Paper 189-28

How SAS® Thinks or Why the DATA Step Does What It Does

Neil Howard, Ingenix, Basking Ridge, NJ

Abstract

The DATA step is the most powerful tool in the SAS® system. Understanding the internals of DATA step processing, what is happening and why, is crucial in mastering code an output. Concepts you will be introduced to:

- the Logical Program Data Vector (LPDV or PDV),
- automatic SAS variables and how are they used,
- the SAS Supervisor,
- understanding the internals of DATA step processing,
- what happens at program compile time,
- what's actually happening at execution time,
- how variable attributes are captured and stored.

This paper focuses on techniques that capitalize on the power of the DATA step and working with (and around) the default actions. Topics include:

- ➢ compile versus execution time activities;
- ➢ organizing your data to maximize execution;
- ➢ data defaults, data conversions;
- ➢ missing values, formatting values;
- ➢ ordering variables;
- ➢ functions for editing data and assigning values.

The Hands-On-Workshop will focus on DATA step COMPILE and EXECUTE, and the impact on your programming tasks. By understanding DATA step processing, you can debug your programs and interpret your results with confidence.

Introduction

SAS procedures are powerful and easy to use, but the DATA step offers the programmer a tool with almost unlimited potential. In the real world, we're lucky if systems are integrated, data is clean and system interfaces are seamless. The DATA step can help you, your programmers, your program, and your users perform better in the real world – especially when you take advantage of the available advanced features. Given that any of the topics/examples covered in this presentation have more than enough details, idiosyncrasies, and caveats to warrant its own tutorial, we will address selected essential processing tips and a range of "real world" situations that illustrate:

- ➢ DATA step compile and execute
- ➢ coding efficiencies
- ➢ maximizing execution
- ➢ data: type conversions and missing values
- ➢ other data issues
- ➢ data set management
- ➢ table lookup.

DATA Step Compile vs. Execute

There is a distinct compile action and execution for each DATA and PROC step in a SAS program. Each step is compiled, then executed, independently and sequentially. Understanding the defaults of each activity in DATA step processing is critical to achieving accurate results. During the compilation of a DATA step, the following actions (among others) occur:

- ➢ syntax scan
- ➢ SAS source code translation to machine language
- ➢ definition of input and output files
- ➢ creates:
  - ◊ input buffer (if reading any non-SAS data),
  - ◊ Program Data Vector (PDV),
  - ◊ and data set descriptor information
- ➢ set variable attributes for output SAS data set
- ➢ capture variables to be initialized to missing

Variables are added to the PDV in the order seen by the compiler during parsing and interpretation of source statements. Their attributes are determined at compile time by the first reference to the compiler. For numeric variables, the length is 8 during DATA step processing; length is an output property. Note that the last LENGTH or ATTRIB statement compiled determines the attributes.

The variables output to the SAS data set are determined at compile time; the automatic variables are never written, unless they have been assigned to SAS data set variables set up in the PDV (_N_, _ERROR_, end=, in=, point=, first., last., and implicit array indices); the variables written are specified by user written DROP and/or KEEP statements or data set options; the default being all non-automatic variables in the PDV. The output routines are also determined at compile time.

The following statements are compile-time only statements. They provide information to the PDV, and cannot by default (except in the macro language) be conditionally executed. Placement of the last six statements (shown below) is critical because the attributes of variables are determined by the first reference to the compiler:

- ➢ drop, keep, rename
- ➢ label
- ➢ retain
- ⇒ length
- ⇒ format, informat
- ⇒ attrib
- ⇒ array
- ⇒ by

⇒ where

Once compilation has completed, the DATA step is executed: the I/O engine supervisor optimizes the executable image by controlling looping, handling the initialize-to-missing instruction, and identifying the observations to be read. Variables in the PDV are initialized, the DATA step program is called, the user-controlled DATA step machine code statements are executed, and the default output of observations is handled.

By understanding the default activities of the DATA step, the SAS programmer can make informed and intelligent coding decisions. Code will be more flexible and efficient, debugging will be straightforward and make more sense, and program results can be interpreted readily.

This workshop will focus on the compile and execute activities of a variety of DATA step activities.

Additional DATA Step Tips:

Coding Efficiencies & Maximizing Execution

The SAS system affords the programmer a multitude of choices in coding the DATA step. The key to optimizing your code lies in recognizing the options and understanding the implications. This may not feel like advanced information, but the application of these practices has far-reaching effects.

Permanently store data in SAS data sets. The SET statement is dramatically more efficient for reading data in the DATA step than any form of the INPUT statement (list, column, formatted). SAS data sets offer additional advantages, most notably the self-documenting aspects and the ability to maintain them with procedures such as DATASETS. And they can be passed directly to other program steps.

A “shell” DATA step can be useful. Code declarative, compile-only statements (LENGTH, RETAIN, ARRAY) grouped, preceding the executable statements. Block-code other non-executables like DROP, KEEP, RENAME, ATTRIB, LABEL statements following the executable statements. Use of this structure will serve as a check list for the housekeeping chores and consistent location of important information. Use consistent case, spaces, indentation, and blank lines liberally for readability and to isolate units of code or to delineate DO-END constructions.

Use meaningful names for data sets and variables, and use labels to enhance the output. Comment as you code; titles and footnotes enhance an audit trail. Based on your understanding of the data, code IF-THEN-ELSE or SELECT statements in order of probability of execution. Execute only the statements you need, in the order that you need them. Read and write data (variables and observations) selectively, reading selection fields first, using DROP/KEEP, creating indexes. Prevent unnecessary processing. Avoid GOTOs. Simplify logical expressions and complex calculations, using parentheses to highlight precedence and for clarification. Use DATA step functions for their simplicity and arrays for their ability to compact code and data references.

Data Conversions

Character to numeric, and numeric to character, conversions occur when:

- incorrect argument types passed to function
- comparisons of unlike type variables occur
- performing type-specific operations (arithmetic) or concatenation (character)

SAS will perform default conversions where necessary and possible, but the programmer should handle all conversions to insure accuracy. The following code illustrates:

- default conversion,
- numeric-to-character conversion using PUT function,
- character-to-numeric conversion with INPUT function:

```
data convert1;
  length x $ 2 y $ 1;
  set insas; *contains numeric variables flag and code;
  x = flag;
  y = code;
run;

data convert2;
  length x $ 2 y 8
  set insas; *contains numeric variables flag and code;
  x = put(flag, 2.);
  y = input(put(code, 1.), 8.);
run;

data convert3;
  length z 2;
  set insas; *contains character variable status;
  z = input(status, 2.);
run;
```

Missing Data

The DATA step provides many opportunities for serious editing of data and handling unknown, unexpected, or missing values. When a programmer is anticipating these conditions, it is straightforward to detect and avoid missing data; treat missing data as acceptable within the scope of an application; and even capitalize on the presence of missing data. When a value is stored as "missing" in a SAS data set, its value is the equivalent of negative infinity, less than any other value that could be present. Numeric missings are represented by a "." (a period); character by " " (blank). Remember this in range checking and recoding. Explicitly handle missing data in IF-THEN-ELSE constructions; in PROC FORMATs used for recoding; and in calculations. The first statement in the following example:

```
if age < 8 then agegroup = "child";
if agegroup = " " then delete;
```

will include any observations where age is missing in the agegroup "child". This may or may not be appropriate for your application. A better statement might be:

```
if (. < age < 8) then agegroup = "child";
```

Depending on the user's application, it may be appropriate to distinguish between different types of missing values encountered in the data. Take advantage of the twenty-eight special missing values:

```
        .              ._              .A - .Z
```

```
if comment = "unknown" then age = .;
else if comment = "refused to answer" then age = .A;
else if comment = "don't remember" then age = .B;
```

All these missing values test the same. Once a missing value has resulted or been assigned, it stays with the data, unless otherwise changed during some stage of processing. It is possible to test for the presence of missing data with the N and NMISS functions:

```
y = nmiss(age, height, weight, name);
    ** y contains the number of nonmissing arguments;

z = n(a,b,c,d);
    ** z contains the number of missings in the list;
```

Within the DATA step, the programmer can encounter missing data in arithmetic operations. Remember that in simple assignment statements, missing values propagate from the right side of the equal sign to the left; if any argument in the expression on right is missing, the result on the left will be missing. Watch for the "missing values generated" messages in the SAS log. Although DATA step functions assist in handling missing values, it is important to understand their defaults as well. Both the SUM and MEAN functions ignore missing values in calculations: SUM will add all the non-missing arguments and MEANS will add the nonmissings and divide by the number of nonmissings. If all the arguments to SUM or MEANS are missing, the result of the calculations will be missing. This works for MEAN, but not for SUM, particularly if the intention is to use the result in a later calculation:

```
x = a + b + c; * if any argument is missing, x = . ;
x = SUM(a,b,c); *with missing argument, x is sum of nonmissings;
x = SUM(a,b,c,0); * if a,b,c are missing, result will be zero;
y = (d + e + f + g) / 4; *number of nonmissings is divided by 4;
y = MEAN(d,e,f,g); * if all argument s are missing, y = . ;
```

Since there are 90+ DATA step functions, the moral of the function story is to research how each handles missing values.