The Basics of the SAS Language

SAS Statements

A *SAS statement* is a type of SAS language element that is used to perform a particular operation in a SAS program or to provide information to a SAS program. SAS statements are free-format. This means that they can begin and end anywhere on a line, that one statement can continue over several lines, and that several statements can be on the same line. Blank or special characters separate words in a SAS statement.

TIP You can specify SAS statements in uppercase or lowercase. In most situations, text that is enclosed in quotation marks is case sensitive.

Here are two important rules for writing SAS programs:

- A SAS statement ends with a semicolon.
- A statement usually begins with a SAS keyword.

There are two types of SAS statements:

- statements that are used in DATA and PROC steps
- statements that are global in scope and can be used anywhere in a SAS program

Global Statements

Global statements are used anywhere in a SAS program and stay in effect until changed or canceled, or until the SAS session ends. Here are some common global statements: TITLE, LIBNAME, OPTIONS, and FOOTNOTE.

DATA Step

The *DATA step* creates or modifies data. The input for a DATA step can be of several types, such as raw data or a SAS data set. The output from a DATA step can be of several types, such as a SAS data set or a report. A *SAS data set* is a data file that is formatted in a way that SAS can understand.

For example, you can use DATA steps to do the following:

- put your data into a SAS data set
- compute values

- check for and correct errors in your data
- produce new SAS data sets by subsetting, supersetting, merging, and updating existing data sets

PROC Step

The *PROC step* analyzes data, produces output, or manages SAS files. The input for a PROC (procedure) step is usually a SAS data set. The output from a PROC step can be of several types, such as a report or an updated SAS data set.

For example, you can use PROC steps to do the following:

- create a report that lists the data
- analyze data
- create a summary report
- produce plots and charts

SAS Program Structure

A SAS program consists of a sequence of steps. A program can be any combination of DATA or PROC steps. A step is a sequence of SAS statements.

Here is an example of a simple SAS program.

Example Code 1 *A Simple SAS Program*

```
title1 'June Billing for Patients Older Than 39';     /*#1*/
data work.junefee;                                    /*#2*/
  set cert.admitjune;
  where age>39;
run;                                                  /*#3*/
proc print data=work.junefee;                         /*#4*/
run;
```

1 The TITLE statement is a global statement. Global statements are typically outside steps and do not require a RUN statement.

2 The DATA step creates a new SAS data set named Work.JuneFee. The SET statement reads in the data from Cert.AdmitJune. The new data set contains only those observations whose value for Age is greater than 39.

3 If a RUN or QUIT statement is not used at the end of a step, SAS assumes that the beginning of a new step implies the end of the previous step. If a RUN or QUIT statement is not used at the end of the last step in a program, SAS Studio and SAS Enterprise Guide automatically submit a RUN and QUIT statement after the submitted code.

4 The PROC PRINT step prints a listing of the new SAS data set. A PROC step begins with a PROC statement, which begins with the keyword PROC.

Output 2.1 *PRINT Procedure Output*

June Billing

Obs	ID	Fee	WriteOff	Insurance	PatientPay
1	2588	$85.20	$8.52	$61.34	$15.34
2	2586	$85.20	$8.52	$61.34	$15.34
3	2458	$85.20	$8.52	$61.34	$15.34
4	2572	$85.20	$8.52	$61.34	$15.34
5	2544	$124.80	$12.48	$89.86	$22.46
6	2574	$149.75	$14.98	$107.82	$26.95
7	2501	$149.75	$14.98	$107.82	$26.95
8	2552	$149.75	$14.98	$107.82	$26.95
9	2462	$124.80	$12.48	$89.86	$22.46
10	2563	$124.80	$12.48	$89.86	$22.46
11	2555	$149.75	$14.98	$107.82	$26.95
12	2575	$124.80	$12.48	$89.86	$22.46
13	2589	$149.75	$14.98	$107.82	$26.95
14	2523	$149.75	$14.98	$107.82	$26.95
15	2584	$124.80	$12.48	$89.86	$22.46
16	2571	$149.75	$14.98	$107.82	$26.95
17	2578	$124.80	$12.48	$89.86	$22.46
18	2568	$124.80	$12.48	$89.86	$22.46
19	2539	$124.80	$12.48	$89.86	$22.46
20	2595	$149.75	$14.98	$107.82	$26.95
21	2579	$149.75	$14.98	$107.82	$26.95

Processing SAS Programs

When a SAS program is submitted for execution, SAS first validates the syntax and then compiles the statements. DATA and PROC statements signal the beginning of a new step. The beginning of a new step also implies the end of the previous step. At a step boundary, SAS executes any statement that has not been previously executed and ends the step.

Example Code 2 *Processing SAS Programs*

```
data work.admit2;                /*#1*/
  set cert.admit;
  where age>39;
proc print data=work.admit2;     /*#2*/
run;                             /*#3*/
```

1 The DATA step creates a new SAS data set named Work.Admit2 by reading Cert.Admit. The DATA statement is the beginning of the new step. The SET statement is used to read data. The WHERE statement conditionally reads only the observations where the value of the variable Age is greater than 39.

2 The PROC PRINT step prints the new SAS data set named Work.Admit2. The PROC PRINT statement serves as a step boundary in this example because a RUN statement was not used at the end of the DATA step. The PROC step also implies the end of the DATA step.

3 The RUN statement ends the PROC step.

TIP The RUN statement is not required between steps in a SAS program. However, it is a best practice to use a RUN statement because it can make the SAS program easier to read and the SAS log easier to understand when debugging.

Log Messages

The SAS log collects messages about the processing of SAS programs and about any errors that occur. Each time a step is executed, SAS generates a log of the processing activities and the results of the processing.

When SAS processes the sample program, it produces the log messages shown below. Notice that you get separate sets of messages for each step in the program.

Log 2.1 *SAS Log Messages for Each Program Step*

```
5    data work.admit2;
6        set cert.admit;
7        where age>39;
8    run;

NOTE: There were 10 observations read from the data set CERT.ADMIT.
      WHERE age>39;
NOTE: The data set WORK.ADMIT2 has 10 observations and 9 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

9    proc print data=work.admit2;
NOTE: Writing HTML Body file: sashtml.htm
10   run;

NOTE: There were 10 observations read from the data set WORK.ADMIT2.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.35 seconds
      cpu time            0.24 seconds
```

Results of Processing

The DATA Step

Suppose you submit the sample program below:

```
data work.admit2;
  set cert.admit;
  where age>39;
run;
```

When the program is processed, it creates a new SAS data set, Work.Admit2, containing only those observations with age values greater than 39. The DATA step creates a new data set and produces messages in the SAS log, but it does not create a report or other output.

The PROC Step

If you add a PROC PRINT step to this same example, the program produces the same new data set as before, but it also creates the following report:

```
data work.admit2;
  set cert.admit;
  where age>39;
run;
proc print data=work.admit2;
run;
```

Figure 2.1 *PRINT Procedure Output*

The SAS System

Obs	ID	Name	Sex	Age	Date	Height	Weight	ActLevel	Fee
1	2523	Johnson, R	F	43	31	63	137	MOD	149.75
2	2539	LaMance, K	M	51	4	71	158	LOW	124.80
3	2568	Eberhardt, S	F	49	27	64	172	LOW	124.80
4	2571	Nunnelly, A	F	44	19	66	140	HIGH	149.75
5	2575	Quigley, M	F	40	8	69	163	HIGH	124.80
6	2578	Cameron, L	M	47	5	72	173	MOD	124.80
7	2579	Underwood, K	M	60	22	71	191	LOW	149.75
8	2584	Takahashi, Y	F	43	29	65	123	MOD	124.80
9	2589	Wilcox, E	F	41	16	67	141	HIGH	149.75
10	2595	Warren, C	M	54	7	71	183	MOD	149.75

Other Procedures

SAS programs often invoke procedures that create output in the form of a report, as is the case with the FREQ procedure:

```
proc freq data=sashelp.cars;
  table origin*DriveTrain;
run;
```

Figure 2.2 *FREQ Procedure Output*

The FREQ Procedure

Frequency
Percent
Row Pct
Col Pct

Table of Origin by DriveTrain

Origin	DriveTrain All	DriveTrain Front	DriveTrain Rear	DriveTrain Total
Asia	34 7.94 21.52 36.96	99 23.13 62.66 43.81	25 5.84 15.82 22.73	158 36.92
Europe	36 8.41 29.27 39.13	37 8.64 30.08 16.37	50 11.68 40.65 45.45	123 28.74
USA	22 5.14 14.97 23.91	90 21.03 61.22 39.82	35 8.18 23.81 31.82	147 34.35
Total	92 21.50	226 52.80	110 25.70	428 100.00

Other SAS programs perform tasks such as sorting and managing data, which have no visible results except for messages in the log. (All SAS programs produce log messages, but some SAS programs produce only log messages.)

```
proc copy in=cert out=work;
   select admit;
run;
```

Log 2.2 *SAS Log: COPY Procedure Output*

```
11   proc copy in=cert out=work;
12       select admit;
13   run;

NOTE: Copying CERT.ADMIT to WORK.ADMIT (memtype=DATA).
NOTE: There were 21 observations read from the data set CERT.ADMIT.
NOTE: The data set WORK.ADMIT has 21 observations and 9 variables.
NOTE: PROCEDURE COPY used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds
```

Error Messages

Types of Errors

SAS can detect several types of errors. Here are two common ones:

- syntax errors that occur when program statements do not conform to the rules of the SAS language
- semantic errors that occur when you specify a language element that is not valid for a particular usage

Syntax Errors

When you submit a program, SAS scans each statement for syntax errors, and then executes the step (if no syntax errors are found). SAS then goes to the next step and repeats the process. Syntax errors, such as misspelled keywords, generally prevent SAS from executing the step in which the error occurred.

Notes are written to the SAS log when the program finishes executing. When a program that contains an error is submitted, messages about the error appear in the SAS log. Here is what SAS does:

- displays the word ERROR
- identifies the possible location of the error
- gives an explanation of the error

Example: Syntax Error Messages

The following program contains a syntax error:

```
data work.admitfee;              /* #1 */
  set cert.admit;
run;
proc prin data=work.admitfee;    /* #2 */
  var id name actlevel fee;      /* #3 */
run;
```

1 The DATA step creates a new SAS data set named Work.Admitfee from the Cert.Admit data set.

2 The SAS keyword PRINT in PROC PRINT is spelled incorrectly. As a result, the PROC step fails.

3 The VAR statement prints the values for the following variables only: ID, Name, ActLevel, and Fee.

When the program is submitted, messages in the SAS log indicate that the procedure PRIN was not found and that SAS stopped processing the PROC step because of errors. No output is produced by the PRINT procedure, because the second step fails to execute.

Here is an explanation of the following log.

- The ERROR keyword is the notification of the error.
- The PRIN keyword in the SAS log is the possible location of the error in the statement.
- The error explanation is **`not found`**.

Log 5.1 *SAS Log*

```
265  proc prin data=work.admitfee;
ERROR: Procedure PRIN not found.
268    var id name actlevel fee;
267  run;
NOTE: The SAS System stopped processing this step because of errors.
```

TIP Errors in your statements or data might not be evident when you look at results in the Results viewer. Review the messages in the SAS log each time you submit a SAS program.

In addition to correcting spelling mistakes, you might need to resolve other common syntax errors such as these:

- missing RUN statement
- missing semicolon
- unbalanced quotation mark

You might also need to correct a semantic error such as this:

- invalid option

Correcting Common Errors

The Basics of Error Correction

To correct simple errors, such as the spelling error here, type over the incorrect text, delete text, or insert text. In the following program, the incorrect spelling of PRINT in the PROC step is corrected.

```
data work.admitfee;
  set cert.admit;
run;
proc print data=work.admitfee;
  var id name actlevel fee;
run;
```

Resubmitting a Revised Program

After correcting your program, you can resubmit it.

Figure 5.1 *Correct PRINT Procedure Output*

The SAS System

Obs	ID	Name	ActLevel	Fee
1	2458	Murray, W	HIGH	85.20
2	2462	Almers, C	HIGH	124.80
3	2501	Bonaventure, T	LOW	149.75
4	2523	Johnson, R	MOD	149.75
5	2539	LaMance, K	LOW	124.80
6	2544	Jones, M	HIGH	124.80
7	2552	Reberson, P	MOD	149.75
8	2555	King, E	MOD	149.75
9	2563	Pitts, D	LOW	124.80
10	2568	Eberhardt, S	LOW	124.80
11	2571	Nunnelly, A	HIGH	149.75
12	2572	Oberon, M	LOW	85.20
13	2574	Peterson, V	MOD	149.75
14	2575	Quigley, M	HIGH	124.80
15	2578	Cameron, L	MOD	124.80
16	2579	Underwood, K	LOW	149.75
17	2584	Takahashi, Y	MOD	124.80
18	2586	Derber, B	HIGH	85.20
19	2588	Ivan, H	LOW	85.20
20	2589	Wilcox, E	HIGH	149.75
21	2595	Warren, C	MOD	149.75

Remember to check the SAS log again to verify that your program ran correctly.

Log 5.2 *SAS Log: No Error Messages*

```
9231  data work.admitfee;
9232      set cert.admit;
9233  run;

NOTE: There were 21 observations read from the data set CERT.ADMIT.
NOTE: The data set WORK.ADMITFEE has 21 observations and 9 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


9234  proc print data=work.admitfee;
9235      var id name actlevel fee;
9236  run;

NOTE: There were 21 observations read from the data set WORK.ADMITFEE.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

The Basics of Logic Errors

The PUTLOG Statement

A *logic error* occurs when the program statements execute, but produce incorrect results. Because no notes are written to the log, logic errors are often difficult to detect. Use the PUTLOG statement in the DATA step to write messages to the SAS log to help identify logic errors.

Syntax, PUTLOG statement

PUTLOG '*message*'**;**

message specifies the message that you want to write to the SAS log. It can include character literals, variable names, formats, and pointer controls.

Note: You can precede your message text with WARNING, MESSAGE, or NOTE to better identify the output in the log.

The PUTLOG statement can be used to write to the SAS log in both batch and interactive modes. If an external file is open for output, use this statement to ensure that debugging messages are written to the SAS log and not to the external file.

Temporary Variables

The temporary variables _N_ and _ERROR_ can be helpful when you debug a DATA step.

Variable	Description	Debugging Use
_N_	The number of times the DATA step iterated	Displays debugging messages for a specified number of iterations of the DATA step

Variable	Description	Debugging Use
ERROR	Initialized to 0, set to 1 when an error occurs	Displays debugging messages when an error occurs

Example: The DATA Step Produces Wrong Results but There Are No Error Messages

The data set contains three test scores and homework grades for four students. The program below is designed to select students whose average score is below 70. Although the program produces incorrect results, there are no error messages in the log.

```
data work.grades;
  set cert.class;
  Homework=Homework*2;
  AverageScore=MEAN(Score1 + Score2 + Score3 + Homework);
    if AverageScore<70;
run;
```

A glance at the data set shows that there should be students whose mean scores are below 70. However, the data set Work.Grades has zero observations and six variables.

```
NOTE: There were 4 observations read from the data set
      CERT.CLASS.
NOTE: The data set WORK.GRADES has 0 observations and 6
      variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Use the PUTLOG statement to determine where the DATA step received incorrect instructions. Place the PUTLOG statement before the subsetting IF.

```
PUTLOG Name= Score1= Score2= Score3= Homework= AverageScore=;
```

```
29457  data work.grades;
29458       set cert.class;
29459       Homework=Homework*2;
29460       AverageScore=MEAN(Score1 + Score2 + Score3 +
29460! Homework);
29461       putlog Name= Score1= Score2= Score3= Homework=
29461! AverageScore=;
29462       if AverageScore<70;
29463  run;

Name=LINDA Score1=53 Score2=60 Score3=66 Homework=84
AverageScore=263
Name=DEREK Score1=72 Score2=64 Score3=56 Homework=64
AverageScore=256
Name=KATHY Score1=98 Score2=82 Score3=100 Homework=96
AverageScore=376
Name=MICHAEL Score1=80 Score2=55 Score3=95 Homework=100
AverageScore=330
NOTE: There were 4 observations read from the data set
      CERT.CLASS.
NOTE: The data set WORK.GRADES has 0 observations and 6
      variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Looking at the log, you can see the result of the PUTLOG statement. The data that is listed in the middle of the log shows that the variables were read in properly, and the variable Homework was adjusted to be weighted the same as Scores1–3. However, the values of AverageScore are incorrect. They are above the available maximum grade.

There is a syntax error in the line that computes AverageScore: Instead of commas separating the three score variables in the MEAN function, there are plus signs. Since functions can contain arithmetic expressions, SAS simply added the four variables together, as instructed, and computed the mean of a single number. That is why no observations had values of AverageScore below 70.

To fix the error, replace the plus signs in the MEAN function with commas. You can remove the PUTLOG statement and use a PROC PRINT statement to view your results.

```
data work.grades;
  set cert.class;
  Homework=Homework*2;
  AverageScore = MEAN(Score1, Score2, Score3, Homework);
    if AverageScore < 70;
run;
proc print data=work.grades;
run;
```

The figure below lists the names of students whose average score is below 70.

Figure 5.2 *Corrected Program Output*

Name	Score1	Score2	Score3	Homework	AverageScore
LINDA	53	60	66	84	65.75
DEREK	72	64	56	64	64.00

PUT Statement

Syntax

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. For more information about IF-THEN/ELSE statements, see Chapter 9, “Creating and Managing Variables,” on page 137.

Syntax, PUT statement:

PUT *specification(s)***;**

specification specifies what is written, how it is written, and where it is written. Here are examples:

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

Example: Using the PUT Statement

The following example illustrates how to use the PUT statement to write messages to the SAS log.

```
data work.test;
  set cert.loan01;
  if code='1' then type='variable';                                  /*#1*/
    else if code='2' then type='fixed';
    else type='unknown';
  if type='unknown' then put 'MY NOTE: invalid value: ' code=;       /*#2*/
run;
```

1 If the value of the variable Code equals **1**, then the program returns the value for Type as **variable**. If the value equals **2**, then the return value for Type is **fixed**. Otherwise, the value of Type is returned as **unknown**.

2 If Type contains the value **unknown**, then the PUT statement writes a message to the log.

Log 5.3 *SAS Log*

```
NOTE: Character values have been converted to numeric
      values at the places given by: (Line):(Column).
      148173:11   148174:18
MY NOTE: invalid value: Code=3
MY NOTE: invalid value: Code=3
MY NOTE: invalid value: Code=3
```

Example: 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.

```
data work.loan01;
  set cert.loan;
  if code='1' then type='variable';
    else if code='2' then type='fixed';
    else type='unknown';
  put 'MY NOTE: The condition was met.';
run;
```

The following is printed to the SAS log.

Log 5.4 *SAS Log*

```
MY NOTE: The condition was met.
MY NOTE: The condition was met.
MY NOTE: The condition was met.
MY NOTE: The condition was met.
MY NOTE: The condition was met.
MY NOTE: The condition was met.
MY NOTE: The condition was met.
MY NOTE: The condition was met.
MY NOTE: The condition was met.
NOTE: There were 9 observations read from the data set
      CERT.LOAN.
NOTE: The data set WORK.LOAN01 has 9 observations and 6
      variables.
```

Example: 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.

Note: 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 to the log, add an equal sign (=) to the variable name.

```
data work.loan01;
  set cert.loan;
  if code='1' then type='variable';
    else if code='2' then type='fixed';
    else type='unknown';
  put 'MY NOTE: Invalid Value: ' code=  type= ;
run;
```

The following is printed to the SAS log.

Log 5.5 *SAS Log*

```
MY NOTE: Invalid Value: Code=1 type=variable
MY NOTE: Invalid Value: Code=1 type=variable
MY NOTE: Invalid Value: Code=1 type=variable
MY NOTE: Invalid Value: Code=2 type=fixed
MY NOTE: Invalid Value: Code=2 type=fixed
MY NOTE: Invalid Value: Code=2 type=fixed
MY NOTE: Invalid Value: Code=3 type=unknown
MY NOTE: Invalid Value: Code=3 type=unknown
MY NOTE: Invalid Value: Code=3 type=unknown
NOTE: There were 9 observations read from the data set
      CERT.LOAN.
NOTE: The data set WORK.LOAN01 has 9 observations and 6
      variables.
```

Example: 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 work.newcalc;
  set cert.loan;
  if rate>0 then Interest=amount*(rate/12);
    else put 'DATA ERROR: ' rate= _n_ = ;
run;
```

The following is printed to the SAS log:

Log 5.6 *SAS Log*

```
DATA ERROR: Rate=. _N_=7
NOTE: There were 10 observations read from the data set
      CERT.LOAN.
NOTE: The data set WORK.NEWCALC has 10 observations and 5
      variables.
```

Missing RUN Statement

Each step in a SAS program is compiled and executed independently from every other step. As a step is compiled, SAS recognizes the end of the current step when it encounters one of the following statements:

- a DATA or PROC statement, which indicates the beginning of a new step
- a RUN or QUIT statement, which indicates the end of the current step

  Note: The QUIT statement ends some SAS procedures.

```
data work.admitfee;                /* #1 */
  set cert.admit;
proc print data=work.admitfee;     /* #2 */
  var id name actlevel fee;
                                   /* #3 */
```

1 Even though there is no RUN statement after the DATA step, the DATA step executes because the PROC step acts as a step boundary.

2 The PROC step does not execute. There is no following RUN statement for the step, nor is there a DATA or PROC step following the PROC PRINT step. Therefore, there is no indication that the step has ended.

3 The RUN statement is necessary at the end of the last step. If the RUN statement is omitted from the last step, the program might not complete processing and might produce unexpected results.

If you are programming in Enterprise Guide or SAS Studio, the system submits a RUN statement after every program that you submit, so the above program would execute normally.

Note: Although omitting a RUN statement is not technically an error, it can produce unexpected results. A best practice is to always end a step with a RUN statement.

To correct the error, submit a RUN statement at the end of the PROC step.

```
run;
```

Missing Semicolon

One of the most common errors is a missing semicolon at the end of a statement. Here is an example:

```
data work.admitfee;
  set cert.admit;
run;
proc print data=work.admitfee
  var id name actlevel fee;
run;
```

When you omit a semicolon, SAS reads the statement that lacks the semicolon (along with the following statement) as one long statement.

Log 5.7 *SAS Log: Error Messages*

```
9240  proc print data=work.admitfee
9241       var id name actlevel fee;
           ---
           22
           76
ERROR 22-322: Syntax error, expecting one of the following: ;, (, BLANKLINE,
CONTENTS, DATA,
              DOUBLE, GRANDTOTAL_LABEL, GRANDTOT_LABEL, GRAND_LABEL,
GTOTAL_LABEL, GTOT_LABEL,
              HEADING, LABEL, N, NOOBS, NOSUMLABEL, OBS, ROUND, ROWS, SPLIT,
STYLE, SUMLABEL,
              UNIFORM, WIDTH.
ERROR 76-322: Syntax error, statement will be ignored.
9242  run
```

Correcting the Error: Missing Semicolon

1. Find the statement that lacks a semicolon. You can usually find it by looking at the underscored keywords in the error message and working backward.
2. Add a semicolon in the appropriate location.
3. Resubmit the corrected program.
4. Check the SAS log again to make sure there are no other errors.

Unbalanced Quotation Marks

Some syntax errors, such as the missing quotation mark after **`HIGH`** in the program below, cause SAS to misinterpret the statements in your program.

```
data work.admitfee;
  set cert.admit;
  where actlevel='HIGH;
run;
proc print data=work.admitfee;
  var id name actlevel fee;
run;
```

When the program is submitted, SAS is unable to resolve the DATA step, and a `DATA STEP running` message appears at the top of the active window.

TIP Both SAS Enterprise Guide and SAS Studio add a final line of code to stop unbalanced quotation marks.

Sometimes a warning appears in the SAS log that indicates the following:

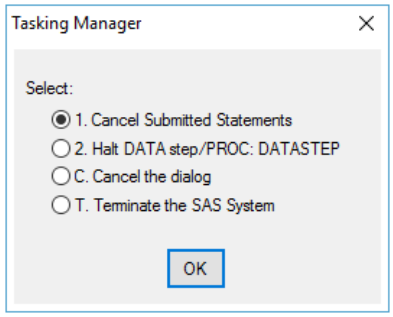
- A quoted string has become too long.
- A statement that contains quotation marks (such as a TITLE or FOOTNOTE statement) is ambiguous because of invalid options or unquoted text.

When you have unbalanced quotation marks, SAS is often unable to detect the end of the statement in which it occurs. In Enterprise Guide or SAS Studio, simply add the balancing quotation mark and resubmit the program. However, in some environments, this technique usually does not correct the error. SAS still considers the quotation marks to be unbalanced.

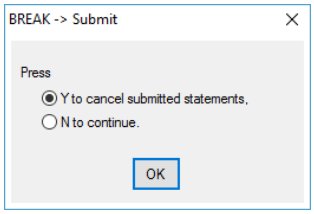
Therefore, you need to resolve the unbalanced quotation mark by canceling the submitted statements (in the Windows and UNIX operating environments) or by submitting a line of SAS code (in the z/OS operating environment) before you recall, correct, and resubmit the program.

Correcting the Error in the Windows Operating Environment

1. Press the Ctrl and Break keys or click the Break Icon on the toolbar.
2. Select **1. Cancel Submitted Statements**, and then click **OK**.



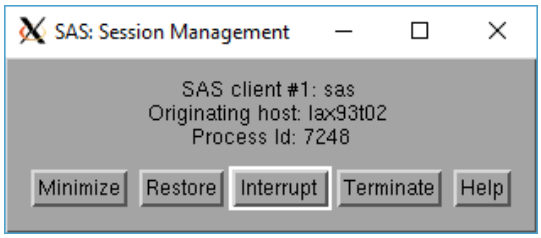
3. Select **Y to cancel submitted statements**, and then click **OK**.



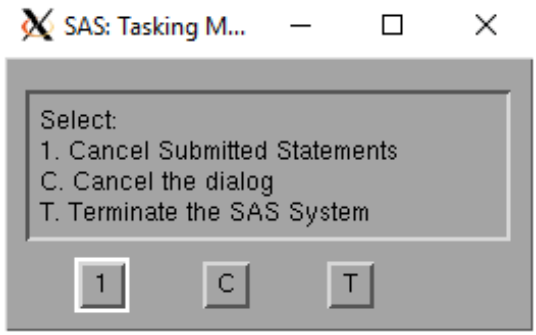
4. Correct the error and resubmit the program.

Correcting the Error in the UNIX Environment

1. Open the Session Management window and click **Interrupt**.



2. Select **1. Cancel Submitted Statements**, and then click **Y**.



3. Correct the error and resubmit the program.

Correcting the Error in the z/OS Operating Environment

1. Submit an asterisk followed by a single quotation mark, a semicolon, and a RUN statement.

```
*'; run;
```

2. Delete the line that contains the asterisk followed by the single quotation mark, the semicolon, and the RUN statement.
3. Insert the missing quotation mark in the appropriate place.
4. Submit the corrected program.

TIP You can also use the above method in the Windows and UNIX operating environments.

Semantic Error: Invalid Option

An invalid option error occurs when you specify an option that is not valid in a particular statement. In the program below, the KEYLABEL option is not valid when it is used with the PROC PRINT statement.

```
data work.admitfee;
  set cert.admit;
  where weight>180 and (actlevel='MOD' or actlevel='LOW');
run;
proc print data=cert.admit keylabel;
  label actlevel='Activity Level';
run;
```

When a SAS statement that contains an invalid option is submitted, a message appears in the SAS log indicating that the option is not valid or not recognized.

Log 5.8 *SAS Log: Syntax Error Message*

```
9254  proc print data=cert.admit keylabel;
                                    --------
                                    22
                                    202
ERROR 22-322: Syntax error, expecting one of the following: ;, (, BLANKLINE,
CONTENTS, DATA,
              DOUBLE, GRANDTOTAL_LABEL, GRANDTOT_LABEL, GRAND_LABEL,
GTOTAL_LABEL, GTOT_LABEL,
              HEADING, LABEL, N, NOOBS, NOSUMLABEL, OBS, ROUND, ROWS, SPLIT,
STYLE, SUMLABEL,
              UNIFORM, WIDTH.
ERROR 202-322: The option or parameter is not recognized and will be ignored.
9255      label actlevel='Activity Level';
9256  run;

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

Correcting the Error: Invalid Option

1. Remove or replace the invalid option, and check your statement syntax as needed.
2. Resubmit the corrected program.
3. Check the SAS log again to make sure there are no other errors.