

# SAS Completion Codes to Make Complex Programs Run Smoothly

## Heidi Markovitz

**Abstract:** Many SAS operations set codes that describe whether the task was a success and what the operation accomplished. Capturing these codes in your programs enables you to (1) halt execution as soon as a failure occurs, (2) produce meaningful error and instructional messages, and (3) conditionally execute DATA or PROC steps depending on the outcome of earlier steps. Using them you can turn your collection of steps into a cohesive program.

### The Problem:

SAS® programs are usually composed of several discrete steps. Later steps are frequently dependent on the success of earlier steps. When an early step in a long program fails, the program may continue processing. This can lead to replacing good data with bad, inaccurate reports, and very difficult recovery processes.

To illustrate, here is a simple program that may be part of a daily process. It creates a SAS data set from raw data and prints the data set. Each time the program is run, it replaces the daily data set.

```
DATA permlib.dailyfile;
  INFILE rawdata;
  INPUT trans_id $10.
         trans_amt 8.2;
  IF trans_amt > .
    THEN OUTPUT;
RUN;
PROC PRINT DATA=permlib.dailyfile;
  SUM trans_amt;
RUN;
```

What happens if the input file, *rawdata*, does not exist? The data step will end without creating a new *permlib.dailyfile*. Depending on your system options, the PROC PRINT may just display a generic error in the log or it may produce a listing of the previous day's data. In the first case, you would sift through the log to find that the PROC PRINT error was really a missing data problem from the previous step. In the latter case, you would first need to notice that the report was not correct and then dig through the log. If the report had been automatically distributed to an audience, that error in the first step of the program would probably take longer to find and be more embarrassing to fix.

Fig.1 Create data set and print. Successful.

Imagine a similar problem in a program with 20 steps dealing with 10 data sources and 4 different reports. How could you run such a thing with confidence, especially if you were not watching every step?

### Solutions:

What is needed is a way to proceed from step to step conditioned on whether previous steps worked as expected. If a step fails, the system could promptly notify staff and possibly stop the process or take corrective action. The SAS System includes a variety of features, including automatic macro variables, functions, and SAS Macro Language, that enable us to schedule the most complex operations while secure in the knowledge that unexpected minor problems will not bloom into major catastrophes. What follows are a few programming examples that incorporate these features and a non-exhaustive listing of SAS elements that can be used for error handling.

### Automatic Macro Variables

Many automatic macro variables provide information on how a program is doing. For instance, a return code is inserted into **syslibrc** whenever a LIBNAME statement or LIBNAME function is executed (see Fig. 2). Automatic macro variable **sysfilrc** is set whenever a FILENAME is defined. When would these be useful?

```
LIBNAME permlib '\wp\docs\sesug02\mypaper';
NOTE: Libref PERMLIB was successfully assigned as follows:
      Engine:          V8
      Physical Name:   C:\wp\docs\sesug02\mypaper
%PUT 1. LIBNAME for valid directory: syslibrc=&syslibrc;
1. LIBNAME for valid directory: syslibrc=0

LIBNAME permlib '\wp\docs\sesug02\nopaper';
NOTE: Library PERMLIB does not exist.
%PUT 2. LIBNAME for non-existent directory:
syslibrc=&syslibrc;
2. LIBNAME for non-existent directory: syslibrc=-70008
```

Consider a program that accesses a library which is regenerated on a regular basis. Occasionally, the library might be missing. To avoid major problems, the program would verify that the LIBNAME statement was successful before using the data in the library. It would do this by checking the value of **syslibrc**.

Fig. 2 Automatic Macro variable syslibrc - good and bad

The most general SAS automatic macro variable is probably **&syserr**. It always contains the completion code for the most recent DATA or PROC step. It is set to 0 if the step was successful or another number if the step failed or issued a warning. For instance, in the following example the DATA step tries to create a data set from raw data. It fails because the raw data file does not exist.

```
/*Cant create data set. Check syserr*/
DATA permlib.dailyfile;
  INFILE norawdata;
  INPUT trans_id $10.
         trans_amt 8.2;
  IF trans_amt > .
    THEN OUTPUT;
RUN;

%PUT syserr=&syserr;
```

Fig.3 Create data set and print. Fail, check syserr.

```
ERROR: Physical file does not exist,
C:\WP\Docs\sastuff\NORAWDATA.
NOTE: The SAS System stopped processing this step because
of errors.
WARNING: The data set PERMLIB.DAILYFILE may be incomplete.
When this step was stopped there were 0 observations and 2
variables.
WARNING: Data set PERMLIB.DAILYFILE was not replaced
because this step was stopped.
NOTE: DATA statement used:
      real time          0.06 seconds

192 %PUT syserr=&syserr;
syserr=1012
```

Fig.4 Log after failed create data set.

To use the `syserr` macro variable effectively, wrap the 2-step program from Figure 1 in a Macro, as in Figure 5. After each step, macro `runreport` checks `syserr` for its return code. If there is a non-zero status, an error message is written to the log and the next step is not run. See Figure 6 for the log produced by macro `runreport` when the input flat file, `norawdata`, does not exist.

```
%MACRO runreport;
DATA permlib.dailyfile;
  INFILE norawdata;
  INPUT trans_id$10.
         trans_amt 8.2;
  IF trans_amt > .
  THEN OUTPUT;
RUN;
%IF &syserr /* NE 0*/ %THEN
  %PUT %QUOTE(>>>ERROR: RC=&syserr
permlib.dailyfile could NOT be created.);
%ELSE %DO; /*syserr=0*/
  PROC PRINT DATA=permlib.dailyfile;
    SUM trans_amt;
  RUN;

  %IF &syserr /* NE 0*/ %THEN
    %PUT %QUOTE(>>>ERROR: RC=&syserr
Report NOT produced.);
%END; /***end report*/
```

Fig.5 Macro to create data set and print. Stop if error.

```
ERROR: Physical file does not exist,
C:\WP\Docs\sastuff\NORAWDATA.
NOTE: The SAS System stopped processing this step because
of errors.
WARNING: The data set PERMLIB.DAILYFILE may be incomplete.
When this step was stopped there were 0 observations and 2
variables.
WARNING: Data set PERMLIB.DAILYFILE was not replaced
because this step was stopped.
NOTE: DATA statement used:
      real time      0.04 seconds
>>>ERROR: RC=1012 permlib.dailyfile could NOT be created.
```

Fig.6 Log from Macro using `syserr`

## Functions

There are several SAS functions that report the success of processes. For instance, `SYSMSG()` returns an error message when a SAS function that involves the operating system fails. `SYSRC()` yields a return code for the same functions. The code is 0 when the function is successful and another number when the function fails or ends with a warning.

Figures 7 and 8 show one way to check whether a physical file exists, using functions `SYSMSG()`, `SYSRC()`, and `FEXIST`. This example does not involve macros. The DATA step uses the `FEXIST` function to determine whether a fileref was previously defined and whether the physical file to which it refers exists. If `FEXIST` returns a non-zero value (all was not well), `SYSMSG()` and `SYSRC()` determine what the exact problem was. A full-size program might respond to the results of those functions by creating a new data file if it didn't exist or by executing a `FILENAME` function or statement if the fileref (logical file name) was just not assigned.

```
DATA _NULL_;
  filehere = FEXIST('inrawdat');
  IF NOT filehere THEN DO;
    message = SYSMSG();
    rc = SYSRC();
    PUT '>>>Raw data file problem:'
      / rc= message=;
  STOP;
END;
PUT '>>>File was found.';
- more code ---
RUN;
```

Fig.7 Functions `SYSMSG()` and `SYSRC()` in use.

```
-- 1 --
>>>File was found.
-- 2 --
>>>Raw data file problems:
rc=-20006 message=WARNING: Physical file does not exist,
C:\wp\docs\sesug02\nopaper\norawdata.
-- 3 --
>>>Raw data file problems:
rc=-20004 message=WARNING: No logical assign for filename
RAWDATA.
```

Fig. 8 Some possible Log messages

## SAS Language Elements Used To Work With Error Conditions

Here is a partial list of SAS Elements that are useful in controlling program execution within and between steps.

### Automatic macro variables:

- sqlrc** Set at end of each SQL statement e.g. after each `SELECT` or `CREATE TABLE`; Remains accessible in non-SQL steps until next SQL statement is run; Successful=0.
- syserr** Set at the end of every DATA or PROC step; For multi-part PROCs such as `DATASETS`, the code is set only after the `QUIT`; Successful=0.
- sysfilrc** Set after every `FILENAME` statement or function; Successful=0.
- syslibrc** Set after every `LIBNAME` statement or function; Successful=0.
- sysmsg** Contains text to display in the message area of a macro window; `SYSMSG()` function is probably more useful.
- sysrc** Contains the last return code generated by your operating system for commands you execute using X, `%SYSEXEC`, `%TSO`, etc.; Does not work under all versions of Windows; I haven't tried it on other platforms; Test before using; Successful=0.

### Functions:

**SYSMSG()** returns the text of error or warning messages from SAS functions that perform an operating system task (e.g. `EXIST`, `FILEEXIST`);

can be reset to blank by running CALL SYSTEM() or X; NOT the same as &sysmsg.

**SYSRC()** returns a system error number from SAS/OS functions (e.g.FEXIST, LIBNAME); not reset until next SAS/OS function; similar to SYSMSG function, but it yields a code number instead of a message; also its value may be reset sooner; NOT the same as &sysrc.

WARNING: Test these functions in your program before you depend on them. I found that the rules for their use were not clearly explained in the documentation. They also vary with the operating system.

**Other:**

**sysrc.sas** is a program included with the SAS System. It contains a macro with a partial list of numeric SAS return codes and their meanings. In the Windows environment it is located in core\sasmacro in the SAS root directory. This list can be useful in interpreting the values in &syserr and other automatic macro variables.

**Conclusion:**

SAS programs can be built to be self-checking. In a production data processing environment, use built-in SAS features to make your systems more robust, by having them take corrective or defensive action when necessary.

**References:**

Macro Variables Set By SAS SQL Statements in SAS System Help

Macro Dictionary in SAS System Help

SAS Functions: Index in SAS System Help

SAS *OnlineDoc*®, Version 8, SAS Institute Inc., 2000. see *Base SAS Software, SAS Language Reference: Dictionary, Dictionary of Language Elements, Functions and CALL Routines* and *SAS Macro Language Reference*.

SAS® *Macro Language: Reference, First Edition*, Cary, NC: SAS institute Inc., 1997.

**Contact:**

Heidi Markovitz - Simply Systems - 161 Crandon Blvd. #325 - Key Biscayne, FL 33149 - [HeidiM@Simply-Systems.com](mailto:HeidiM@Simply-Systems.com) - 305-365-0439

SAS® and all other SAS Institute Inc product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.