

# The Secret Life of the DATA Step

Thomas Miron  
Miron InfoTec, Inc., Madison, WI

## The Secret Life of the DATA Step

Each SAS DATA step functions as a self-contained mini-program that is compiled and run within your overall SAS program. Much of DATA step processing is implicit. For example, DATA step statements are executed within an implied loop even though you do not code loop control statements.

This tutorial discusses several of the implied DATA step statements that control how your DATA step really works.

## DATA Step Magic

- Files are opened and closed.
- The DATA step starts and stops at (usually) just the right place.
- Certain statements seem to perform amazing feats of data processing in a single stroke.

## The DATA Step Is Not Magic!

- Overall DATA step processing is governed by a strict set of rules.
- Each DATA step statement causes a well-defined and limited set of actions in your program.

## The "Secret" DATA Step

- Understanding the underlying processes, the "secret life" of the DATA step, is critical if you are:
  - creating sophisticated DATA step applications
  - want to build efficient programs
  - need to maintain programs

## What's This Tutorial About?

- We will look at some of the *automagical* DATA step statements and processes.
- We will probe the DATA step using programming experiments to demonstrate underlying DATA step processes.
- Functional description of the DATA step

This tutorial is intended as a functional description of the DATA step, not as a technical discussion of actual DATA step memory structures and processes.

## Some DATA Step Basics

- Execution phases
- Resources
- Program Data Vector
- Instruction Pointer

## SAS DATA Steps Phases

- Compile Phase
- Run or Execution Phase

## Compile Phase

- Syntax Check
- Resources Established
- Implied Executable Statements Generated

## DATA Step Resources

- SAS Data Sets and External Files
- In-Memory Data Structures
- Variables and Variable Attributes

## Variable Attributes

- Name
- Type
- Length
- Format, Informat, Label
- Processing Flags:
  - retain flag
  - missing protect
  - keep flag
- Current Value of the Variable

## Program Data Vector (PDV)

- The Program Data Vector or PDV can be thought of as a table of variable attributes.

Program Data Vector (PDV)									
Name	Label	Type	Length	Retain Flag	Missing Protect Flag	Keep Flag	Format	Informat	Value

This is an expanded vision of the PDV. It is commonly represented as holding only the current value of variables.

---

### Run Phase

- All executable statements are processed in sequence.
- Non-executable or declarative statements are not visible during the run phase.

For example, DATA, RUN, and KEEP statements are not executed at run time. These statements are processed at compile time, before the DATA step is run.

---

### Declarative Statements:

- KEEP
- LENGTH
- ATTRIB
- others

---

### Two Types of Executable Statement

- Explicit
- Implicit

---

### Explicit Statements


- Statements You Explicitly Code In Your Program:

```
SET
INPUT
IF
assignment ( x = 2; )
```

---

### Instruction Pointer

- The instruction pointer points to the next statement to execute.

```
data a;
    set b;
     x = y + 1;
    z = y/2;
run;
```

Running a DATA step really means executing each statement in turn. One way to visualize this is to think of an instruction pointer that points to the next DATA step statement to be executed. Normally, the instruction pointer jumps from one statement to the next in the sequence that the statements were coded. This is why the order of your DATA step executable statements is critical

---

### Implied Statements

- Account for much DATA step magic.
- Control the DATA step loop.
- Open and close files.
- Stop the DATA step when an end-of-file condition is encountered.
- Many Other Processes

The statements that execute when your DATA step is run are made up of explicit statements plus implied statements added during the compile phase.

---

### Implied OUTPUT and GOTO

- The top of the DATA step is assigned an implied label.
- By default, at the bottom of each DATA step are implied OUTPUT and GOTO statements.

```
data a;
TOP-OF-DATA-STEP:
    set b;
output
goto top-of-DATA-step
run;
```

---

### The DATA Step Loop

- The instruction pointer "lands" on each executable statement.
- Each executable statement causes some specific action.
- The implied GOTO statement creates a loop of executable statements.

The implied DATA step loop is fundamental to understanding how the DATA step works. During the compile phase an implied TOP-OF-DATA-STEP label is added to your program and an implied GOTO TOP-OF-DATA-STEP statement is added at the end of your program. The instruction pointer executes each statement in your program then jumps back to the TOP-OF-DATA-STEP label to begin another loop.

---

### Ending The Loop

- The DATA step loop ends with an attempt to read passed the end of file.
- Other conditions and statements can stop the DATA step as well.

The instruction pointer, implied TOP-OF-DATA-STEP label, and GOTO statement work to process the basic DATA step loop, but what ends the loop? Another implied statement is added during the compile phase: IF END-OF-FILE THEN STOP.

Data Set B	
OBS	X
1	1
2	2
3	3
4	4

```

data a;
TOP-OF-DATA-STEP:
  put 'loop';
  set b;
if EOF then stop
output
goto top-of-DATA-step
run;

```

```

loop
loop
loop
loop
loop

```

NOTE: The data set WORK.A has 4 observations and 1 variables.

### **\_N\_ Assignment**

- **\_N\_** is an automatic variable
- **\_N\_** is set to the number of times the DATA loop has started

```

data a;
TOP-OF-DATA-STEP:
  _N_ = start_count
  put _n_ =;
  set b;
if EOF then stop
output
goto top-of-DATA-step
run;

```

```

_N_ =1
_N_ =2
_N_ =3
_N_ =4
_N_ =5

```

An implied statement that sets the value of **\_N\_** is also added to your DATA step program. There is often confusion as to what **\_N\_** is. It is not the count of records read. It is not the number of loops through the DATA step. **\_N\_** indicates the number of times the DATA step loop as started. The first statement at the top of each DATA step is an implied statement: **\_N\_ = number of times the DATA loop has started.**

### **\_N\_ Is Assigned a Value, Not Incremented**

```

data a;
TOP-OF-DATA-STEP:
  _N_ = start_count
  put _n_ =;
  set b;
  _n_ = 0;
  put _n_ =;
if EOF then stop
output
goto top-of-DATA-step
run;

```

```

_N_ =1
_N_ =0
_N_ =2
_N_ =0
_N_ =3
_N_ =0
_N_ =4
_N_ =0
_N_ =5

```

### **Input Must Be Executed On Each Loop**

- Another implied statement checks to see if any data read statement was executed in the just-completed loop. If not, the step is terminated with a looping error.

```

data a;
if no_input_last_loop then stop
  put _n_ =;
  if _n_ ne 2 then set b;
  put _n_ =;

run;

```

```

_N_ =1
_N_ =1
_N_ =2
_N_ =2

```

NOTE: DATA STEP stopped due to looping.

**What If There Is No Input File?**

```
data a;
  if no_input_last_loop then stop
    put _n_=;
run;

_N_=1
```

NOTE: The data set WORK.A has 1 observations and 0 variables.

Note that the implied no-input check statement is still operative. If no INPUT or SET, etc. statement is present, no looping error message is generated.

**The "Real" Data Step So Far**

```
data a;
TOP-OF-DATA-STEP:
  if no_input_last_time then stop
  _N_ = start_count
    set b;
  if EOF then stop
  output
  goto top-of-DATA-step
run;
```

**Reset To Missing Values**

- Variables created in the step are assigned missing values: retain flag = no
- Variables present on input SAS data sets are protected from being reset: retain flag = yes
- The RETAIN FLAG attribute in the PDV determines if a variable will be reset.

Program Data Vector (PDV)									
Name	Label	Type	Length	Retain Flag	Missing Protect Flag	Keep Flag	Format	Inform	Value
X				YES					
Z				NO					

At the top of the DATA step, implied statements are added that reset variables to missing values. The general rule is: if a variable is created in the DATA step via an INPUT or assignment statement, it is reset to missing values. Variables present on input SAS data sets are not reset.

Data Set B	
OBS	X
1	1
2	2
3	3
4	4

```
data a;
  if not RESET_FLAG then set to missing
    put x= z=;
  set b;
  z = _n_;
run;
```

X=. Z=.  
X=1 Z=.  
X=2 Z=.  
X=3 Z=.  
X=4 Z=.

**Explicit Control Statements**

- OUTPUT
- DELETE
- subsetting IF
- RETURN

**OUTPUT Statement**

- Writes an observation to the output SAS data set.
- Signals the compiler not to add an implied OUTPUT statement at the end of the DATA step.

```
data a;
  put _n_=;
  set b;
  if _n_ = 2 then output;
goto TOP-OF-DATA-STEP
run;

_N_=1
_N_=2
_N_=3
_N_=4
_N_=5
```

Explicit OUTPUT statement

No implied output statement

NOTE: The data set WORK.A has 1 observations and 1 variables.

**DELETE Statement**

- The DELETE statement doesn't delete anything.
- The DELETE statement is simply a GOTO.

```
data a;
  set b;
  output;
  delete;
run;
```

```
data a;
  set b;
  output;
goto TOP-OF-DATA-STEP
goto TOP-OF-DATA-STEP
run;
```

DELETED statement

Implied GOTO

```
data a;
  set b;
goto TOP-OF-DATA-STEP
  output;
goto TOP-OF-DATA-STEP
run;
```

DELETED statement

Implied GOTO

NOTE: The data set WORK.A has 0 observations and 1 variables.

**Subsetting IF**

- The subsetting IF expands into an implied conditional DELETE statement: if <condition> then; else delete;

```
data a;
  set b;
  if _n_ = 1;
run;
```

NOTE: The data set WORK.A has 1 observations and 1 variables.

```
data a;
  set b;
if _N_ = 1 then;
else goto TOP-OF-DATA-STEP (delete)
run;
```

Subsetting IF expanded

NOTE: The data set WORK.A has 1 observations and 1 variables.

**RETURN Statement**

- The RETURN statement expands into OUTPUT and GOTO TOP-OF-DATA-STEP (aka DELETE).

```
data a;
  set b;
  put "here";
  return;
run;
```

```
data a;
  set b;
  put "here";
output
goto TOP-OF-DATA-STEP
run;
```

RETURN statement expanded

here  
here  
here  
here

NOTE: The data set WORK.A has 4 observations and 1 variables.

```
data a;
  set b;
output
goto TOP-OF-DATA-STEP
  put "here";
run;
```

RETURN statement expanded

NOTE: The data set WORK.A has 4 observations and 1 variables.

**OUTPUT and RETURN Statements**

- An explicit OUTPUT statement turns off the implied output that is part of the RETURN statement.

```
data a;
  set b;
  output;
  return;
  put "here";
run;
```

NOTE: The data set WORK.A has 4 observations and 1 variables.

**Implied RETURN**

- The implied OUTPUT and GOTO at the bottom of the DATA step can be treated as an implied RETURN

```
data a;
  set b;

return
  (output)
  (goto TOP-OF-DATA-STEP)

run;
```

**INFILE and INPUT Statements**

- INFILE: Names the file that the INPUT statement will read from.
- INPUT: Read a record and pull it apart

Line one  
Line two  
Line three  
Line four

```
filename tfile 'd:\test\input.txt';

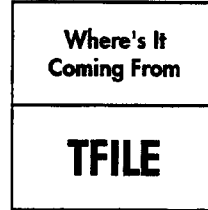
data c;
  infile tfile;
  input word1 $ word2 $;
run;

title "Data Set C";
proc print data=c;
run;
```

Data Set C		
OBS	WORD1	WORD2
1	Line	one
2	Line	two
3	Line	three
4	Line	four

**INFILE statement**

- INFILE is an executable statement.
- INFILE loads the WHERE'S-IT-COMING-FROM box.
- The WHERE'S-IT-COMING-FROM box is cleared at the top of the DATA step.



```
filename tfile 'd:\test\input.txt';

data c;
  load TFILE into WHERE'S-IT-COMING-FROM
  input word1 $ word2 $;
run;
```

INFILE statement

**INFILE Execution Required On Each Loop**

- WHERE'S-IT-COMING-FROM is reset at the top of the DATA step.

```
filename tfile 'd:\test\input.txt';

data c;
  reset WHERE'S-IT-COMING-FROM
  if _n_ = 1 then infile tfile;
  input word1 $ word2 $;
run;
```

ERROR: INPUT statement executed before INFILE statement.

NOTE: 1 record was read from the infile TFILE.

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

**INPUT Statement**

- Reads a record into the input buffer.
- Pulls characters (bytes) from input buffer and passes them through an informat.
- Loads the informat results into the PDV.
- Checks for end of file.

```

filename tfile 'd:\test\input.txt';

data c;
  infile tfile;
  read a record from WHERE'S-IT-COMING-FROM
  if at end of file then stop
  read fields
  pass field through informat into PDV
run;

```

INPUT statement  
expanded

**The Secret DATA step**

```

data c;
TOP-OF-DATA-STEP:
if no_input_last_loop then stop
  _N_ = start_count
reset WHERE'S-IT-COMING-FROM
if not RESET_FLAG then set to missing

  (infile tfile;)
load TFILE into WHERE'S-IT-COMING-FROM

  (input word1 $ word2 $;)
read a record from WHERE'S-IT-COMING-FROM
if at end of file then stop
read fields
pass field through informat into PDV

output
goto TOP-OF-DATA-STEP
run;

```

**Summary**

- Implied statements are added to your DATA step during the compile phase.
- Several explicit executable DATA step statements translate into two or more implied executable statements.
- Implied statements are executed in sequential order just like explicit statements you code yourself.
- Implied statements control the DATA step loop and normally determine when your DATA step stops.
- There are many other implied statements not discussed here. Some of them handle:
  - creating, opening, and closing files
  - BY statement processing
  - merging
  - INPUT and PUT statement trailing @
  - handling SET and INFILE options

**More Information**

Chapter 2, "The DATA step", SAS Language: Reference, Version 6, First Edition, pages 13-42

Tom Miron  
Miron InfoTec, Inc.  
118 S. Hancock St.  
Madison, WI 53703  
(608)255-3531