SAS Certification Prep Guide: Base Programming for SAS 9, Third Edition

by SAS Institute

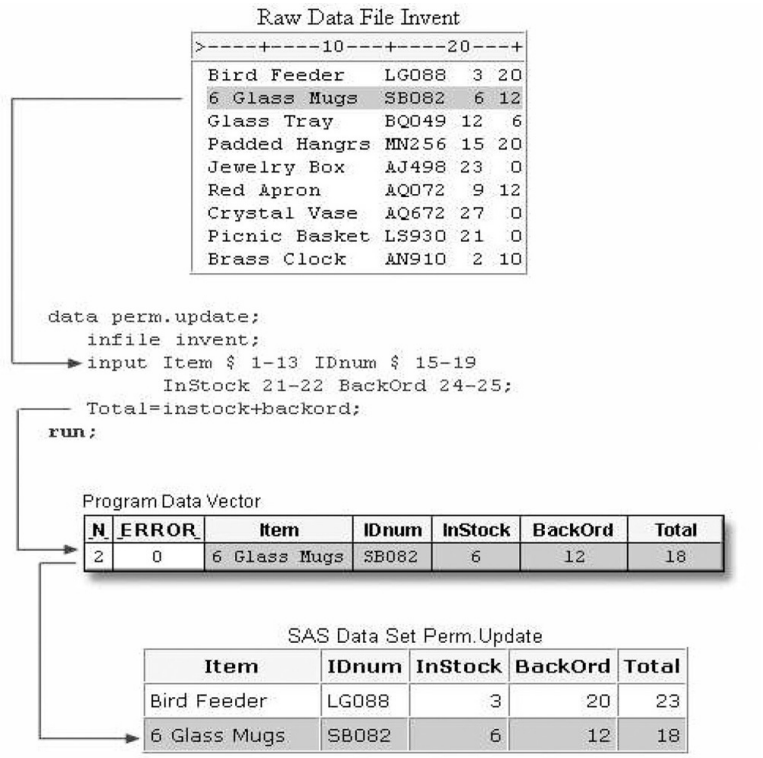
Chapter 6: Understanding DATA Step Processing

Overview

Introduction

"Creating SAS Data Sets from External Files" on page 152 explained how to read data, perform basic modifications, and create a new SAS data set.

This chapter teaches you what happens *behind the scenes* when the DATA step reads raw data. You'll examine the program data vector, which is a logical framework that SAS uses when creating SAS data sets.



Understanding how the program operates can help you to anticipate how variables will be created and processed, to plan your modifications, and to interpret and debug program errors. It also gives you useful strategies for preventing and correcting common DATA step errors.

Objectives

In this chapter, you learn to

- identify the two phases that occur when a DATA step is processed
- interpret automatic variables
- identify the processing phase in which an error occurs
- debug SAS DATA steps
- validate and clean invalid data
- test programs by limiting the number of observations that are created
- flag errors in the SAS log.

Writing Basic DATA Steps

"Creating SAS Data Sets from External Files" on page 152 explained how to write a DATA step to create a permanent SAS data set from raw data that is stored in an external file.

```
data clinic.stress;
   infile tests;
   input ID 1-4 Name $ 6-25 RestHR 27-29
         MaxHR 31-33 RecHR 35-37
         TimeMin 39-40 TimeSec 42-43
         Tolerance $ 45;
run;
```

Partial Raw Data File Tests

```
1---+----10---+----20---+----30---+----40---+-
2458 Murray, W             72   185 128 12 38 D
2462 Almers, C             68   171 133 10  5 I
2501 Bonaventure, T        78   177 139 11 13 I
2523 Johnson, R            69   162 114  9 42 S
2539 LaMance, K            75   168 141 11 46 D
2544 Jones, M              79   187 136 12 26 N
2552 Reberson, P           69   158 139 15 41 D
2555 King, E               70   167 122 13 13 I
```

You learned how to submit the DATA step and how to check the log to see whether the step ran successfully.

```
NOTE: The infile TESTS is:
      Filename=c:\rawdata\tests,
      RECFM=V,LRECL=256,File Size (bytes)=376,
      Last Modified=16May2011:11:23:09,
      Create Time=17May2011:07:42:32

NOTE: 8 records were read from the infile TESTS.
      The minimum record length was 45.
      The maximum record length was 45.
NOTE: The data set CLINIC.STRESS has 8 observations and 8 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

You also learned how to display the contents of the data set with the PRINT procedure.

```
proc print data=clinic.stress;
run;
```

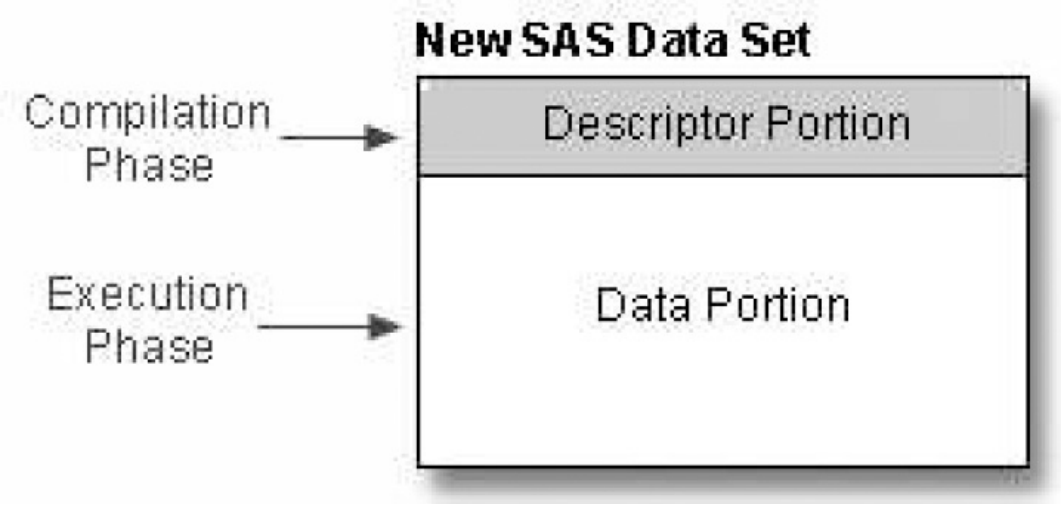
Obs	ID	Name	RestHi	MaxHR	RecHR	TimeMin	TimeSec	Tolerance
1	2458	Murray, W	72	185	128	12	38	D
2	2462	Almers, C	68	171	133	10	5	I
3	2501	Bonaventure, T	78	177	139	11	13	I
4	2523	Johnson, R	69	162	114	9	42	S
5	2539	LaMance, K	75	168	141	11	46	D
6	2544	Jones, M	79	187	136	12	26	N
7	2552	Reberson, P	69	158	139	15	41	D
8	2555	King, E	70	167	122	13	13	I

Figure 6.1: Output from the PRINT Procedure

How SAS Processes Programs

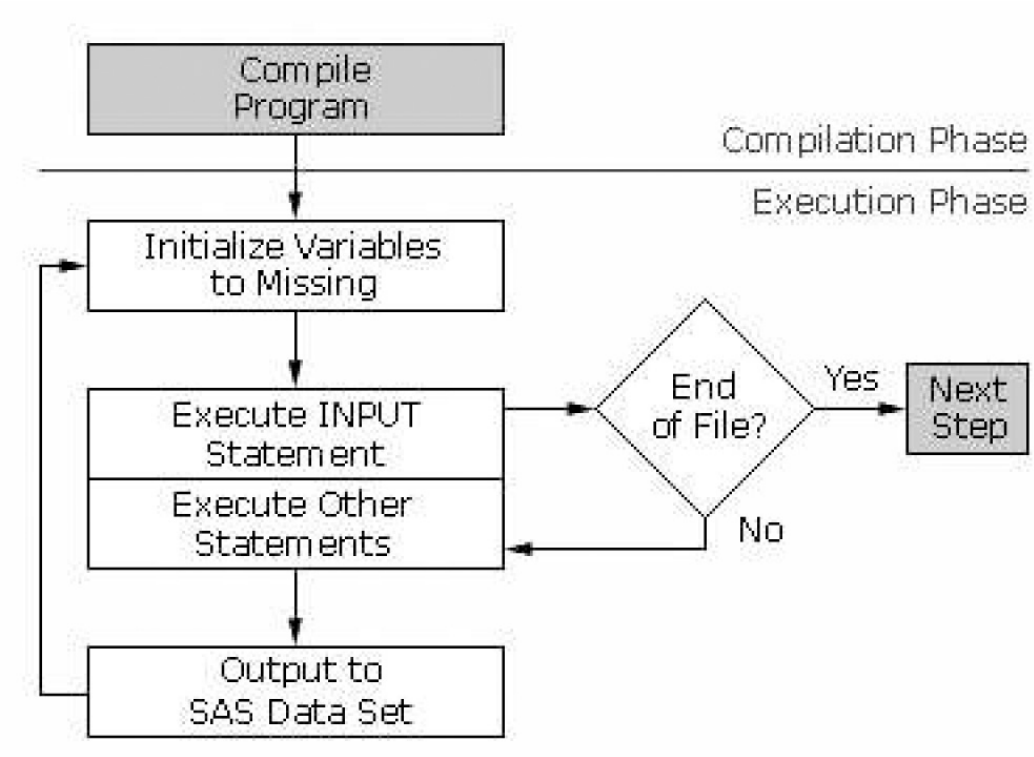
When you submit a DATA step, SAS processes the DATA step and then creates a new SAS data set. Let's see exactly how that happens.

A SAS DATA step is processed in two phases:



- During the compilation phase, each statement is scanned for syntax errors. Most syntax errors prevent further processing of the DATA step. When the compilation phase is complete, the descriptor portion of the new data set is created.
- If the DATA step compiles successfully, then the execution phase begins. During the execution phase, the DATA step reads and processes the input data. The DATA step executes once for each record in the input file, unless otherwise directed.

The diagram below shows the general flow of DATA step processing for reading raw data. We'll examine both the compilation phase and the execution phase in this chapter.

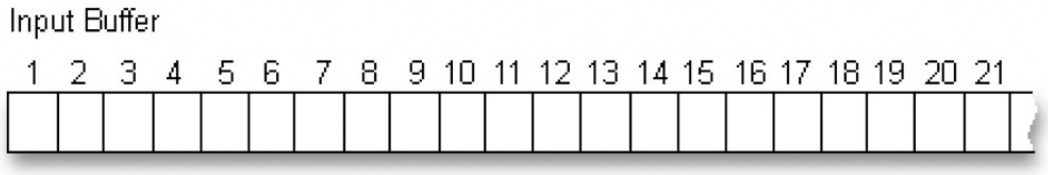


Let's start with the compilation phase.

Compilation Phase

Input Buffer

At the beginning of the compilation phase, the input buffer (an area of memory) is created to hold a record from the external file. The input buffer is created only when raw data is read, not when a SAS data set is read.

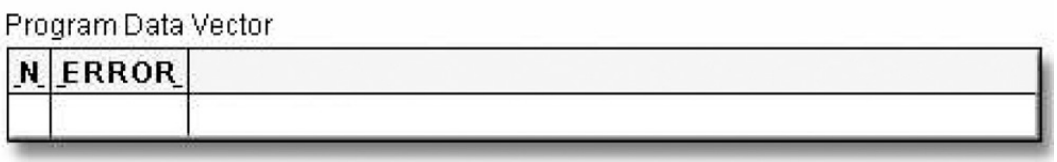


Program Data Vector

After the input buffer is created, the program data vector is created. The program data vector is the area of memory where SAS holds one observation at a time.

The program data vector contains two automatic variables that can be used for processing but which are not written to the data set as part of an observation.

- _N_ counts the number of times that the DATA step begins to execute.
- _ERROR_ signals the occurrence of an error that is caused by the data during execution. The default value is 0, which means there is no error. When one or more errors occur, the value is set to 1.



Syntax Checking

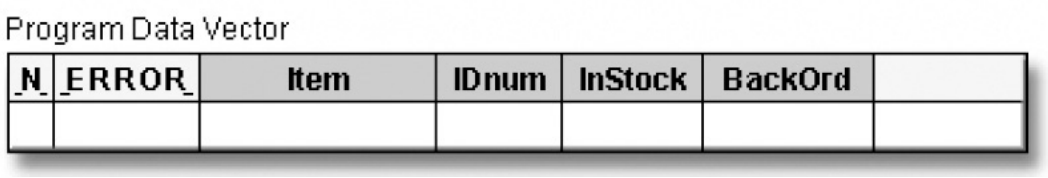
During the compilation phase, SAS also scans each statement in the DATA step, looking for syntax errors. Syntax errors include

- missing or misspelled keywords
- invalid variable names
- missing or invalid punctuation
- invalid options.

Data Set Variables

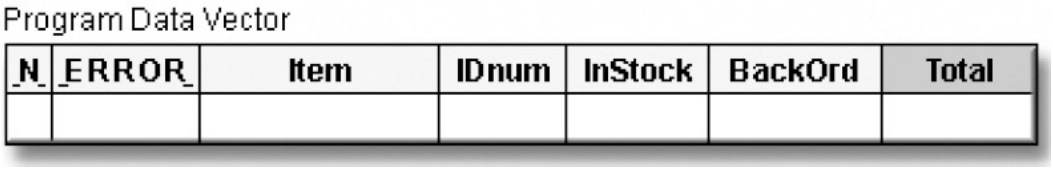
As the INPUT statement is compiled, a slot is added to the program data vector for each variable in the new data set. Generally, variable attributes such as length and type are determined the first time a variable is encountered.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
    Total=instock+backord;
run;
```



Any variables that are created with an assignment statement in the DATA step are also added to the program data vector. For example, the assignment statement below creates the variable Total. As the statement is compiled, the variable is added to the program data vector. The attributes of the variable are determined by the expression in the statement. Because the expression contains an arithmetic operator and produces a numeric value, Total is defined as a numeric variable and is assigned the default length of 8.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```



Descriptor Portion of the SAS Data Set

At the bottom of the DATA step (in this example, when the RUN statement is encountered), the compilation phase is complete, and the descriptor portion of the new SAS data set is created. The descriptor portion of the data set includes

- the name of the data set
- the number of variables
- the names and attributes of the variables.

```
                                The SAS System
                            The CONTENTS Procedure

Data Set Name        PERM.UPDATE                              Observations          9
Member Type          DATA                                     Variables             5
Engine               V9                                       Indexes               0
Created              Thursday, May 05, 2011 03:38:15 PM       Observation Length    48
Last Modified        Thursday, May 05, 2011 03:38:15 PM       Deleted Observations  0
Protection                                                    Compressed            NO
Data Set Type                                                Sorted                NO
Label
Data Representation  WINDOWS_32
Encoding             wlatin1  Western (Windows)


                     Engine/Host Dependent Information

Data Set Page Size          4096
Number of Data Set Pages    1
First Data Page             1
Max Obs per Page            84
Obs in First Data Page      9
Number of Data Set Repairs  0
Filename                    C:\Documents and Settings\user-name\sasuser\update.sas7bdat
Release Created             9.0301M0
Host Created                XP_PRO


                  Alphabetic List of Variables and Attributes

                #    Variable    Type    Len

                4    BackOrd     Num       8
                2    IDnum       Char      5
                3    InStock     Num       8
                1    Item        Char     13
                5    Total       Num       8
```

At this point, the data set contains the five variables that are defined in the input data set and in the assignment statement. Remember, _N_ and _ERROR_ are not written to the data set. There are no observations because the DATA step has not yet executed. During execution, each raw data record is processed and is then written to the data set as an observation.

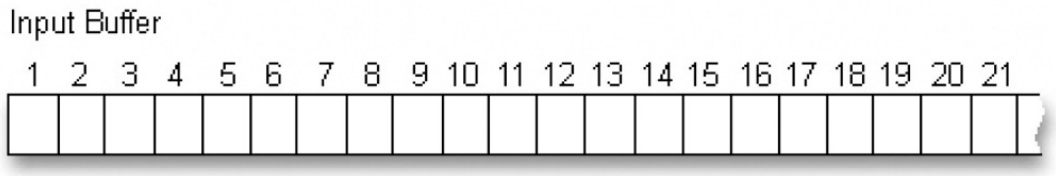
Additional Note For additional information about assigning attributes to variables, see the SAS documentation.

Summary of the Compilation Phase

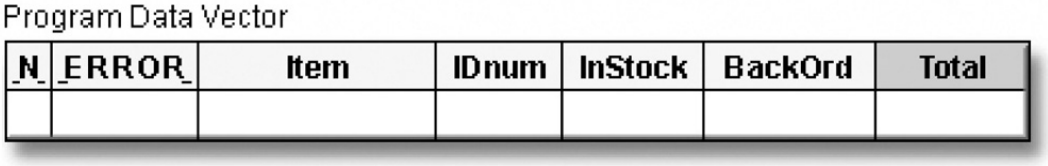
Let's review the compilation phase.

```
data perm.update;
     infile invent;
     input Item $ 1-13 IDnum $ 15-19
           InStock 21-22 BackOrd 24-25;
     Total=instock+backord;
run;
```

During the compilation phase, the input buffer is created to hold a record from the external file.



The program data vector is created to hold the current observation.



The descriptor portion of the SAS data set is created.

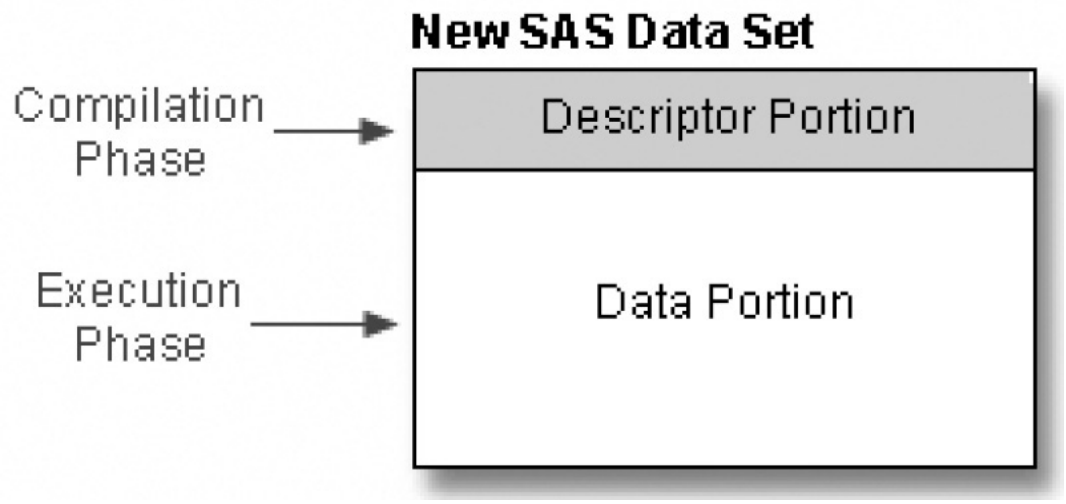
Data Set Descriptor (Partial)

```
Data Set Name:        PERM.UPDATE
Member Type:          DATA
Engine:               V9
Created:              Tuesday,May 03, 2011 07:26:52 AM
Observations:         9
Variables:            5
Indexes:              0
Observation Length:   48
```

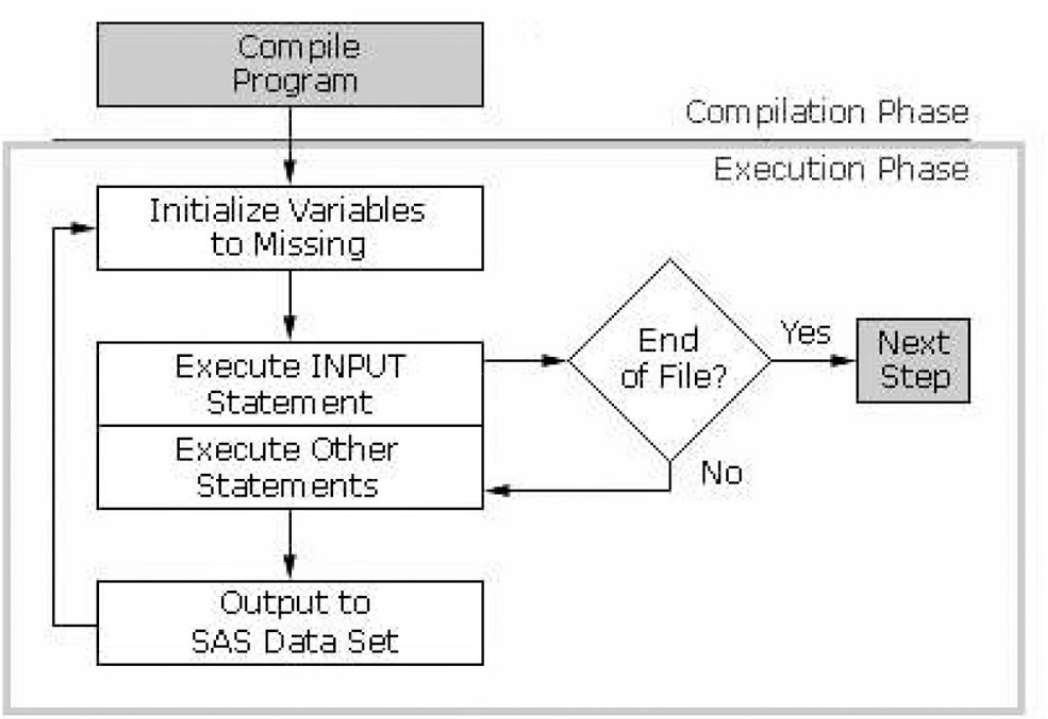
Execution Phase

Overview

After the DATA step is compiled, it is ready for execution. During the execution phase, the data portion of the data set is created. The data portion contains the data values.



During execution, each record in the input raw data file is read into the input buffer, copied to the program data vector, and then written to the new data set as an observation. The DATA step executes once for each record in the input file, unless otherwise directed by additional statements.



Example

The following DATA step reads values from the file Invent and executes nine times because there are nine records in the file.

```
data perm.update;
 infile invent;
 input Item $ 1-13 IDnum $ 15-19
   InStock 21-22 BackOrd 24-25;
 Total=instock+backord;
run;
```

Raw Data File Invent

```
1---+----10---+----20---+-
Bird Feeder    LG088  3 20
6 Glass Mugs   SB082  6 12
Glass Tray     BQ049 12  6
Padded Hangrs  MN256 15 20
Jewelry Box    AJ498 23  0
Red Apron      AQ072  9 12
Crystal Vase   AQ672 27  0
Picnic Basket  LS930 21  0
Brass Clock    AN910  2 10
```

Initializing Variables

At the beginning of the execution phase, the value of _N_ is 1. Because there are no data errors, the value of _ERROR_ is 0.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0			.	.	.

Initialized to Missing

The remaining variables are initialized to missing. Missing numeric values are represented by periods, and missing character values are represented by blanks.

INFILE Statement

Next, the INFILE statement identifies the location of the raw data.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

Raw Data File Invent

```
1---+----10---+----20---+-
Bird Feeder    LG088  3 20
6 Glass Mugs   SB082  6 12
Glass Tray     BQ049 12  6
Padded Hangrs  MN256 15 20
Jewelry Box    AJ498 23  0
Red Apron      AQ072  9 12
Crystal Vase   AQ672 27  0
Picnic Basket  LS930 21  0
Brass Clock    AN910  2 10
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0			.	.	.

INPUT Statement

Next, the INPUT statement reads a record into the input buffer. Then, the raw data in columns 1-13 is read and is assigned to Item in the program data vector.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

Raw Data File Invent

```
1---+----10---+----20---+-
Bird Feeder    LG088  3 20
6 Glass Mugs   SB082  6 12
Glass Tray     BQ049 12  6
Padded Hangrs  MN256 15 20
Jewelry Box    AJ498 23  0
Red Apron      AQ072  9 12
Crystal Vase   AQ672 27  0
Picnic Basket  LS930 21  0
Brass Clock    AN910  2 10
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0	Bird Feeder		.	.	.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

Raw Data File Invent

```
1---+----10--V+----20---+-
Bird Feeder  •LG088   3 20
6 Glass Mugs  SB082   6 12
Glass Tray    BQ049  12  6
Padded Hangrs MN256  15 20
Jewelry Box   AJ498  23  0
Red Apron     AQ072   9 12
Crystal Vase  AQ672  27  0
Picnic Basket LS930  21  0
Brass Clock   AN910   2 10
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0	Bird Feeder		•	•	•

Next, the data in columns 15-19 is read and is assigned to IDnum in the program data vector.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

Raw Data File Invent

```
1---+----10---+----V0---+-
Bird Feeder    LG088 • 3 20
6 Glass Mugs   SB082   6 12
Glass Tray     BQ049  12  6
Padded Hangrs  MN256  15 20
Jewelry Box    AJ498  23  0
Red Apron      AQ072   9 12
Crystal Vase   AQ672  27  0
Picnic Basket  LS930  21  0
Brass Clock    AN910   2 10
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0	Bird Feeder	LG088	•	•	•

Likewise, the INPUT statement reads the values for InStock from columns 21-22, and it reads the values for BackOrd from columns 24-25.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

Raw Data File Invent

```
1---+----10---+----20---+V
Bird Feeder    LG088  3  20•
6 Glass Mugs   SB082  6  12
Glass Tray     BQ049 12   6
Padded Hangrs  MN256 15  20
Jewelry Box    AJ498 23   0
Red Apron      AQ072  9  12
Crystal Vase   AQ672 27   0
Picnic Basket  LS930 21   0
Brass Clock    AN910  2  10
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0	Bird Feeder	LG088	3	20	•

Next, the assignment statement executes. The values for InStock and BackOrd are added to produce the values for Total.

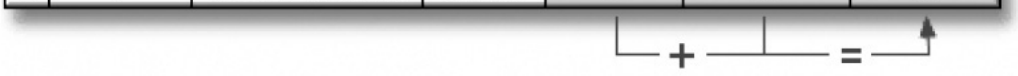
```
data perm.update;
  infile invent;
  input Item $ 1-13 IDnum $ 15-19
        InStock 21-22 BackOrd 24-25;
  Total=instock+backord;
run;
```

Raw Data File Invent

```
1---+----10---+----20---+V
Bird Feeder    LG088   3 20
6 Glass Mugs   SB082   6 12
Glass Tray     BQ049  12  6
Padded Hangrs  MN256  15 20
Jewelry Box    AJ498  23  0
Red Apron      AQ072   9 12
Crystal Vase   AQ672  27  0
Picnic Basket  LS930  21  0
Brass Clock    AN910   2 10
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0	Bird Feeder	LG088	3	20	23



End of the DATA Step

At the end of the DATA step, several actions occur. First, the values in the program data vector are written to the output data set as the first observation.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

Raw Data File Invent

```
1---+----10---+----20---+V
Bird Feeder    LG088   3 20
6 Glass Mugs   SB082   6 12
Glass Tray     BQ049  12  6
Padded Hangrs  MN256  15 20
Jewelry Box    AJ498  23  0
Red Apron      AQ072   9 12
Crystal Vase   AQ672  27  0
Picnic Basket  LS930  21  0
Brass Clock    AN910   2 10
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0	Bird Feeder	LG088	3	20	23

SAS Data Set Perm.Update

Item	IDnum	InStock	BackOrd	Total
Bird Feeder	LG088	3	20	23

Next, control returns to the top of the DATA step and the value of _N_ increments from 1 to 2. Finally, the variable values in the program data vector are re-set to missing. Notice that the automatic variable _ERROR_ is reset to zero if necessary.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

Raw Data File Invent

```
V---+----10---+----20---+-
Bird Feeder    LG088   3 20
6 Glass Mugs   SB082   6 12
Glass Tray     BQ049  12  6
Padded Hangrs  MN256  15 20
Jewelry Box    AJ498  23  0
Red Apron      AQ072   9 12
Crystal Vase   AQ672  27  0
Picnic Basket  LS930  21  0
Brass Clock    AN910   2 10
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
2	0			•	•	•

Set to Missing

SAS Data Set Perm.Update

Item	IDnum	InStock	BackOrd	Total
Bird Feeder	LG088	3	20	23

Iterations of the DATA Step

You can see that the DATA step works like a loop, repetitively executing statements to read data values and create observations one by one. Each loop (or cycle of execution) is called an iteration. At the beginning of the second iteration, the value of _N_ is 2, and _ERROR_ is still 0. Notice that the input pointer points to the second record.

```
data perm.update;
     infile invent;
     inpute Item $ 1-13 IDnum $ 15-19
               InStock 21-22 BackOrd 24-25;
     Total=instock+backord;
run;
```

Raw Data File Invent

```
V---+----10---+----20---+-
Bird Feeder    LG088  3 20
6 Glass Mugs   SB082  6 12
Glass Tray     BQ049 12  6
Padded Hangrs  MN256 15 20
Jewelry Box    AJ498 23  0
Red Apron      AQ072  9 12
Crystal Vase   AQ672 27  0
Picnic Basket  LS930 21  0
Brass Clock    AN910  2 10
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
2	0			.	.	.

SAS Data Set Perm.Update

Item	IDnum	InStock	BackOrd	Total
Bird Feeder	LG088	3	20	23

As the INPUT statement executes for the second time, the values from the second record are read into the input buffer and then into the program data vector.

Raw Data File Invent

```
1---+----10---+----20---+V
Bird Feeder    LG088  3 20
6 Glass Mugs   SB082  6 12
Glass Tray     BQ049 12  6
Padded Hangrs  MN256 15 20
Jewelry Box    AJ498 23  0
Red Apron      AQ072  9 12
Crystal Vase   AQ672 27  0
Picnic Basket  LS930 21  0
Brass Clock    AN910  2 10
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
2	0	6 Glass Mugs	SB082	6	12	

SAS Data Set Perm.Update

Item	IDnum	InStock	BackOrd	Total
Bird Feeder	LG088	3	20	23

Next, the value for Total is calculated based on the current values for InStock and BackOrd. The RUN statement indicates the end of the DATA step loop.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

Raw Data File Invent

```
1---+----10---+----20---+V
Bird Feeder    LG088   3 20
6 Glass Mugs   SB082   6 12
Glass Tray     BQ049  12  6
Padded Hangrs  MN256  15 20
Jewelry Box    AJ498  23  0
Red Apron      AQ072   9 12
Crystal Vase   AQ672  27  0
Picnic Basket  LS930  21  0
Brass Clock    AN910   2 10
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
2	0	6 Glass Mugs	SB082	6	12	18

SAS Data Set Perm.Update

Item	IDnum	InStock	BackOrd	Total
Bird Feeder	LG088	3	20	23

At the bottom of the DATA step, the values in the program data vector are written to the data set as the second observation.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

Next, the value of _N_ increments from 2 to 3, control returns to the top of the DATA step, and the values for Item, IDnum, InStock, BackOrd, and Total are re-set to missing.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

Raw Data File Invent

```
V---+----10---+----20---+-
Bird Feeder    LG088  3 20
6 Glass Mugs   SB082  6 12
Glass Tray     BQ049 12  6
Padded Hangrs  MN256 15 20
Jewelry Box    AJ498 23  0
Red Apron      AQ072  9 12
Crystal Vase   AQ672 27  0
Picnic Basket  LS930 21  0
Brass Clock    AN910  2 10
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
3	0			•	•	•

Reset to Missing

SAS Data Set Perm.Update

Item	IDnum	InStock	BackOrd	Total
Bird Feeder	LG088	3	20	23
6 Glass Mugs	SB082	6	12	18

End-of-File Marker

The execution phase continues in this manner until the end-of-file marker is reached in the raw data file. When there are no more records in the raw data file to be read, the data portion of the new data set is complete and the DATA step stops.

Raw Data File Invent

```
1---+----10---+----20---+-
Bird Feeder    LG088  3 20
6 Glass Mugs   SB082  6 12
Glass Tray     BQ049 12  6
Padded Hangrs  MN256 15 20
Jewelry Box    AJ498 23  0
Red Apron      AQ072  9 12
Crystal Vase   AQ672 27  0
Picnic Basket  LS930 21  0
Brass Clock    AN910  2 10
```

This is the output data set that SAS creates:

SAS Data Set Perm.Udpate

Item	IDnum	InStock	BackOrd	Total
Bird Feeder	LG088	3	20	23
6 Glass Mugs	SB082	6	12	18
Glass Tray	BQ049	12	6	18
Padded Hangrs	MN256	15	20	35
Jewelry Box	AJ498	23	0	23
Red Apron	AQ072	9	12	21
Crystal Vase	AQ672	27	0	27
Picnic Basket	LS930	21	0	21
Brass Clock	AN910	2	10	12

Additional Note When reading variables from raw data, SAS sets the value of each variable in the DATA step to missing at the beginning of each cycle of execution, with these exceptions:

- variables that are named in a RETAIN statement
- variables that are created in a sum statement

- data elements in a _TEMPORARY_ array
- any variables that are created with options in the FILE or INFILE statements
- automatic variables.

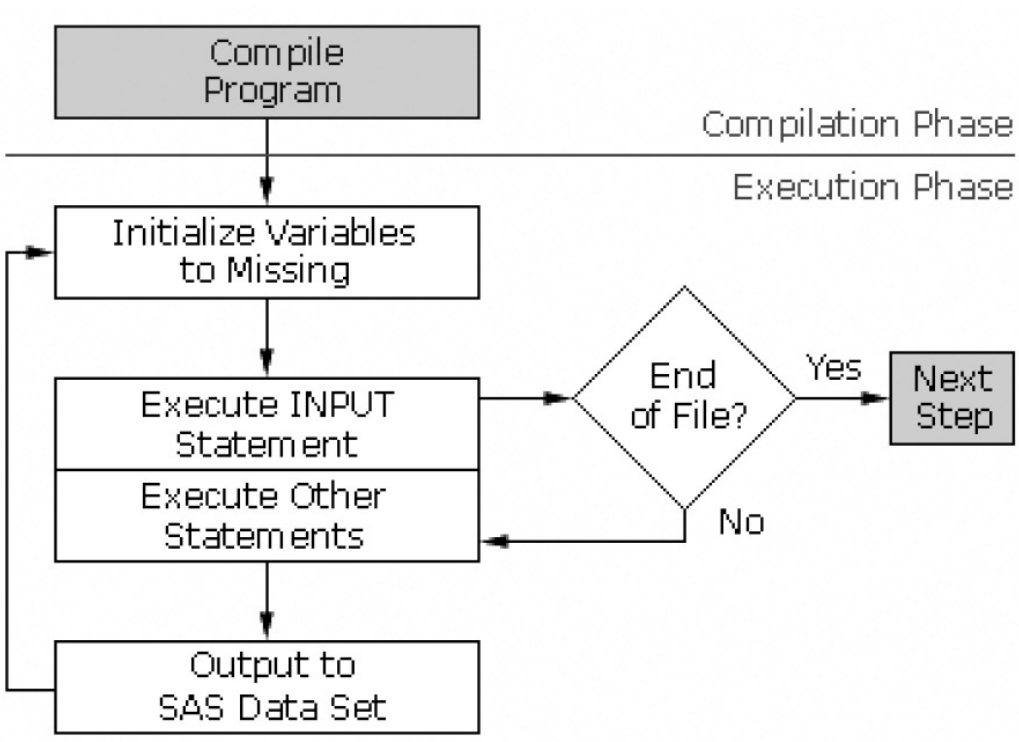
In contrast, when reading variables from a SAS data set, SAS sets the values to missing only before the first cycle of execution of the DATA step. Therefore, the variables retain their values until new values become available (for example, through an assignment statement or through the next execution of a SET or MERGE statement). Variables that are created with options in the SET or MERGE statements also retain their values from one cycle of execution to the next. (You learn about reading SAS data sets, working with arrays, and using the SET and MERGE statements in later chapters.)

Summary of the Execution Phase

You've seen how the DATA step iteratively reads records in the raw data file. Now take a minute to review execution-phase processing.

During the execution phase

- variables in the program data vector are initialized to missing before each execution of the DATA step
- each statement is executed sequentially
- the INPUT statement reads into the input buffer the next record from the external file identified by the INFILE statement, and then reads the data into the program data vector
- other statements can then further modify the current observation
- the values in the program data vector are written to the SAS data set at the end of the DATA step
- program flow returns to the top of the DATA step
- the DATA step is executed until the end-of-file marker is reached in the external file.



End of the Execution Phase

At the end of the execution phase, the SAS log confirms that the raw data file was read, and it displays the number of

observations and variables in the data set.

SAS Log

```
NOTE: 9 records were read from the infile INVENT.
NOTE: The data set PERM.UPDATE has 9 observations
      and 5 variables.
```

You already know how to display the data set with the PRINT procedure.

```
proc print data=perm.update;
run;
```

Obs	Item	IDnum	InStock	BackOrd	Total
1	Bird Feeder	LG088	3	20	23
2	6 Glass Mugs	SB082	6	12	18
3	Glass Tray	BQ049	12	6	18
4	Padded Hangrs	MN256	15	20	35
5	Jewelry Box	AJ498	23	0	23
6	Red Apron	AQ072	9	12	21
7	Crystal Vase	AQ672	27	0	27
8	Picnic Basket	LS930	21	0	21
9	Brass Clock	AN910	2	10	12

Figure 6.2: Output from the PRINT Procedure

Debugging a DATA Step

Diagnosing Errors in the Compilation Phase

Now that you know how a DATA step is processed, you can use that knowledge to correct errors. Chapter 3, "Editing and Debugging SAS Programs," on page 75 provided examples of errors that are detected during the compilation phase, including

- misspelled keywords and data set names
- unbalanced quotation marks
- invalid options.

During the compilation phase, SAS can interpret some syntax errors (such as the keyword DATA misspelled as DAAT). If it cannot interpret the error, SAS

- prints the word ERROR followed by an error message in the log
- compiles but does not execute the step where the error occurred, and prints the following message to warn you:
  `NOTE: The SAS System stopped processing this step because of errors.`

Some errors are explained fully by the message that SAS prints; other error messages are not as easy to interpret. For example, because SAS statements are free-format, when you fail to end a SAS statement with a semicolon, SAS cannot detect the error.

Diagnosing Errors in the Execution Phase

As you have seen, errors can occur in the compilation phase, resulting in a DATA step that is compiled but not executed.

Errors can also occur during the execution phase. When SAS detects an error in the execution phase, the following can occur, depending on the type of error:

- A note, warning, or error message is displayed in the log.
- The values that are stored in the program data vector are displayed in the log.
- The processing of the step either continues or stops.

Example

Suppose you misspelled the fileref in the INFILE statement below. This is not a syntax error, because SAS does not validate the file that you reference until the execution phase. During the compilation phase, the fileref Invnt is assumed to reference some external raw data file.

```
data perm.update;
   infile invnt;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

This error is not detected until the execution phase begins. Because there is no external file that is referenced by the fileref Invnt, the DATA step stops processing.

```
3486  data perm.update;
3487     infile invnt;
3488     input Item $ 1-13 IDnum $ 15-19 InStock 21-22 BackOrd 24-25;
3489     Total=instock+backord;
3490
3491  run;

ERROR: No logical assign for filename INVNT.
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set  PERM.UPDATE may be incomplete.  When this step was
         stopped there were 0 observations and 5 variables.
WARNING: Data set PERM.UPDATE was not replaced because this step was stopped.
         stopped.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Because Invent is misspelled, the statement in the DATA step that identifies the raw data is incorrect. Note, however, that the correct number of variables was defined in the descriptor portion of the data set.

Incorrectly identifying a variable's type is another common execution-time error. As you know, the values for IDnum are character values. Suppose you forget to place the dollar sign ($) after the variable's name in your INPUT statement. This is not a compile-time error, because SAS cannot verify IDnum's type until the data values for IDnum are read.

Raw Data File Invent

```
----+----10---+----20---+-
Bird Feeder    LG088  3 20
6 Glass Mugs   SB082  6 12
Glass Tray     BQ049 12  6
Padded Hangrs  MN256 15 20
Jewelry Box    AJ498 23  0
Red Apron      AQ072  9 12
Crystal Vase   AQ672 27  0
Picnic Basket  LS930 21  0
Brass Clock    AN910  2 10
```

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

In this case, the DATA step completes the execution phase, and the observations are written to the data set. However, several notes appear in the log.

SAS Log

```
NOTE: Invalid data for IDnum in line 7 15-19.
RULE: ----+----1----+----2----+----3----+----4--
07    Crystal Vase  AQ672 27 0
Item=Crystal Vase IDnum=. InStock=27 BackOrd=0
Total=27 _ERROR_=1 _N_=7
NOTE: Invalid data for IDnum in line 8 15-19.
08    Picnic Basket LS930 21 0
Item=Picnic Basket IDnum=. InStock=21 BackOrd=0
Total=21 _ERROR_=1 _N_=8
NOTE: Invalid data for IDnum in line 9 15-19.
09    Brass Clock   AN910 2 10
Item=Brass Clock IDnum=. InStock=2 BackOrd=10
Total=12 _ERROR_=1 _N_=9

NOTE: 9 records were read from the infile INVENT.
NOTE: The data set PERM.UPDATE has 9 observations
      and 5 variables.
```

Each note identifies the location of the invalid data for each observation. In this example, the invalid data is located in columns 15-19 for all observations.

The second line in each note (excluding the RULE line) displays the raw data record. Notice that the second field displays the values for IDnum, which are obviously character values.

SAS Log

```
NOTE: Invalid data for IDnum in line 7 15-19.
RULE: ----+----1----+----2----+----3----+----4--
07    Crystal Vase  AQ672 27 0
Item=Crystal Vase IDnum=. InStock=27 BackOrd=0
Total=27 _ERROR_=1 _N_=7
NOTE: Invalid data for IDnum in line 8 15-19.
08    Picnic Basket LS930 21 0
Item=Picnic Basket IDnum=. InStock=21 BackOrd=0
Total=21 _ERROR_=1 _N_=8
NOTE: Invalid data for IDnum in line 9 15-19.
09    Brass Clock   AN910 2 10
Item=Brass Clock IDnum=. InStock=2 BackOrd=10
Total=12 _ERROR_=1 _N_=9

NOTE: 9 records were read from the infile INVENT.
NOTE: The data set PERM.UPDATE has 9 observations
      and 5 variables.
```

The third and fourth lines display the values that are stored in the program data vector. Here, the values for IDnum are missing, although the other values have been correctly assigned to their respective variables. Notice that _ERROR_ has a value of **1**, indicating that a data error has occurred.

SAS Log

```
NOTE: Invalid data for IDnum in line 7 15-19.
RULE: ----+----1----+----2----+----3----+----4--
07    Crystal Vase  AQ672 27 0
Item=Crystal Vase IDnum=. InStock=27 BackOrd=0
Total=27 _ERROR_=1 _N_=7
NOTE: Invalid data for IDnum in line 8 15-19.
08    Picnic Basket LS930 21 0
Item=Picnic Basket IDnum=. InStock=21 BackOrd=0
Total=21 _ERROR_=1 _N_=8
NOTE: Invalid data for IDnum in line 9 15-19.
09    Brass Clock   AN910 2 10
Item=Brass Clock IDnum=. InStock=2 BackOrd=10
Total=12 _ERROR_=1 _N_=9

NOTE: 9 records were read from the infile INVENT.
NOTE: The data set PERM.UPDATE has 9 observations
      and 5 variables.
```

The PRINT procedure displays the data set, showing that the values for IDnum are missing. In this example, the periods indicate that IDnum is a numeric variable, although it should have been defined as a character variable.

```
proc print data=perm.update;
run;
```

Obs	Item	IDnum	InStock	BackOrd	Total
1	Bird Feeder	.	3	20	23
2	6 Glass Mugs	.	6	12	18
3	Glass Tray	.	12	6	18
4	Padded Hangrs	.	15	20	35
5	Jewelry Box	.	23	0	23
8	Red Apron	.	9	12	21
7	Crystal Vase	.	27	0	27
8	Picnic Basket	.	21	0	21
9	Brass Clock	.	2	10	12

Figure 6.3: Output from the PRINT Procedure Showing Missing Values for IDnum

When you read raw data with the DATA step, it's important to check the SAS log to verify that your data was read correctly. Here is a typical message:

SAS Log

```
WARNING: The data set PERM.UPDATE may be incomplete.
         When this step was stopped there were
         0 observations and 5 variables.
```

When no observations are written to the data set, you should check to see whether your DATA step was completely

executed. Most likely, a syntax error or another error was detected at the beginning of the execution phase.

Validating and Cleaning Data

Data errors occur when data values are not appropriate for the SAS statements that are specified in a program. SAS detects data errors during program execution. When a data error is detected, SAS continues to execute the program.

In general, SAS procedures analyze data, produce output, or manage SAS files. In addition, SAS procedures can be used to detect invalid data. The following procedures can be used to detect invalid data:

- PROC PRINT
- PROC FREQ
- PROC MEANS

The PRINT procedure displays the data set, showing that the values for IDnum are missing. Periods indicate that IDnum is a numeric variable, although it should have been defined as a character variable.

```
proc print data=perm.update;
run;
```

Obs	Item	IDnum	InStock	BackOrd	Total
1	Birci Feeder	.	3	20	23
2	6 Glass Mugs	.	6	12	18
3	Glass Tray	.	12	6	18
4	Padded Hangrs	.	15	20	35
5	Jewelry Box	.	23	0	23
6	Red Apron	.	9	12	21
7	Crystal Vase	.	27	0	27
8	Picnic Basket	.	21	0	21
9	Brass Clock	.	2	10	12

Figure 6.1: Output from the PRINT Procedure Showing Missing Values for IDnum

The FREQ procedure detects invalid character and numeric values by looking at distinct values. You can use PROC FREQ to identify any variables that were not given an expected value.

General form, FREQ procedure:

PROC FREQ DATA=*SAS-data-set* <*NLEVELS*>;

TABLES *variable(s)*;

RUN;

where

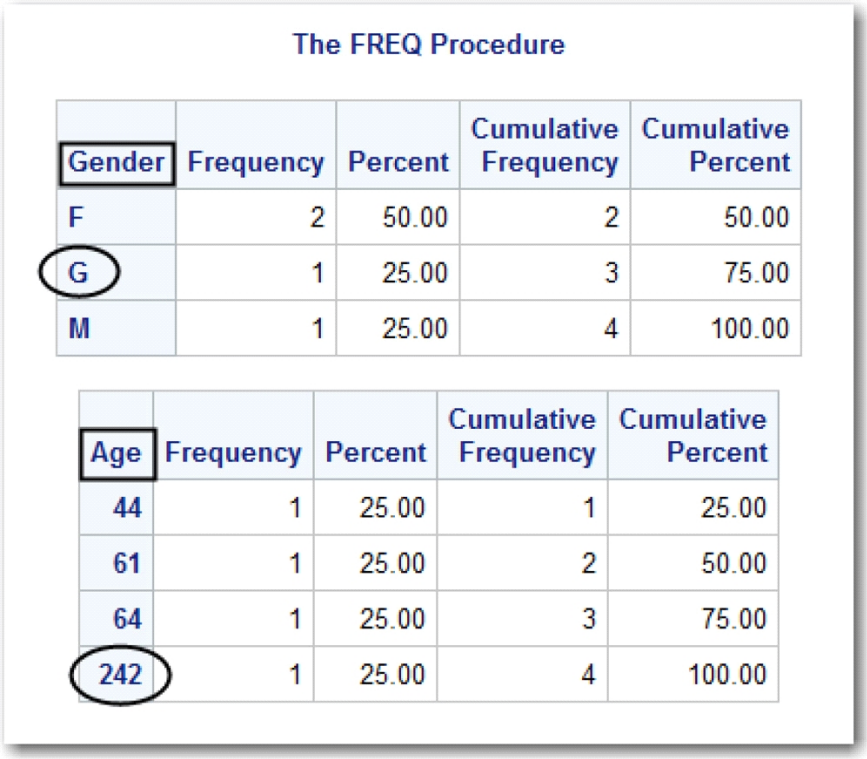
- The TABLES statement specifies the frequency tables to produce.
- The NLEVELS option displays a table that provides the number of distinct values for each variable named in the TABLES statement.

In the following example, the data set contains invalid characters for the variables Gender and Age. PROC FREQ displays the distinct values of variables and is therefore useful for finding invalid values in data. You can use PROC FREQ with the TABLES statement to produce a frequency table for specific variables.

```
proc freq data=work.Patients;
      tables Gender Age;
```

```
run;
```

In the output you can see both the valid **(M** and **F)** and invalid **(G)** values for Gender, and the valid and invalid (242) values for age. In both the gender and age FREQ tables, one observation needs data cleaned as shown below:



The FREQ Procedure

Gender	Frequency	Percent	Cumulative Frequency	Cumulative Percent
F	2	50.00	2	50.00
G	1	25.00	3	75.00
M	1	25.00	4	100.00

Age	Frequency	Percent	Cumulative Frequency	Cumulative Percent
44	1	25.00	1	25.00
61	1	25.00	2	50.00
64	1	25.00	3	75.00
242	1	25.00	4	100.00

The MEANS procedure can also be used to validate data because it produces summary reports displaying descriptive statistics. For example, PROC MEANS can show whether the values for a particular variable are within their expected range.

General form, MEANS procedure:

PROC MEANS DATA=*SAS-data-set* <*statistics*>;

VAR *variable(s);*

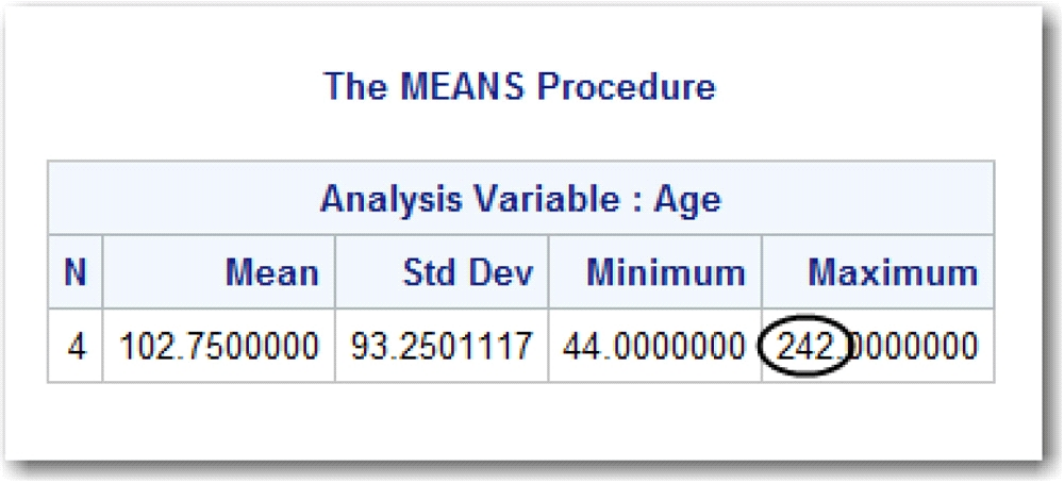
RUN;

- The VAR statement specifies the analysis variables and their order in the results.
- The statistics to display can be specified in the PROC MEANS statement.

Using the same data set as in the previous example, we can submit PROC MEANS to determine if the Age of all test subjects is within a reasonable range. Notice the VAR statement is specified with that particular variable to get the statistical information, or range, of the data values.

```
proc means data=work.Patients;
      var Age;
run;
```

The output for the MEANS procedure displays a range of 44 to 242, which clearly indicates that there is invalid data somewhere in the Age column.



The MEANS Procedure

Analysis Variable : Age				
N	Mean	Std Dev	Minimum	Maximum
4	102.7500000	93.2501117	44.0000000	242.0000000

Cleaning the Data

You can use an assignment statement or a conditional clause to programmatically clean invalid data once it is identified. For example, if your input data contains a field called Gender, and that field has an invalid value (a value other than M or F), then you can clean your data by changing the invalid value to a valid value for Gender. To avoid overwriting your original data set, you can use the DATA statement to create a new data set. The new data set will contain all of the data from your original data set, along with the correct values for invalid data.

The following example assumes that in your input data, Gender has an invalid value of G. This error might be the result of a data entry error. After examining your data, you determine that G should actually be F. You can fix the invalid data for Gender by using an assignment statement along with an IF-THEN statement:

```
data work.clean_data;
   set work.patients;
   gender=upcase(Gender);
   if Gender='G' then Gender='F';
run;

proc print data=work.clean_data;
run;
```

When you examine your input data further, you find that two observations contain invalid values for Age. These values exceed a maximum value of 110. You can uniquely identify each of the observations by the variable Empid. By checking the date of birth in each of the observations, you determine the correct value for Age. To change the data, you can use an IF-THEN-ELSE statement:

```
data work.clean_data;
   set work.patients;
   if empid=3294 then age=65;
   else if empid=7391 then age=75;
run;
proc print data=work.clean_data;
run;
```

Another way of ensuring that your output data set contains valid data is to programmatically identify invalid data and delete the associated observations from your output data set:

```
data work.clean_data;
   set work.patients;
   if Age>110 then delete;
run;

proc print data=work.clean_data;
run;
```

Testing Your Programs

Limiting Observations

Remember that you can use the OBS= option in the INFILE statement to limit the number of observations that are read or created during the execution of the DATA step.

```
data perm.update;
   infile invent obs=10;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

When processed, this DATA step creates the perm.update data set with variables but with only ten observations.

PUT Statement

When the source of program errors is not apparent, you can use the PUT statement to examine variable values and to print your own message in the log. For diagnostic purposes, you can use IF-THEN/ELSE statements to conditionally check for values. You can learn about IF-THEN/ELSE statements in detail in "Creating and Managing Variables" on page 304.

```
data work.test;
   infile loan;
   input Code $ 1 Amount 3-10 Rate 12-16
         Account $ 18-25 Months 27-28;
   if code='1' then type='variable';
   else if code='2' then type='fixed';
   else type='unknown';
   if type=unknown then put 'MY NOTE: invalid value: '
   code=;
run;
```

In this example, if CODE does not have the expected values of **1** or **2**, the PUT statement writes a message to the log:

```
MY NOTE: invalid value: Code=V
NOTE: 9 records were read from the infile LOAN.
      The minimum record length was 28.
      The maximum record length was 28.
NOTE: The data set WORK.TEST has 9 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.03 seconds
```

Figure 6.2: SAS Log

General form, simple PUT statement:

PUT *specification(s);*

where *specification* specifies what is written, how it is written, and where it is written. This can include

- a character string
- one or more data set variables
- the automatic variables _N_ and _ERROR_
- the automatic variable _ALL_

and much more. The following pages show examples of PUT specifications.

Character Strings

You can use a PUT statement to specify a character string to identify your message in the log. The character string must be enclosed in quotation marks.

```
put 'MY NOTE: The condition was met.';
```

```
MY NOTE: The condition was met.
NOTE: 9 records were read from the infile LOAN.
      The minimum record length was 28.
      The maximum record length was 28.
NOTE: The data set WORK.TEST has 9 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Figure 6.3: SAS Log

Data Set Variables

You can use a PUT statement to specify one or more data set variables to be examined for that iteration of the DATA step:

```
put 'MY NOTE: invalid value: '
     code type;
```

```
MY NOTE: invalid value: V unknown
NOTE: 9 records were read from the infile LOAN.
      The minimum record length was 28.
      The maximum record length was 28.
NOTE: The data set WORK.TEST has 9 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Figure 6.4: SAS Log

Note that when you specify a variable in the PUT statement, only its value is written to the log. To write both the variable name and its value in the log, add an equal sign (=) to the variable name.

```
put 'MY NOTE: invalid value: '
     code= type=;
```

```
MY NOTE: invalid value: Code=V type=unknown
NOTE: 9 records were read from the infile LOAN.
      The minimum record length was 28.
      The maximum record length was 28.
NOTE: The data set WORK.TEST has 9 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

Figure 6.5: SAS Log

Automatic Variables

You can use a PUT statement to display the values of the automatic variables _N_ and _ERROR_. In some cases, knowing the value of _N_ can help you locate an observation in the data set:

```
put 'MY NOTE: invalid value: '
     code= _n_=_error_=;
```

```
MY NOTE: invalid value: Code=V _N_=3 _ERROR_=0
NOTE: 9 records were read from the infile LOAN.
      The minimum record length was 28.
      The maximum record length was 28.
NOTE: The data set WORK.TEST has 9 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Figure 6.6: SAS Log

You can also use a PUT statement to write all variable names and variable values, including automatic variables, to the log. Use the _ALL_ specification:

```
put 'MY NOTE: invalid value:' _all_ ;
```

```
MY NOTE: invalid value: Code=V Amount=10000 Rate=10.5 Account=101-1289 Months=8
type=unknown  _ERROR_=0 _N_=3
NOTE: 9 records were read from the infile LOAN.
      The minimum record length was 28.
      The maximum record length was 28.
NOTE: The data set WORK.TEST has 9 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Figure 6.7: SAS Log

Conditional Processing

You can use a PUT statement with conditional processing (that is, with IF-THEN/ELSE statements) to flag program errors or data that is out of range. In the example below, the PUT statement is used to flag any missing or zero values for the variable Rate.

```
data finance.newcalc;
   infile newloans;
   input LoanID $ 1-4 Rate 5-8 Amount 9-19;
   if rate>0 then
      Interest=amount*(rate/12);
   else put 'DATA ERROR' rate= _n_=;
run;
```

Additional Note The PUT statement can accomplish a wide variety of tasks. This chapter shows a few ways to use the PUT statement to help you debug a program or examine variable values. For a complete description of the PUT statement, see the SAS documentation.

Chapter Summary

Text Summary

How SAS Processes Programs

A SAS DATA step is processed in two distinct phases. During the compilation phase, each statement is scanned for syntax errors. During the execution phase, the DATA step writes observations to the new data set.

Compilation Phase

At the beginning of the compilation phase, the input buffer and the program data vector are created. The program data vector is the area of memory where one observation at a time is processed. Two automatic variables are also created: _N_ counts the number of DATA step executions, and _ERROR_ signals the occurrence of a data error. DATA step statements are checked for syntax errors, such as invalid options or misspellings.

Execution Phase

During the execution phase, one record at a time from the input file is read into the input buffer, copied to the program data

vector, and then written to the new data set as an observation. The DATA step executes once for each record in the input file, unless otherwise directed.

Diagnosing Errors in the Compilation Phase

Missing semicolons, misspelled keywords, and invalid options will cause syntax errors in the compilation phase. Detected errors are underlined and are identified with a number and message in the log. If SAS can interpret a syntax error, the DATA step compiles and executes; if SAS cannot interpret the error, the DATA step compiles but doesn't execute.

Diagnosing Errors in the Execution Phase

Illegal mathematical operations or processing a character variable as numeric will cause data errors in the execution phase. Depending on the type of error, the log might show a warning and might include invalid data from the input buffer or the program data vector, and the DATA step either stops or continues.

Testing Your Programs

To detect common errors and save development time, use the OBS= option in the INFILE statement to limit the number of observations that are read or created during the DATA step. You can also use the PUT statement to examine variable values and to generate your own message in the log. For information about Informats, see "Using Informats" on page 521.

Syntax

```
DATA <_NULL_|SAS-data-set>;
       INFILE file-specification OBS=n;
       INPUT variable-1 informat-1
       < …variable-n informat-n;>
       PUT specification(s);
RUN;
```

Sample Programs

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;

data work.test;
   infile loan;
   input Code $ 1 Amount 3-10;
   if code='1' then type='variable';
   else if code='2' then type='fixed';
   else put 'MY NOTE: invalid value: '
        code=;
run;
```

Points to Remember

- Making, diagnosing, and resolving errors is part of the process of writing programs. However, checking for common errors will save you time and trouble. Make sure that
  - each SAS statement ends with a semicolon
  - filenames are spelled correctly
  - keywords are spelled correctly.
- In SAS output, missing numeric values are represented by periods, and missing character values are left blank.
- The order in which variables are defined in the DATA step determines the order in which the variables are stored in the data set.
- Standard character values can include numbers, but numeric values cannot include characters.

Chapter Quiz

Select the best answer for each question. After completing the quiz, you can check your answers using the answer key in the appendix.

1. Which of the following is *not* created during the compilation phase? ?

a. the data set descriptor

b. the first observation

c. the program data vector

d. the _N_ and _ERROR_ automatic variables

2. During the compilation phase, SAS scans each statement in the DATA step, looking for syntax errors. Which of the following is *not* considered a syntax error? ?

a. incorrect values and formats

b. invalid options or variable names

c. missing or invalid punctuation

d. missing or misspelled keywords

3. Unless otherwise directed, the DATA step executes… ?

a. once for each compilation phase.

b. once for each DATA step statement.

c. once for each record in the input file.

d. once for each variable in the input file.

4. At the beginning of the execution phase, the value of _N_ is **1**, the value of _ERROR_ is **0**, and the values of the remaining variables are set to: ?

a. **0**

b. **1**

c. undefined

d. missing

5. Suppose you run a program that causes three DATA step errors. What is the value of the automatic variable _ERROR_ when the observation that contains the third error is processed? ?

a. **0**

b. **1**

c. **2**

d. **3**

6. Which of the following actions occurs at the beginning of an iteration of the DATA step? ?

a. The automatic variables _N_ and _ERROR_ are incremented by one.

b. The DATA step stops execution.

c. The descriptor portion of the data set is written.

d. The values of variables created in programming statements are re-set to missing in the program data vector.

7. Look carefully at the DATA step shown below. Based on the INPUT statement, in what order will the variables be stored in the new data set? ?

```
data perm.update;
     infile invent;
     input IDnum $ Item $ 1-13 Instock 21-22
           BackOrd 24-25;
     Total=instock+backord;
run;
```

a. IDnum Item InStock BackOrd Total

b. Item IDnum InStock BackOrd Total

c. Total IDnum Item InStock BackOrd

d. Total Item IDnum InStock BackOrd

8. If SAS cannot interpret syntax errors, then… ?

a. data set variables will contain missing values.

b. the DATA step does not compile.

c. the DATA step still compiles, but it does not execute.

d. the DATA step still compiles and executes.

9. What is wrong with this program? ?

```
data perm.update;
   infile invent
   input Item $ 1-13 IDnum $ 15-19 Instock 21-22
         BackOrd 24-25;
   total=instock+backord;
run;
```

a. missing semicolon on second line

b. missing semicolon on third line

c. incorrect order of variables

d. incorrect variable type

10. Look carefully at this section of a SAS session log. Based on the note, what was the most likely problem with the DATA step? ?

```
NOTE: Invalid data for IDnum in line 7 15-19.
RULE: ----+----1----+----2----+----3----+----4
7       Bird Feeder LG088 3 20
Item=Bird Feeder IDnum=. InStock=3 BackOrd=20
Total=23 _ERROR_=1 _N_=1
```

a. A keyword was misspelled in the DATA step.

b. A semicolon was missing from the INFILE statement.

c. A variable was misspelled in the INPUT statement.

d. A dollar sign was missing in the INPUT statement.

Answers

1. Correct answer: b

During the compilation phase, the program data vector is created. The program data vector includes the two automatic variables _N_ and _ERROR_. The descriptor portion of the new SAS data set is created at the end of the compilation phase. The descriptor portion includes the name of the data set, the number of observations and variables, and the names and attributes of the variables. Observations are not written until the execution phase.

2. Correct answer: a

Syntax checking can detect many common errors, but it cannot verify the values of variables or the correctness of formats.

3. Correct answer: c

The DATA step executes once for each record in the input file, unless otherwise directed.

4. Correct answer: d

The remaining variables are initialized to missing. Missing numeric values are represented by periods, and missing character values are represented by blanks.

5. Correct answer: b

The default value of _ERROR_ is 0, which means there is no data error. When an error occurs, whether one error or multiple errors, the value is set to 1.

6. Correct answer: d

By default, at the end of the DATA step, the values in the program data vector are written to the data set as an observation, control returns to the top of the DATA step, the value of the automatic variable _N_ is incremented by one, and the values of variables created in programming statements are reset to missing. The automatic variable _ERROR_ is reset to 0 if necessary.

7. Correct answer: a

The order in which variables are defined in the DATA step determines the order in which the variables are stored in the data set.

8. Correct answer: c

When SAS cannot detect syntax errors, the DATA step compiles, but it does not execute.

9. Correct answer: a

A semicolon is missing from the second line. It will cause an error because the INPUT statement will be interpreted as invalid INFILE statement options.

10. Correct answer: d

The third line of the log displays the value for IDnum, which is clearly a character value. The fourth line displays the values in the program data vector and shows a period, the symbol for a missing numeric value, for IDnum, the other values are correctly assigned. Thus, it appears that numeric values were expected for IDnum. A dollar sign, to indicate character values, is missing from the INPUT statement.