Chapter 6

Understanding DATA Step Processing

Overview

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. This can help you to

- anticipate how variables will be created and processed
- plan your modifications
- interpret and debug program errors
- give you useful strategies for preventing and correcting common DATA step errors

Topics

- 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.

Basic DATA Steps

```
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
```

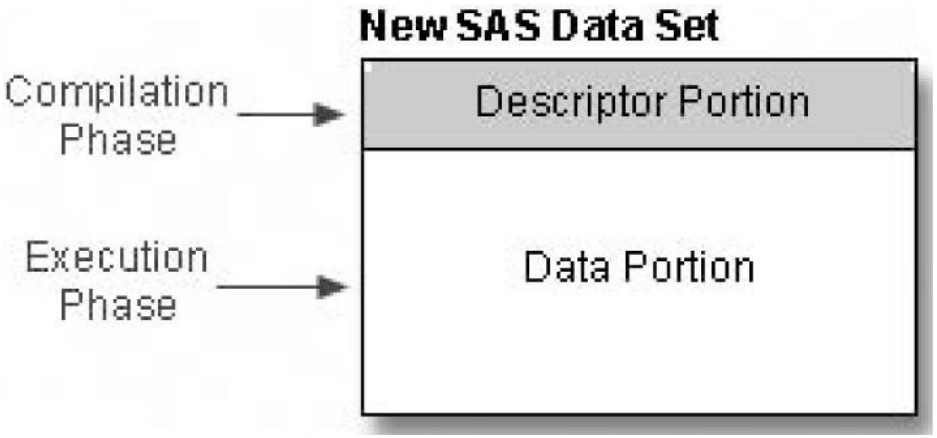
Obs	ID	Name	RestHr	MaxHR	RecHR	TimeMin	TimeSec	Tolerance
1	2458	Murray, W	72	185	128	12	38	D
2	2462	Almers, C	68	171	133	10	5	I
3	2501	Bonaventure, T	78	177	139	11	13	I
4	2523	Johnson, R	69	162	114	9	42	S
5	2539	LaMance, K	75	168	141	11	46	D
6	2544	Jones, M	79	187	136	12	26	N
7	2552	Reberson, P	69	158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

```
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
```

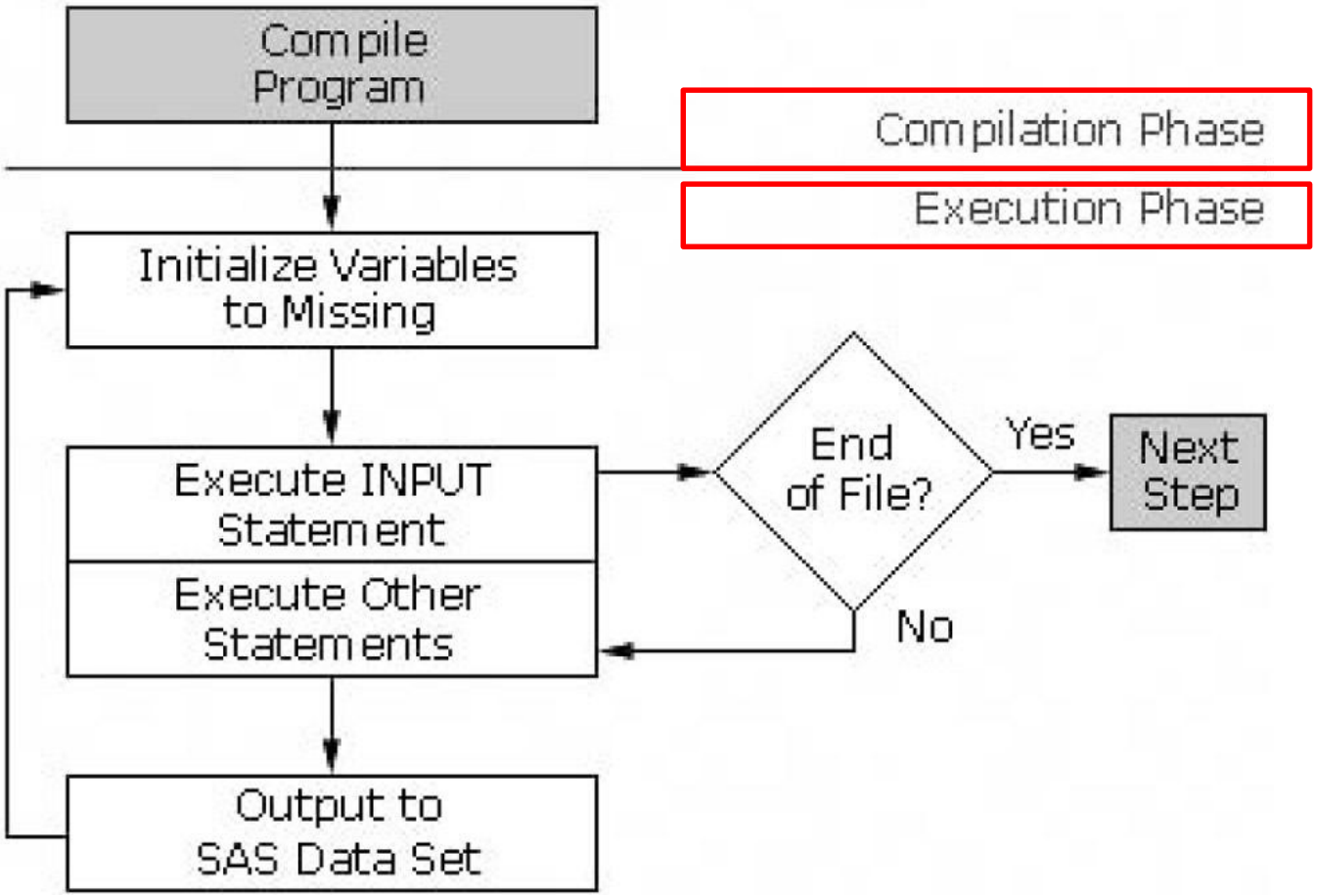
How SAS Processes Program

A SAS DATA step is processed in two phases:



- During the compilation phase, statements are 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.
- Once compilation is successful, the execution phase begins. DATA step executes **once** for each row in the input file (there are also exceptions).

General flow of DATA step processing for reading raw data



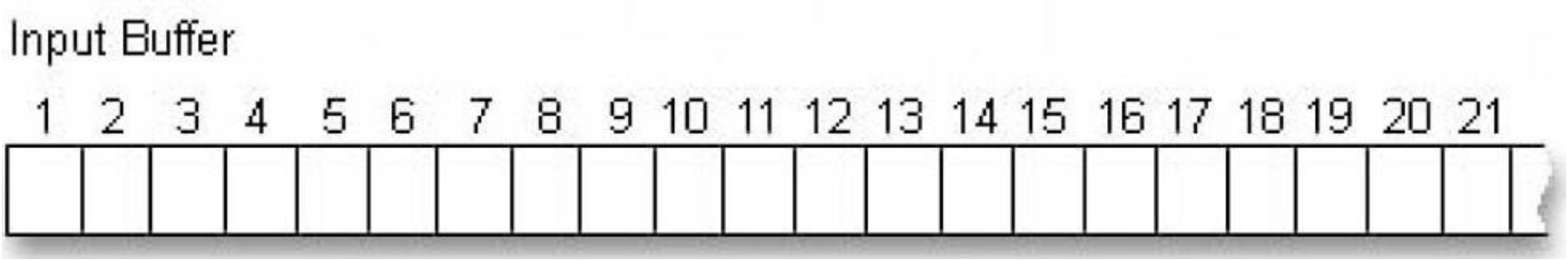
Compilation Phase

- Create **Input buffer** (if needed)
- Create **Program Data Vector (PDV),** which is created after the input buffer (if any) is created
- Check **Syntax** for errors
- Create **Data Set Variables**
- At last create the **descriptor portion** of the SAS data set

Input Buffer(An Area of Memory)

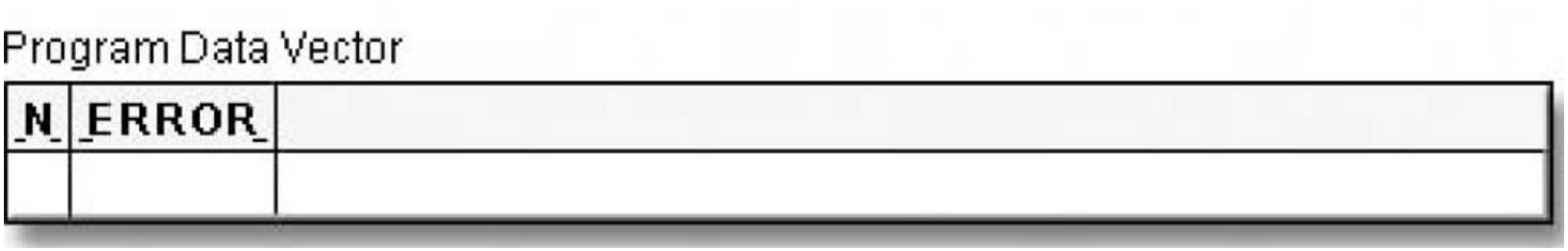
The input buffer is created

- only when raw data are read from an external file (not when a SAS data set is read)
- to hold a record from the external file
- not in a true storage area



Program Data Vector (PDV)

- After the input buffer is created, the program data vector (PDV) is created. The PDV is the area of memory where SAS holds one observation at a time.
- The PDV contains two automatic variables:
  - _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 (default = 0, meaning no error).



Syntax Checking

SAS scans each statement in the DATA step, looking for syntax errors, including

- missing or misspelled keywords
- invalid variable names
- missing or invalid punctuations
- 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.
- Any variables that are created with an assignment statement in the DATA step are also added to 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;
```

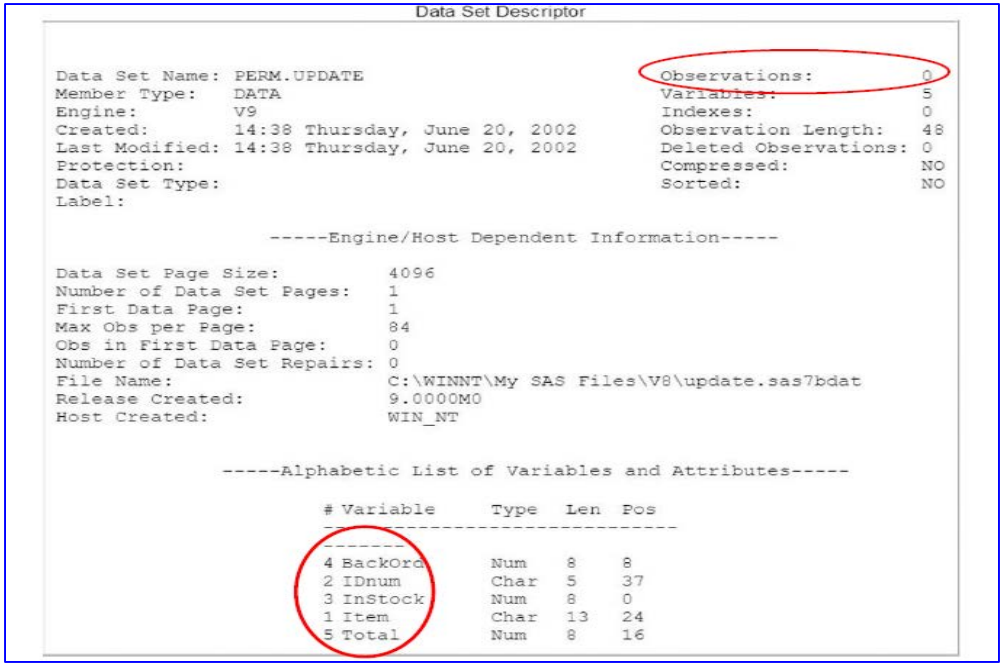
Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total

Descriptor Portion of the SAS Data Set

- At the bottom of the DATA step (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
  - number of variables
  - names and attributes of the variables.

- The data set in this example contains 5 variables.
- _N_ and _ERROR_ are not written to the data set.
- There are no observations because the DATA step has not yet executed.



Data Set Descriptor

Data Set Name:	PERM.UPDATE	Observations:	0
Member Type:	DATA	Variables:	5
Engine:	V9	Indexes:	0
Created:	14:38 Thursday, June 20, 2002	Observation Length:	48
Last Modified:	14:38 Thursday, June 20, 2002	Deleted Observations:	0
Protection:		Compressed:	NO
Data Set Type:		Sorted:	NO
Label:			

-----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:	0
Number of Data Set Repairs:	0
File Name:	C:\WINNT\My SAS Files\V8\update.sas7bdat
Release Created:	9.0000M0
Host Created:	WIN_NT

-----Alphabetic List of Variables and Attributes-----

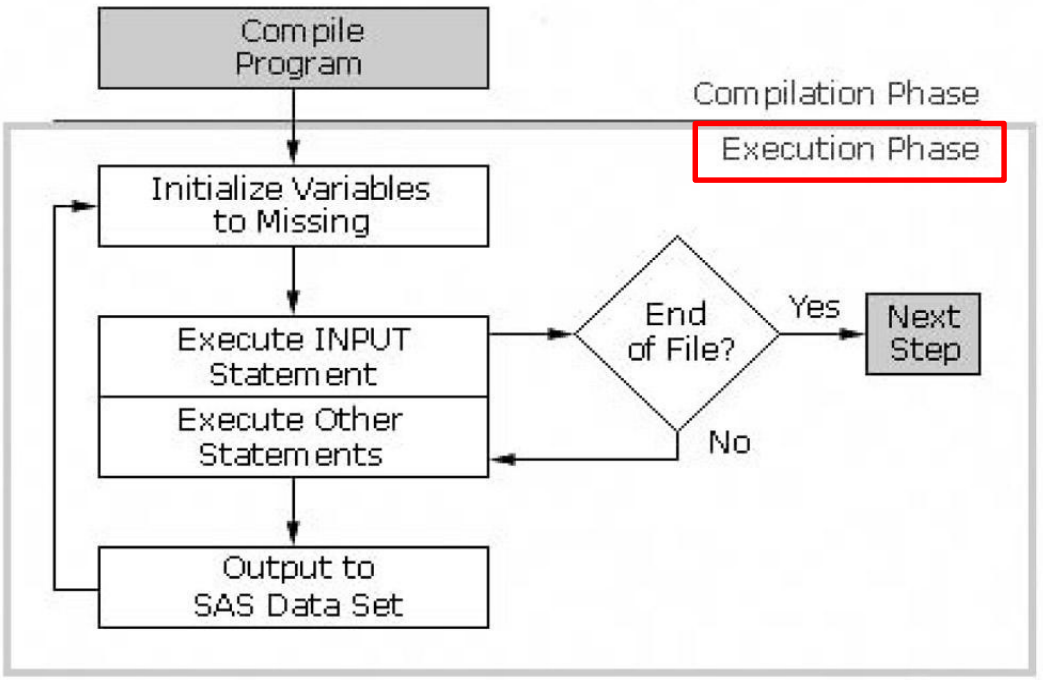
#	Variable	Type	Len	Pos
4	BackOrd	Num	8	8
2	IDnum	Char	5	37
3	InStock	Num	8	0
1	Item	Char	13	24
5	Total	Num	8	16

Execution Phase

During the execution phase, the data portion of the data set is created. It contains the following steps:

- Initializing Variables
- Identifying an external file with the INFILE Statement
- Reading in data with the INPUT Statement
- End of the DATA Step
- Iterations of the DATA Step
- End-of-File Marker
- End of the Execution Phase

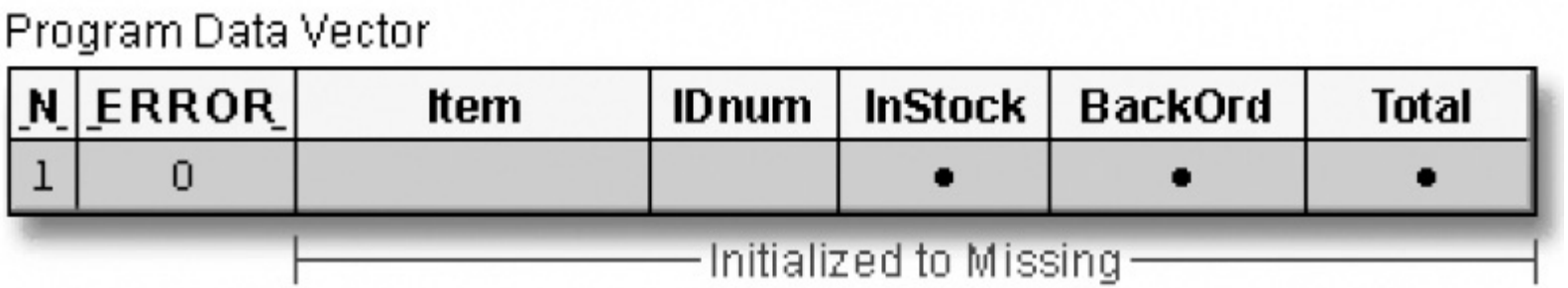
Recall:



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

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



Identifying the Location of an External File With the INFILE Statement

```
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			.	.	.

Reading a Record Into the PDV With the INPUT Statement

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		•	•	•

Reading a Record Into the PDV With the INPUT Statement

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	•	•	•

Reading a Record Into the PDV With the INPUT Statement

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	•

Reading a Record Into the PDV With the INPUT Statement

After that, 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

Actions Occurred at the End of a DATA Step

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

Actions Occurred at the End of a DATA Step

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

The DATA step works like a loop, repetitively executing statements to read and create observations one by one. Each loop 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;
   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			.	.	.

SAS Data Set Perm.Update

Item	IDnum	InStock	BackOrd	Total
Bird Feeder	LG088	3	20	23

Iterations of the DATA Step

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.

```
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	.

SAS Data Set Perm.Update

Item	IDnum	InStock	BackOrd	Total
Bird Feeder	LG088	3	20	23

Iterations of the DATA Step

Next, the value for Total is calculated based on the current values for InStock and BackOrd.

```
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

Iterations of the DATA Step

The RUN statement indicates the end of the DATA step loop. 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;
```

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

Iterations of the DATA Step: End-of-File Marker

The execution phase continues in this manner until the end-of-file marker is reached in the raw data file. At that time the data portion of the new data set is complete and the DATA step stops.

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

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.
```

Debugging a DATA Step

Diagnosing Errors in the Compilation Phase

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.

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

Debugging a DATA Step

Diagnosing Errors in 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.

When SAS does not execute a compiled step where the error occurred, it prints the following message:

NOTE: The SAS System stopped processing this step because of errors.

Debugging a DATA Step

Example 1: Diagnosing Errors in the Execution Phase

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

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. Because there is no external file that is referenced by the fileref Invnt, the DATA step stops processing.

Debugging a DATA Step

Example 1: Diagnosing Errors in the Execution Phase

```
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
```

Note that in the log the correct number of variables was defined in the descriptor portion of the data set.

Debugging a DATA Step

Example 2: Diagnosing Errors in the Execution Phase

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

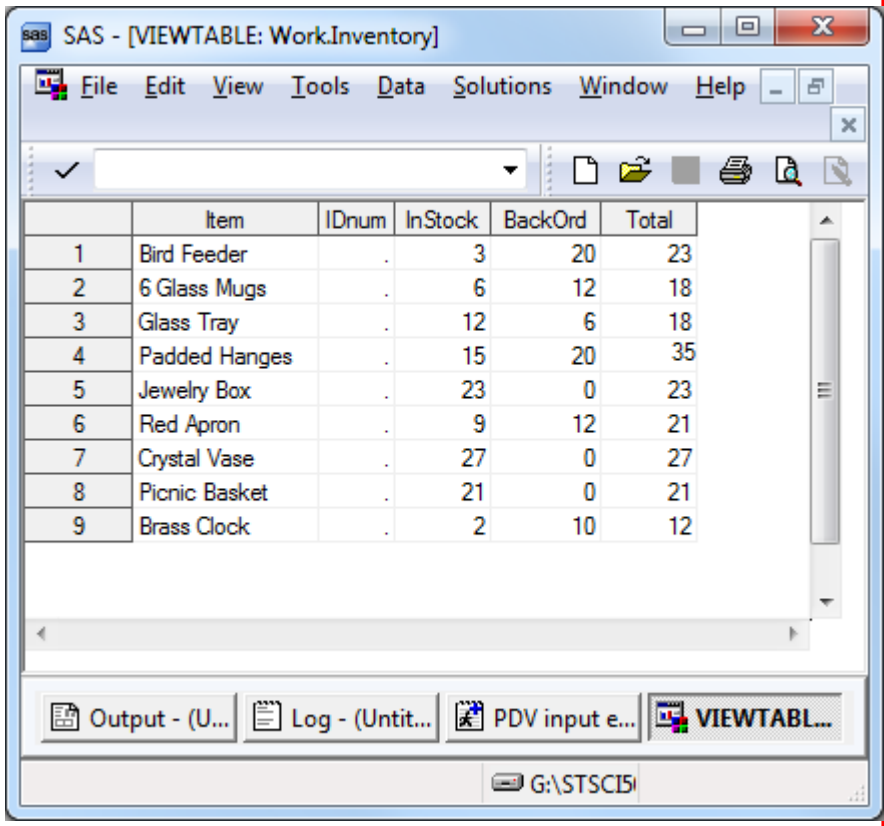
Incorrectly identifying a variable's type is another common execution-time error. This is not a compile-time error, because SAS cannot verify a variable's type until the data values for the variable are read.

Debugging a DATA Step

Example 2: Diagnosing Errors in the Execution Phase

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, and the values for IDnum are missing in the data set.

```
NOTE: Invalid data for IDnum in line 1 15-19.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+-
1          Bird Feeder   LG088  3 20
      89                                                        136
Item=Bird Feeder IDnum=. InStock=3 BackOrd=20 Total=23 _ERROR_=1 _N_=1
NOTE: Invalid data for IDnum in line 2 15-19.
2          6 Glass Mugs  SB082  6 12
      89                                                        136
Item=6 Glass Mugs IDnum=. InStock=6 BackOrd=12 Total=18 _ERROR_=1 _N_=2
NOTE: Invalid data for IDnum in line 3 15-19.
3          Glass Tray    BQ049 12  6
      89                                                        136
Item=Glass Tray IDnum=. InStock=12 BackOrd=6 Total=18 _ERROR_=1 _N_=3
NOTE: Invalid data for IDnum in line 4 15-19.
4          Padded Hanges MN256 15 20
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+-
      89                                                        136
Item=Padded Hanges IDnum=. InStock=15 BackOrd=20 Total=35 _ERROR_=1 _N_=4
NOTE: Invalid data for IDnum in line 5 15-19.
5          Jewelry Box   AJ498 23  0
      89                                                        136
Item=Jewelry Box IDnum=. InStock=23 BackOrd=0 Total=23 _ERROR_=1 _N_=5
NOTE: Invalid data for IDnum in line 6 15-19.
6          Red Apron     AQ072  9 12
      89                                                        136
Item=Red Apron IDnum=. InStock=9 BackOrd=12 Total=21 _ERROR_=1 _N_=6
NOTE: Invalid data for IDnum in line 7 15-19.
7          Crystal Vase  AQ672 27  0
      89                                                        136
Item=Crystal Vase IDnum=. InStock=27 BackOrd=0 Total=27 _ERROR_=1 _N_=7
NOTE: Invalid data for IDnum in line 8 15-19.
8          Picnic Basket LS930 21  0
      89                                                        136
Item=Picnic Basket IDnum=. InStock=21 BackOrd=0 Total=21 _ERROR_=1 _N_=8
NOTE: Invalid data for IDnum in line 9 15-19.
9          Brass Clock   AN910  2 10  26
Item=Brass Clock IDnum=. InStock=2 BackOrd=10 Total=12 _ERROR_=1 _N_=9
NOTE: 9 records were read from the infile INVENT.
      The minimum record length was 26.
      The maximum record length was 136.
NOTE: The data set WORK.INVENTORY has 9 observations and 5 variables.
```



	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 Hanges	.	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

Validating and Cleaning Data

- Data errors occur when data values are not appropriate for the SAS statements. SAS detects data errors during program execution. When a data error is detected, SAS continues to execute the program.
- Some SAS procedures can be used to detect invalid data:
    - PROC PRINT
    - PROC FREQ
    - PROC MEANS

Validating and Cleaning Data

Validating data with PROC FREQ

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

Invalid values for Gender (G)

Invalid values for age (242)

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

Validating and Cleaning Data

Validating data with PROC MEANS

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

The MEANS Procedure

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

Invalid values for age

Validating and Cleaning Data

Cleaning the Data

You can fix the invalid data (e.g., Gender and Age) by using an assignment statement along with an IF-THEN statement.

```
data work.clean_data;
   set work.patients;
   if Gender='G' then Gender='F';
   if Age>110 then delete;
run;

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

Testing Your Programs

Limiting Observations

```
data perm.update;
  infile invent obs=2;
  input Item $ 1-13 IDnum $ 15-19 InStock 21-22 BackOrd 24-25;
  Total=instock+backord;
run;
```

Only 2 observations are read and written. This will be convenient for testing your program.

Testing Your Programs

Creating a NULL Data Set

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

No data set will be created but errors (if any) will be written to the log.

Testing Your Programs

Using the PUT Statement

- The PUT statement can be used to examine variable values and to print your own message in the log. IF-THEN/ELSE statements are often used to conditionally check for values.
- A PUT statement can specify what, how, and where to write your message. Here are examples that you can write:
  - a character string
    (put "MY NOTE: The condition was met. ";)
  - one or more data set variables
    (put 'MY NOTE: invalid value: ' code type;
    or
    put 'MY NOTE: invalid value: ' code= type=;)
  - the automatic variables _N_ and _ERROR_
    (put 'MY NOTE: invalid value: ' code= _n_= _error_=;)
  - the automatic variable _ALL_
    (put 'MY NOTE: invalid value:' _all_;)

Testing Your Programs

Example 1: Using the PUT Statement

```
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;
```

```
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
```

Testing Your Programs

Example 2: Using the PUT Statement

```
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;
```