observation.

Table 21.1 Default Execution for Statements in a DATA Step

Structure of a DATA Step	Action Taken
DATA statement	begins the step
	counts iterations
Data-reading statements: *	
INPUT	describes the arrangement of values in the input data record from a raw data source
SET	reads an observation from one or more SAS data sets
MERGE	joins observations from two or more SAS data sets into a single observation
MODIFY	replaces, deletes, or appends observations in an existing SAS data set in place
UPDATE	updates a master file by applying transactions
Optional SAS programming statements, for example:	further processes the data for the current observation.
FirstQuarter=Jan+Feb+Mar;	computes the value for FirstQuarter for the current observation.
if RetailPrice < 500;	subsets by value of variable RetailPrice for the current observation
Default actions at the end of processing an observation	
At end of DATA step:	writes an observation to a SAS data set
Automatic write, automatic return	returns to the DATA statement
At top of DATA step:	resets values to missing in program data vector
Automatic reset	

* The table shows the default processing of the DATA step. You can alter the sequence of statements in the DATA step. You can code optional programming statements, such as creating or reinitializing a constant, before you code a data-reading statement.

Note: You can also use functions to read and process data. For information about how statements and functions process data differently, see “Using Functions to

Manipulate Files" on page 48. For specific information about SAS functions, see the SAS Functions listed under the "SAS I/O Files" and "External Files" categories in the SAS Functions section of *SAS Language Reference: Dictionary*. △

Changing the Default Sequence of Execution

Using Statements to Change the Default Sequence of Execution

You can change the default sequence of execution to control how your program executes. SAS language statements offer you a lot of flexibility to do this in a DATA step. The following list shows the most common ways to control the flow of execution in a DATA step program.

Table 21.2 Common Methods that Alter the Sequence of Execution

When ...	you can ...
Reading a record	merge, modify, join data sets
	read multiple records to create a single observation
	randomly select records for processing
	read from multiple external files
	read selected fields from a record by using statement or data set options
Processing data	use conditional logic
	retain variable values
Writing an observation	write to a SAS data set or to an external file
	control when output is written to a data set
	write to multiple files

For more information, see the individual statements in *SAS Language Reference: Dictionary*.

Using Functions to Change the Default Sequence of Execution

You can also use functions to read and process data. For information about how statements and functions process data differently, see "Using Functions to Manipulate Files" on page 48. *SAS Language Reference: Concepts*. For specific information about SAS functions, see the SAS Functions listed under the "SAS I/O Files" and "External Files" categories in the SAS Functions section of *SAS Language Reference: Dictionary*.

Altering the Flow for a Given Observation

You can use statements, statement options, and data set options to alter the way SAS processes specific observations. The following table lists SAS language elements and their effects on processing.

Table 21.3 Language Elements that Alter Programming Flow

SAS Language Element	Function
subsetting IF statement	stops the current iteration when a condition is false, does not write the current observation to the data set, and returns control to the top of the DATA step.
IF-THEN/ELSE statement	stops the current iteration when a conditon is true, writes the current observation to the data set, and returns control to the top of the DATA step.
DO loops	cause parts of the DATA step to be executed multiple times.
LINK and RETURN statements	alter the flow of control, execute statements following the label specified, and return control of the program to the next statement following the LINK statement.
HEADER= option in the FILE statement	alters the flow of control whenever a PUT statement causes a new page of output to begin; statements following the label specified in the HEADER= option are executed until a RETURN statement is encountered, at which time control returns to the point from which the HEADER= option was activated.
GO TO statement	alters the flow of execution by branching to the label that is specified in the GO TO statement. SAS executes subsequent statements then returns control to the beginning of the DATA step.
EOF= option in an INFILE statement	alters the flow of execution when the end of the input file is reached; statements following the label that is specified in the EOF= option are executed at that time.
_N_ automatic variable in an IF-THEN construct	causes parts of the DATA step to execute only for particular iterations.
SELECT statement	conditionally executes one of a group of SAS statements.
OUTPUT statement in an IF-THEN construct	outputs an observation before the end of the DATA step, based on a condition; prevents automatic output at the bottom of the DATA step.
DELETE statement in an IF-THEN construct	deletes an observation based on a condition and causes a return to the top of the DATA step.

SAS Language Element	Function
ABORT statement in an IF-THEN construct	stops execution of the DATA step and instruct SAS to resume execution with the next DATA or PROC step. It can also stop executing a SAS program altogether, depending on the options specified in the ABORT statement and on the method of operation.
WHERE statement or WHERE= data set option	causes SAS to read certain observations based on one or more specified criteria.

Step Boundary — How To Know When Statements Take Effect

Understanding step boundaries is an important concept in SAS programming because step boundaries determine when SAS statements take effect. SAS executes program statements only when SAS crosses a default or an explicit step boundary. Consider the following DATA steps:

```
data _null_;  ❶
   set allscores(drop=score5-score7);
   title 'Student Test Scores';  ❷

data employees;  ❸
   set employee_list;
run;
```

❶ The DATA statement begins a DATA step and is a step boundary.
❷ The TITLE statement is in effect for both DATA steps because it appears before the boundary of the first DATA step. (Because the TITLE statement is a global statement,
❸ The DATA statement is the default boundary for the first DATA step.

The TITLE statement in this example is in effect for the first DATA step as well as for the second because the TITLE statement appears before the boundary of the first DATA step. This example uses the default step boundary **`data employees;`**.

The following example shows an OPTIONS statement inserted after a RUN statement.

```
data scores; ❶
   set allscores(drop=score5-score7);
run; ❷

options firstobs=5 obs=55; ❸

data test;
   set alltests;
run;
```

The OPTIONS statement specifies that the first observation that is read from the input data set should be the 5th, and the last observation that is read should be the 55th. Inserting a RUN statement immediately before the OPTIONS statement causes the first DATA step to reach its boundary (**`run;`**)before SAS encounters the OPTIONS statement. In this case, the step boundary is explicit. The OPTIONS statement settings, therefore, are put into effect for the second DATA step only.

❶ The DATA statement is a step boundary.
❷ The RUN statement is the explicit boundary for the first DATA step.
❸ The OPTIONS statement affects the second DATA step only.

Following the statements in a DATA step with a RUN statement is the simplest way to make the step begin to execute, but a RUN statement is not always necessary. SAS recognizes several step boundaries for a SAS step:

- another DATA statement
- a PROC statement
- a RUN statement.

  Note: For SAS programs executed in interactive mode, a RUN statement is required to signal the step boundary for the last step you submit. △
- the semicolon (with a DATALINES or CARDS statement) or four semicolons (with a DATALINES4 or CARDS4 statement) after data lines
- an ENDSAS statement
- in noninteractive or batch mode, the end of a program file containing SAS programming statements
- a QUIT statement (for some procedures).

When you submit a DATA step during interactive processing, it does not begin running until SAS encounters a step boundary. This fact enables you to submit statements as you write them while preventing a step from executing until you have entered all the statements.

What Causes a DATA Step to Stop Executing

DATA steps stop executing under different circumstances, depending on the type and number of sources of input.

Table 21.4 Causes that Stop DATA Step Execution

A DATA step that reads …	from …	with these statements	stops …
no data			after only one iteration
any data			when it executes STOP or ABORT
			when the data is exhausted
raw data	instream data lines	INPUT statement	after the last data line is read
	one external file	INPUT and INFILE statements	when end-of-file is reached
	multiple external files	INPUT and INFILE statements	when end-of-file is first reached on any of the files
observations sequentially	one SAS data set	SET and MODIFY statements	after the last observation is read

A DATA step that reads …	from …	with these statements	stops …
	multiple SAS data sets	one SET, MERGE, MODIFY, or UPDATE statement	when all input data sets are exhausted
	multiple SAS data sets	multiple SET, MERGE, MODIFY, or UPDATE statements	when end-of-file is reached by any of the data-reading statements

A DATA step that reads observations from a SAS data set with a SET statement that uses the POINT= option has no way to detect the end of the input SAS data set. (This method is called direct or random access.) Such a DATA step usually requires a STOP statement.

A DATA step also stops when it executes a STOP or an ABORT statement. Some system options and data set options, such as OBS=, can cause a DATA step to stop earlier than it would otherwise.

Creating a SAS Data Set with a DATA Step

Creating a SAS Data File or a SAS Data View

You can create either a SAS data file, a data set that holds actual data, or a SAS data view, a data set that references data that is stored elsewhere. By default, you create a SAS data file. To create a SAS data view instead, use the VIEW= option on the DATA statement. With a data view you can, for example, process monthly sales figures without having to edit your DATA step. Whenever you need to create output, the output from a data view reflects the current input data values.

The following DATA statement creates a data view called MONTHLY_SALES.

```
data monthly_sales / view=monthly_sales;
```

The following DATA statement creates a data file called TEST_RESULTS.

```
data test_results;
```

Sources of Input Data

You select data-reading statements based on the source of your input data. There are at least six sources of input data:

- raw data in an external file
- raw data in the jobstream (instream data)
- data in SAS data sets
- data that is created by programming statements
- data that you can remotely access through an FTP protocol, TCP/IP socket, a SAS catalog entry, or through a URL
- data that is stored in a Database Management System (DBMS) or other vendor's data files.

Usually DATA steps read input data records from only one of the first three sources of input. However, DATA steps can use a combination of some or all of the sources.

Reading Raw Data

Example 1: Reading External File Data

The components of a DATA step that produce a SAS data set from raw data stored in an external file are outlined here.

```
data weight; ❶
   infile 'your-input-file'; ❷
   input IDnumber $ Week1 Week16; ❸
   WeightLoss=Week1-Week16; ❹
run; ❺

proc print data=weight; ❻
run;  ❼
```

❶ Begin the DATA step and create a SAS data set called WEIGHT.

❷ Specify the external file that contains your data.

❸ Read a record and assign values to three variables.

❹ Calculate a value for variable WeightLoss.

❺ Execute the DATA step.

❻ Print data set WEIGHT using the PRINT procedure.

❼ Execute the PRINT procedure.

Example 2: Reading Instream Data Lines

This example reads raw data from instream data lines.

```
data weight2; ❶
   input IDnumber $ Week1 Week16; ❷
   WeightLoss2=Week1-Week16; ❸
   datalines; ❹
2477 195  163
2431 220  198
2456 173  155
2412 135  116
;                ❺

proc print data=weight2;  ❻
run;  ❼
```

❶ Begin the DATA step and create SAS data set WEIGHT2.

❷ Read a data line and assign values to three variables.

❸ Calculate a value for variable WeightLoss2.

❹ Begin the data lines.

❺ Signal end of data lines with a semicolon and execute the DATA step.

❻ Print data set WEIGHT2 using the PRINT procedure.

❼ Execute the PRINT procedure.

Example 3: Reading Instream Data Lines with Missing Values

You can also take advantage of options on the INFILE statement when you read instream data lines. This example shows the use of the MISSOVER statement option, which assigns missing values to variables for records that contain no data for those variables.

```
data weight2;
   infile datalines missover;  ❶
   input IDnumber $ Week1 Week16;
   WeightLoss2=Week1-Week16;
   datalines;  ❷
2477 195  163
2431
2456 173  155
2412 135  116
;                ❸

proc print data=weight2; ❹
run;  ❺
```

❶ Use the MISSOVER option to assign missing values to variables that do not contain values.

❷ Begin data lines.

❸ Signal end of data lines and execute the DATA step.

❹ Print data set WEIGHT2 using the PRINT procedure.

❺ Execute the PRINT procedure.

Example 4: Using Multiple Input Files in Instream Data

This example shows how to use multiple input files as instream data to your program. This example reads the records in each file and creates the ALL_ERRORS SAS data set. The program then sorts the observations by Station, and creates a sorted data set called SORTED_ERRORS. The print procedure prints the results.

```
options pageno=1 nodate linesize=60 pagesize=80;

data all_errors;
   length filelocation $ 60;
   input filelocation;  /* reads instream data */
   infile daily filevar=filelocation
               filename=daily end=done;
   do while (not done);
      input Station $ Shift $ Employee $ NumberOfFlaws;
      output;
   end;
   put 'Finished reading ' daily=;
   datalines;
. . .myfile_A. . .
. . .myfile_B. . .
. . .myfile_C. . .
;
```

```
proc sort data=all_errors out=sorted_errors;
   by Station;
run;

proc print data = sorted_errors;
   title 'Flaws Report sorted by Station';
run;
```

Output 21.2 Multiple Input Files in Instream Data

Flaws Report sorted by Station 1

Obs	Station	Shift	Employee	Number OfFlaws
1	Amherst	2	Lynne	0
2	Goshen	2	Seth	4
3	Hadley	2	Jon	3
4	Holyoke	1	Walter	0
5	Holyoke	1	Barb	3
6	Orange	2	Carol	5
7	Otis	1	Kay	0
8	Pelham	2	Mike	4
9	Stanford	1	Sam	1
10	Suffield	2	Lisa	1

Reading Data from SAS Data Sets

This example reads data from one SAS data set, generates a value for a new variable, and creates a new data set.

```
data average_loss;  ❶
   set weight;      ❷
   Percent=round((AverageLoss * 100) / Week1);  ❸
run;                                            ❹
```

❶ Begin the DATA step and create a SAS data set called AVERAGE_LOSS.

❷ Read an observation from SAS data set WEIGHT.

❸ Calculate a value for variable Percent.

❹ Execute the DATA step.

Generating Data from Programming Statements

You can create data for a SAS data set by generating observations with programming statements rather than by reading data. A DATA step that reads no input goes through only one iteration.

```
data investment;   ❶
   begin='01JAN1990'd;
   end='31DEC2009'd;
   do year=year(begin) to year(end);   ❷
```

```
      Capital+2000 + .07*(Capital+2000);
      output;     ❸
   end;
   put 'The number of DATA step iterations is '_n_;  ❹
run;     ❺

proc print data=investment;     ❻
   format Capital dollar12.2;    ❼
run;    ❽
```

❶ Begin the DATA step and create a SAS data set called INVESTMENT.

❷ Calculate a value based on a $2,000 capital investment and 7% interest each year from 1990 to 2009. Calculate variable values for one observation per iteration of the DO loop.

❸ Write each observation to data set INVESTMENT.

❹ Write a note to the SAS log proving that the DATA step iterates only once.

❺ Execute the DATA step.

❻ To see your output, print the INVESTMENT data set with the PRINT procedure.

❼ Use the FORMAT statement to write numeric values with dollar signs, commas, and decimal points.

❽ Execute the PRINT procedure.

Writing a Report with a DATA Step

Example 1: Creating a Report without Creating a Data Set

You can use a DATA step to generate a report without creating a data set by using _NULL_ in the DATA statement. This approach saves system resources because SAS does not create a data set. The report can contain both TITLE statements and FOOTNOTE statements. If you use a FOOTNOTE statement, be sure to include FOOTNOTE as an option on the FILE statement in the DATA step.

```
title1 'Budget Report';       ❶
title2 'Mid-Year Totals by Department';
footnote 'compiled by Manager,
Documentation Development Department';  ❷

data _null_;     ❸
   set budget;    ❹
   file print footnote;  ❺
   MidYearTotal=Jan+Feb+Mar+Apr+May+Jun;  ❻
   if _n_=1 then    ❼
      do;
         put @5 'Department' @30 'Mid-Year Total';
      end;
   put @7 Department @35 MidYearTotal;    ❽
run;    ❾
```

❶ Define titles.

❷ Define the footnote.

❸ Begin the DATA step. _NULL_ specifies that no data set will be created.

❹ Read one observation per iteration from data set BUDGET.

❺ Name the output file for the PUT statements and use the PRINT fileref. By default, the PRINT fileref specifies that the file will contain carriage control characters and titles. The FOOTNOTE option specifies that each page of output will contain a footnote.

❻ Calculate a value for the variable MidYearTotal on each iteration.

❼ Write variable name headings for the report on the first iteration only.

❽ Write the current values of variables Department and MidYearTotal for each iteration.

❾ Execute the DATA step.

The example above uses the FILE statement with the PRINT fileref to produce listing output. If you want to print to a file, specify a fileref or a complete file name. Use the PRINT option if you want the file to contain carriage control characters and titles. The following example shows how to use the FILE statement in this way.

```
file 'external-file' footnote print;
```

You can also use the **data _null_;** statement to write to an external file. For more information about writing to external files, see the "FILE" statement in *SAS Language Reference: Dictionary*, and the SAS documentation for your operating environment.

Example 2: Creating a Customized Report

You can create very detailed, fully customized reports by using a DATA step with PUT statements. The following example shows a customized report that contains three distinct sections: a header, a table, and a footer. It contains existing SAS variable values, constant text, and values that are calculated as the report is written.

Output 21.3 Sample of a Customized Report

Expense Report 1

Around The World Retailers

EMPLOYEE BUSINESS, TRAVEL, AND TRAINING EXPENSE REPORT

Employee Name: ALEJANDRO MARTINEZ Destination: CARY, NC Departure Date: 11JUL1999
Department: SALES & MARKETING Purpose of Trip/Activity: MARKETING TRAINING Return Date: 16JUL1999
Trip ID#: 93-0002519 Activity from: 11JUL1999
to: 16JUL1999

EXPENSE DETAIL	SUN 07/11	MON 07/12	TUE 07/13	WED 07/14	THU 07/15	FRI 07/16	SAT 07/17	TOTALS	PAID BY COMPANY	PAID BY EMPLOYEE
Lodging, Hotel	92.96	92.96	92.96	92.96	92.96			464.80	464.80	
Telephone	4.57	4.73						9.30		9.30
Personal Auto 36 miles @.28/mile	5.04					5.04		10.08		10.08
Car Rental, Taxi, Parking, Tolls		35.32	35.32	35.32	35.32	35.32		176.60	176.60	
Airlines, Bus, Train (Attach Stub)	485.00					485.00		970.00	970.00	
Dues										
Registration Fees	75.00							75.00		75.00
Other (explain below)						5.00		5.00		5.00
Tips (excluding meal tips)	3.00					3.00		6.00		6.00
Meals										
Breakfast						7.79		7.79		7.79
Lunch										
Dinner	36.00	28.63	36.00	36.00	30.00			166.63		166.63
Business Entertainment										
TOTAL EXPENSES	641.57	176.64	179.28	179.28	173.28	541.15		1891.20	1611.40	279.80

Travel Advance to Employee $0.00

Reimbursement due Employee (or ATWR) $279.80

Other: (i.e. miscellaneous expenses and/or names of employees sharing receipt.)

16JUL1999 CAR RENTAL INCLUDE $5.00 FOR GAS

APPROVED FOR PAYMENT BY: Authorizing Manager: _____ Emp. # _____

Employee Signature: _____ Emp. # 1118

Charge to Division: ATW Region: TX Dept: MKT Acct: 6003 Date: 27JUL1999

The code shown below generates the report example (you must create your own input data). It is beyond the scope of this discussion to fully explain the code that generated the report example. For a complete explanation of this example, see *SAS Guide to Report Writing: Examples*.

```
options ls=132 ps=66 pageno=1 nodate;

data travel;

  /* infile 'SAS-data-set' missover; */
  infile '/u/lirezn/input_for_concepts.dat' missover;
 input acct div $ region $ deptchg $ rptdate : date9.
     other1-other10 /
     empid empname & $char35. / dept & $char35. /
     purpose & $char35. / dest & $char35. / tripid & $char35. /
     actdate2 date9. /
     misc1 & $char75. / misc2 & $char75. / misc3 & $char75. /
     misc4 & $char75. /
     misc5 & $char75. / misc6 & $char75. / misc7 & $char75. /
     misc8 & $char75. /
     dptdate : date9. rtrndate : date9. automile permile /
     hotel1-hotel10 /
     phone1-phone10 / peraut1-peraut10 / carrnt1-carrnt10 /
     airlin1-airlin10 / dues1-dues10 / regfee1-regfee10 /
     tips1-tips10 / meals1-meals10 / bkfst1-bkfst10 /
     lunch1-lunch10 / dinner1-dinner10 / busent1-busent10 /
     total1-total10 / empadv reimburs actdate1 : date9.;
run;

proc format;
   value category 1='Lodging, Hotel'
                  2='Telephone'
                  3='Personal Auto'
                  4='Car Rental, Taxi, Parking, Tolls'
                  5='Airlines, Bus, Train (Attach Stub)'
                  6='Dues'
                  7='Registration Fees'
                  8='Other (explain below)'
                  9='Tips (excluding meal tips)'
                 10='Meals'
                 11='Breakfast'
                 12='Lunch'
                 13='Dinner'
                 14='Business Entertainment'
                 15='TOTAL EXPENSES';
   value blanks  0=' '
            other=(|8.2|);
   value $cuscore ' '='________';
   value nuscore   . ='________';
run;

data _null_;
   file print;
   title 'Expense Report';
   format rptdate actdate1 actdate2 dptdate rtrndate date9.;
   set travel;

   array expenses{15,10} hotel1-hotel10  phone1-phone10
                         peraut1-peraut10 carrnt1-carrnt10
```

```
                    airlin1-airlin10 dues1-dues10
                    regfee1-regfee10 other1-other10
                    tips1-tips10 meals1-meals10
                    bkfst1-bkfst10 lunch1-lunch10
                    dinner1-dinner10 busent1-busent10
                    total1-total10;
array misc{8} $ misc1-misc8;
array mday{7} mday1-mday7;
dptday=weekday(dptdate);
mday{dptday}=dptdate;
if dptday>1 then
   do dayofwk=1 to (dptday-1);
     mday{dayofwk}=dptdate-(dptday-dayofwk);
   end;
if dptday<7 then
   do dayofwk=(dptday+1) to 7;
      mday{dayofwk}=dptdate+(dayofwk-dptday);
   end;
if rptdate=. then rptdate="&sysdate9"d;

tripnum=substr(tripid,4,2)||'-'||substr(scan(tripid,1),6);

put // @1 'Around The World Retailers' //

       @1 'EMPLOYEE BUSINESS, TRAVEL, AND TRAINING EXPENSE REPORT' ///

       @1 'Employee Name: ' @16 empname
       @44 'Destination: ' @57 dest
       @106 'Departure Date:' @122 dptdate /

       @4 'Department: ' @16 dept
       @44 'Purpose of Trip/Activity: ' @70 purpose
       @109 'Return Date:' @122 rtrndate /

       @6  'Trip ID#: ' @16 tripnum
       @107 'Activity from:' @122 actdate1 /

       @118 'to:' @122 actdate2 //
       @1 '+-----------------------------------+--------+--------+'
          '--------+--------+--------+--------+--------+--------+' /

       @1 '|                                    |   SUN  |   MON  |'
          '   TUE  |   WED  |   THU  |   FRI  |   SAT  |        |'
          '   PAID BY  PAID BY' /

       @1 '| EXPENSE DETAIL                   '
          ' |  ' mday1 mmddyy5. ' |  ' mday2  mmddyy5.
          ' |  ' mday3 mmddyy5. ' |  ' mday4  mmddyy5.
          ' |  ' mday5 mmddyy5. ' |  ' mday6  mmddyy5.
          ' |  ' mday7 mmddyy5.
       @100 '| TOTALS |   COMPANY EMPLOYEE' ;
do i=1 to 15;

   if i=1 or i=10 or i=15 then
```

```
          put @1 '|-----------------------------------|--------|--------|'
                 '--------|--------|--------|--------|--------|--------|';
        if i=3 then
           put @1 '|' i category. @16 automile 4.0 @21 'miles @'
             @28 permile 3.2 @31 '/mile'  @37 '|' @;
           else put @1 '|' i category.  @37 '|' @;
        col=38;
        do j=1 to 10;
          if j<9 then put @col expenses{i,j} blanks8. '|' @;
             else if j=9 then put @col expenses{i,j} blanks8. @;
             else put @col expenses{i,j} blanks8.;
          col+9;
          if j=8 then col+2;
        end;
      end;
      Put @1 '+-----------------------------------+--------+--------+'
             '--------+--------+--------+--------+--------+--------+' //

          @1 'Travel Advance to Employee ..............................'
             '.................................................'
          @121 empadv dollar8.2 //

          @1 'Reimbursement due Employee (or ATWR) .....................'
             '.................................................'
          @121 reimburs dollar8.2 //

          @1 'Other: (i.e. miscellaneous expenses and/or names of '
             'employees sharing receipt.)' /;
      do j=1 to 8;
        put @1 misc{j} ;
      end;
      put / @1  'APPROVED FOR PAYMENT BY: Authorizing Manager:'
          @48 '_____________________________________________________'
          @100 'Emp. #  _______' ///

          @27 'Employee Signature:'
          @48 '_____________________________________________________'
          @100 'Emp. #  ' empid ///

          @6 'Charge to Division:' @26 div $cuscore.
          @39 'Region:'            @48 region $cuscore.
          @59 'Dept:'              @66 deptchg $cuscore.
          @79 'Acct:'              @86 acct nuscore.
          @100 'Date:'             @107 rptdate /
          _page_;
run;
```

The Output Delivery System (ODS)

The Output Delivery System (ODS) is a method of delivering output in a variety of formats and making these formats easy to access. ODS provides templates that define the structure of the output from DATA steps and from PROC steps. The DATA step allows you to use the ODS option in a FILE statement and in a PUT statement.

ODS combines raw data with one or more templates to produce several types of output called output objects. Output objects are sent to "destinations" such as the output destination, the listing destination, the printer destination, or Hypertext Markup Language (HTML). For a discussion of basic ODS concepts and examples, see "Creating Output Using the Output Delivery System (ODS)" on page 206. For complete information about ODS, see *The Complete Guide to the SAS Output Delivery System*.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Language Reference: Concepts*, Cary, NC: SAS Institute Inc., 1999. 554 pages.

SAS Language Reference: Concepts

ISBN 1–58025–441–1

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, November 1999