

# Graham's Guide to Object/SAS Programming Diagnostics Error, Normal Execute, QA

Kevin Graham, Montura, USA

## SUCCESS METRICS

Success occurs when QA/QC teams and Managers can diagnose the root cause of an issue without knowing the solution is coded in SAS. Architects and Programmers design every source code to provide the following information elements, either in the SAS log or an external file.

- Name and physical location of every SAS source code.
- Positive and negative results following the execution of every SAS source code.
- Pass/Fail indicators consisting of return codes and row counts following the execution of every data step and SAS procedure.
- A listing of every global variable (or parameter) that is used within each SAS source code, provided after the code executes, available even when the app crashes.
- A clear description of the process and the data evaluated when a condition is evaluated and the result is a decision is to execute or bypass a block of code.

### Easy and Non-Technical

Switching to Object SAS makes life easy because everything that needs to be known about SAS apps can be answered with three simple questions.

## (1) WHERE'S MY STUFF?

### Objective

Have SAS applications coded to self-identify every source code that is required for normal execution. Standard diagnostics should be programmed into the controller by the Architect. Error-detect routines in the controller should provide plain-English error messages if source code is missing or is restricted due to user-permissions.

### Initial Development Cycle

Ask the SAS Architect to create a global variable that contains the following information.

- Path information for every Libname and Filename
- Name of every SAS program and SAS catalog that is referenced within in the app, without regard to conditional execution.
- Exclude information that is used only by the application controller because that is system-level information created by the SAS Architect that is not made accessible to SAS programmers.

### QA and Production Use

Ask the SAS Architect to provide a file that contains the following information in a file that is separate from the SAS log. Make sure the file has a unique name each time the app is executed.

- Every SAS source code, format, and catalog that is required for normal program execution must be flagged as present or absent by the application controller, without crashing to application.
- A listing that shows the sequence of execution and physical location of every SAS source code that is referenced by the app.
- A listing that shows relevant information when non-SAS programs are executed
- A listing that shows connection information when a non-SAS database is used.
- Data element that indicates programs that are not part of the standard package and were conditionally added into the program worklist.

### Implementation

Ensure major problems are detected BEFORE the apps begin normal execution. Ask the architect to verify each SAS source code is present and ready for normal execution.

## Required information

- Name, description, and purpose present so normal humans can understand.
- Entry point or invocation.
- Execution sequence.
- Physical location of all required source code
- Physical location of all conditionally executed source code.
- Name of every SAS catalog that contains app source code (exclude system catalogs – SASHELP, MAPS, etc.).

## Example

Application Name: HOTEL-RATE-FORECAST

Catalog: MONTURA.RATEPOPULATE.ENTRY.SCL

```
Sequence: 001 Program: MONTURA.RATE.LOGMESSAGES.CLASS
Sequence: 002 Program: MONTURA.COMMON.REQUESTHEADER.CLASS
Sequence: 003 Program: MONTURA.COMMON.ODBCTENANT.CLASS
Sequence: 004 Program: MONTURA.RATE.REQUESTDETAILSEXTRA.CLASS
Sequence: 005 Program: MONTURA.RATE.SOURCEXMLLIB.CLASS
Sequence: 006 Program: MONTURA.RATE.XMLDATA1.CLASS
Sequence: 007 Program: MONTURA.RATE.XMLDATA2.CLASS
Sequence: 008 Program: MONTURA.RATE.VALIDPROPERTY.CLASS
Sequence: 009 Program: MONTURA.RATE.RECORDTYPE.CLASS
Sequence: 010 Program: MONTURA.RATE.ACCOMTYPE.CLASS
Sequence: 011 Program: MONTURA.RATE.EXTRACTPREPARED.CLASS
Sequence: 012 Program: MONTURA.RATE.EXTRACTDATES.CLASS
Sequence: 013 Program: MONTURA.RATE.SYSTEMCOMPARE.CLASS
Sequence: 014 Program: MONTURA.RATE.LOADSTATUS.CLASS
Sequence: 015 Program: MONTURA.RATE.RATEMATCH.CLASS
Sequence: 016 Program: MONTURA.RATE.RATEUNQUALIFIED.CLASS
Sequence: 017 Program: MONTURA.RATE.RATEQUALIFIED.CLASS
Sequence: 018 Program: MONTURA.RATE.RATEUNQUALIFIEDDETAILS.CLASS
Sequence: 019 Program: MONTURA.RATE.RATEQUALIFIEDDETAILS.CLASS
Sequence: 020 Program: MONTURA.RATE.REVENUEVALUE.CLASS
Sequence: 021 Program: MONTURA.RATE.MARKETSEGMENT.CLASS
Sequence: 022 Program: MONTURA.RATE.ACCOMODATION.CLASS
Sequence: 023 Program: MONTURA.RATE.EXECUTESTATUS.CLASS
```

## Code Location(s)

LIBNAME MONTURA: C:\DEVELOPMENT\REVENUEMANAGEMENT

LIBNAME XMLDATA: C:\DEVELOPMENT\EXTRACT

FILENAME XMLMAP: C:\DEVELOPMENT\XMLMAP

## SAS Catalogs

Montura.Rate

Montura.Common

## (2) WHAT HAPPENED?

Ask SAS Architects to create external log information that indicates when each source code begins and ends execution, even when the app crashes.

Example – a minimum of four lines are logged for each SAS source code.

1. Blank line
2. @@+ source-code-name

3. @@- source code-name
4. Blank line

Ask the SAS Programmer create external log information after executing each block of code. Result information must be present, even when the app crashes.

**Positive Information**

- Information that indicates success.
  - Row counts
  - SQL return codes.
- Log information that identifies the name and location of output that will exist following a successful run.
- Information that can be used by QA to eliminate the possibility of a false positive. SAS logs normally contain row counts, notes, and warning by default. SAS apps can be programmed to explicitly indicate the expected result for non-technical analysis.

**Negative Information**

- Information that indicates the reason for failure.
  - SQL return codes
  - Dataset row counts
  - Foreign RDBMS message
  - Operating system message
- Catalog, program, and method where the issue occurred.
- Name of absent global variable, dataset, folder, etc.
- Name and values of a global variable when an unexpected or invalid value is detected.

**Behavior Information**

- Execute status when a source code is executed conditionally.
  - Data values or condition that is evaluated.
  - Message to indicate status will be enabled or disabled.

Ask the SAS Architect to create external log information that describes standard and optional components within the SAS application. Capture information that indicates when a source code is executed conditionally, as well as the condition that is evaluated.

**Content Information**

- Message to indicate when two types of source code are detected
  - Required.
  - Optional (i.e. conditional business rules).
- Information that indicates the condition or data evaluated when optional code is detected.
  - Message indicating optional code will be used or excluded.
  - Data or condition evaluated.

**Summary Information**

Ask Architects to itemize execution results. Collect and summarize positive, negative, and conditional-execute information from each source code and report the totals, even when the app crashes.

```
*****
APPLICATION MESSAGES

23: NORMAL COMPLETION
00: ERRORS
00: WARNINGS
00: CONDITIONALLLY EXECUTED PROGRAMS

*****
APPLICATION STATUS: SUCCESS
```

## Example

```
@@+ MONTURA.RATE.XMLDATA1.CLASS
proc sql;
  create table temp as
  select *
  from sashelp.vtable;
quit;
NOTE: Table WORK.TEMP created, with 490 rows and 41 columns.

13 quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time          8.62 seconds
      cpu time           2.77 seconds

@@ Global Variables
      (SQLOBS=490 ERRORCOUNT=0 SQLRC=0)[14]

@@- MONTURA.RATE.XMLDATA1.CLASS
```

## Source Code Formatting

Ask SAS programmers to apply the SCL directive CONTROL ASIS just before submitting code for execution. This tells The SAS System to present code in the Log exactly as it was coded – picture perfect with indentations, spaces, and tabs.

Encourage SAS programmers to write readable code, or else. The term “readable” means source code can be copy-and-paste directly into a SAS manual for instant use by other humans. SCL editors provide support for picture-perfect code so there is never an excuse for “ugly code”. Ugly code signals coder disorganization.

## (3) WHERE ARE THE DETAILS?

### Objective

Every time the SAS app is run the controller program should create a new and unique file that contains all of the information that answers “What Happened” and “Where’s My Stuff”. Ask to architect to mark a CLEAR difference between validations and errors.

- Data validations signal clean or dirty data.
- Anything else is probably an error.

### Example: Error Details

```
(ERROR) PROGRAM=MONTURA.RATE.XMLDATA2.CLASS, METHOD=INTERFACE3
(CATEGORY) Configuration Error
(CODE) M233
(RESOLVE) REQUEST.MAP was not found, it must be located in: C:\development\new\testcase101
```

## INTELLECTUAL PROPERTY INFORMATION

COPYRIGHT © 1989 - 2011 Montura, Inc. All rights reserved. This material may not be published, broadcast, rewritten or redistributed. All material in this paper is drawn directly from US Patent Repository Relationship Programming 7,984,422. Reading any page in this paper is the same as reading patent US 7,984,422.

## **SAS SOLUTIONS – CRYPTIC, DISORGANIZED, ERROR-PRONE**

Base/SAS applications are usually a spaghetti code mess.

### **Problems Start**

- Printed flowcharts are rarely in synch with production code.
- Documentation is incomplete because many details were hashed out through email.
- The original written specification and current application results are different because so many details were modified through verbal agreements over a period of time.

### **SAS Solutions Get Complicated**

- Computer science people tend to write code that is focused on CPU and time efficiency, with an accounting for potential errors due to gaps in the specs.
- MBA grads stop coding when results look right, based on eyeball analysis.

### **SAS Solutions Get Messy**

- Tracing through code to identify “how the app works” can be a complete nightmare. Most SAS programs have to be executed and the SAS Log examined before tracing can be performed.
- Code for a single app may be stored in multiple SAS catalogs. Any source code may conditionally include additional source code. Each SAS catalog may contain hundreds of SAS programs, macros, compiled macros, object oriented code, formats, and GUI screens.

### **Bottom Line**

- %include statements are the root of all evil, when it comes to 24/7 production support.
- Error-trapping in Base/SAS is non-existent.