

Debugging Standard Macros

Anthony Cooper, GlaxoSmithKline, Harlow, UK

ABSTRACT

HARP (Harmonisation of Analysis & Reporting Program) is GlaxoSmithKline's in-house developed global standard reporting system. The two main components of HARP are the web-based HARP Application and the suite of standard SAS macros, the HARP Reporting Tools. Recent reviews of both components highlighted the need for enhancements to help users when debugging the macros.

The purpose of this paper is to describe some of the design principles of the HARP Application and Reporting Tools, to discuss the feedback from the reviews, and finally to cover the planned changes. It will also touch upon some of the debugging functionality available in SAS.

INTRODUCTION

Several very good papers have been written which discuss SAS debugging techniques and tools in detail: [1]. [2]. [3]. In this paper, we will look at the subject of debugging from the point of view of the HARP Application and Reporting Tools, which are used within GlaxoSmithKline to support the Analysis & Reporting process for Clinical Trials and Submissions.

First we will clarify what we mean by debugging and the types of error that can occur.

WHAT IS DEBUGGING?

A good definition of debugging is *the process of locating and correcting errors (bugs) in a computer program*.

In very general terms, two types of error may occur:

1. Those where the program executes with visible SAS error, warning or note messages that give you some clue where the problem is
2. Those where the program executes without any apparent problem, but does not produce the results you expect (e.g. percentages are calculated incorrectly)

The definition states that debugging is a process, i.e. there a set of steps that's should be followed to solve the problem:

1. Recognise that a problem exists
2. Isolate the source of the problem
3. Identify the cause of the problem
4. Determine a solution
5. Apply/test the solution

Debugging is a programming skill. It is about knowing what tools are available and when to apply them. It is a mixture of logic (following a process and applying tools) and instinct (asking "Have I seen this problem before?" or "What if ?") and is a skill that is learnt through practice.

WHAT TYPES OF ERROR CAN OCCUR?

SAS errors can be classified into 4 categories [2], [3]:

1. Syntax errors - These are errors that SAS detects when compiling code and are caused by violating the rules of the SAS system. Examples include misspelled keywords, missing semi-colons, unmatched quotes,

PhUSE 2007

invalid variable names and resource errors (e.g. a dataset does not exist). In these cases, SAS will write an ERROR or WARNING message to the log.

2. Execution or run-time errors - These are errors that occur when SAS executes syntactically correct code and are generally caused by incorrect programming. Examples include attempting BY processing when the data has not been sorted correctly, uninitialised variables, missing values and numeric to character conversions. Depending on the severity, SAS may write an ERROR or NOTE to the log and continue executing.
3. Data errors - These are errors that occur when SAS reads in or processes data and are caused by contradictions between the code and the actual data. Examples include attempting to read in data with an inappropriate informat, character field truncation and invalid function arguments. When data errors occur, SAS will write a NOTE message to the log.
4. Logic “errors” – These are errors that occur when the code runs without error but undesirable results are produced. Examples include forgetting to use an output statement when summarising a dataset using BY processing (i.e. output dataset contains too many observations) and incorrect results from a merge. Typically SAS will not write any ERROR, WARNING or NOTE messages to the log in these situations. Therefore, these are usually the hardest type of errors to detect and may require detailed examination of intermediate datasets etc.

HARP DESIGN PRINCIPLES

To put the feedback and planned enhancements into context, it is important to understand some of the design principles of the HARP Application and Reporting Tools. Further information on the background, design and implementation of HARP can be found in “Design Principles & Implementation of a Reporting System” [4].

HARP APPLICATION

The HARP Application is a web-based system which allows users to plan and create their Analysis & Reporting (A&R) datasets and data displays (tables, figures and listings). It is a front-end to the HARP Reporting Tools, the suite of standard SAS macros.

The HARP environment has production and development UNIX areas which are a mirror image of each other in terms of the directory structure. However, users can only write to the production area via the HARP application to ensure a full audit trail and 21 CFR Part 11 compliance.

If a standard Reporting Tool can be used to create the required analysis ready dataset (i.e. variables collected on the Case Report Form plus derived variables) or data display, the user can select the macro via the HARP application, edit the macro parameters (if required) before running the macro to produce the dataset or data display.

If a standard Reporting Tool is not available, users can develop and test their own non-standard macros within the development UNIX area before importing and running it into the production area via the HARP Application.

When the users submit A&R dataset and data display creation jobs via the HARP Application, they receive notification messages indicating when the jobs have been submitted and completed and whether they completed successfully.

If jobs completed with SAS errors or warnings, the user opens the log file in UNIX to view the details. If the messages are not sufficient to solve the problem, the next step is usually to copy the code to the development area so that it can be run with SAS macro debugging options switched on or run interactively so that intermediate datasets can be viewed.

HARP REPORTING TOOLS

The HARP Reporting Tools are a cohesive collection of in-house developed SAS macros, which integrate with the HARP Application to create A&R datasets and data displays. The macros can either be used “off the shelf” to produce purely standard datasets and data displays, or as part of the user’s own non-standard macro to reduce development effort and ensure consistency with standard outputs.

For each A&R dataset and data display, a SAS “driver” program is created which calls a “setup” macro followed by the macro which produces the output. The main purpose of the “setup” macro is to define libnames and global macro variables which are used by the Reporting Tools.

PhUSE 2007

The HARP Reporting Tools follow a “building block” design:

- Utility macros - Code blocks that are designed to perform one job and do it well. For example to check whether a variable exists in a dataset, or to merge common variables (age, sex, race etc.) onto a dataset.
- Package macros - A macro that contains a series of calls to utilities macros to perform a specific task. For example to create data displays of frequency counts or summary statistics.
- Wrapper macros – Contain little or no coding. For data displays, they pass default macro parameter values into the appropriate package macro to create a specific display (e.g. a standard adverse event table). For A&R datasets they call a series of utility macro to create a dataset with the appropriate standard variables.

Each Reporting Tool was developed in-line with a set of defined programming standards to maintain quality and to ensure consistency. When each macro is called, it writes the macro name, macro version and a list of any global and local macro variables used to the SAS log.

Extensive parameter validation is performed in all utility and package macros, for example to check if a required dataset exists and is not empty, and to check that all required parameters have valid values. If the macros detect any errors, meaningful messages are written to the log and the macro is sensibly aborted.

Finally, at the end of its execution each macro calls a “tidyup” macro which deletes any temporary SAS datasets and global macro variables created by the macro.

REVIEW FEEDBACK

The purpose of the HARP reviews was to determine user satisfaction with the HARP Application and Reporting Tools and to gather feedback on aspects that could be improved.

Specific comments related to debugging were:

1. Logs can be long and difficult to understand
2. Nested macros can make it difficult to track down the problem
3. Inappropriate messages may be generated by the macros
4. Debugging method is not easy
5. Not possible to view the SAS log via the HARP Application
6. It is difficult to run the macros in interactive SAS
7. Need more documentation on debugging

Comments 1 and 2 are related to the design of the Reporting Tools. Since every macro called echoes information to the SAS log, the log can be large in size and may mainly contain information about the low level utility macros, which is not of particular interest to the user. The nesting of macros can also make it hard for the user to trace the cause of the problem. They may know which low level utility produces the error, but cannot easily determine which higher level macros have called it and whereabouts in the higher level macro's execution it occurs.

As well as performing extensive validation, the macros also perform a few basic data quality checks, e.g. when merging on treatment variables a warning is written to the log if there are subjects in the data which have not been randomised. However, one warning is issued per observation per subject, which can result in a lot of warning when for example creating lab datasets. Other situations where inappropriate messages (comment 3) were being produced include where code was left over from development.

Comment 4 refers to the need to copy code from the production to the development UNIX area in order to perform debugging before adding in SAS macro debug options etc. to determine the cause of the problem. Not being able to view the log through the HARP application can increase the time taken to find simple errors such as invalid parameter values (comment 5). Due to the design of the Reporting Tools, it can also be difficult to step through the code interactively (comment 6).

Finally, the users wanted additional information on the tools available for debugging the HARP Reporting Tools, how best to go about the debugging process in the HARP environment and details of common problems (comment 7).

PLANNED ENHANCEMENTS

As a result of the feedback collected, a number of enhancements are planned to both the HARP Application and the Reporting Tools.

In addition, documentation, with tips and examples, and training will be provided to help the users debug their code with less help from support staff. This should also increase their understanding of how the Reporting Tools work and reduce any black-box mentality towards the macros that may exist.

HARP APPLICATION

Three changes are planned to the HARP Application:

- Option to view SAS logs.
- Option to run datasets/data displays with SAS macro debugging options switched on.
- Option to export 'debug ready' code from HARP to the development area.

Being able to view the log via the Application will save the user having to go into UNIX to perform this task. This along with being able run drivers with standard SAS macro debugging options should save users time when they encounter relatively straightforward errors. The third change will eliminate the manual steps of copying code into the development area and changing references to the production to the development area in SAS libnames etc.

HARP REPORTING TOOLS

The key change to the Reporting Tools is the addition of a parameter to the "Setup" macro to define the level of debugging required while running a dataset or data display "driver" program. Each value of the "debug" parameter will provide a different amount of debugging functionality:

- 0 (default) – Suppress non-essential messages/information being written to the log.
- 1 – Write all messages/information to the log.
- 2 to 4 – Switch on SAS macro debugging options (MPRINT/MPRINTNEST, MLOGIC/MLOGICNEST and SYMBOLGEN).
- 5 – Override "tidyup" macro (i.e. do not delete temporary datasets and global macro variables) and output PROC PRINT of datasets at key stages of macro execution.
- 9 – Output macro generated code to a file using MFILE option.

The debugging levels are additive in nature, i.e. level 2 includes the functionality of level 1, level 5 includes the functionality of levels 1 to 4 etc.

Preventing non-essential messages and information being written to the log by default, will significantly reduce the size of the logs, thus making the execution of the key macros more transparent to the user.

MPRINT, MLOGIC and SYMBOLGEN are used to output information on SAS statements generated, trace execution and macro variable resolution to the log. MPRINTNEST and MLOGICNEST are new parameters in SAS v9 which display macro nesting information. For example, MPRINTNEST will produce output similar to this:

```
MPRINT(REPORT.LABELS.TIDYUP):  proc datasets memtype=(data view) nolist;
MPRINT(REPORT.LABELS.TIDYUP):  delete _labels;;
MPRINT(REPORT.LABELS.TIDYUP):  run;
```

The user may know in which nested macro the problem occurs in, but the use of these new options will make it clear which macro is calling it.

Preventing the "tidyup" macro from deleting temporary datasets will help users to examine intermediate datasets, particularly when running the macros in interactive SAS. Similarly, printing the content of the SAS datasets at key points in the macro execution (e.g. before and after transposing the data) can help to narrow down where problems first occur. Both of these may highlight issues with the datasets being passed to the macros.

PhUSE 2007

Finally, the MFILE option will re-direct the SAS statements written out by MPRINT to a file as well as the SAS log. The syntax is as follows:

```
options mfile mprint;  
filename mprint '/home/userid/code/myreport_debug.sas';
```

This will allow the statements generated during macro execution to be run in interactive SAS, i.e. the user can step through the code.

CONCLUSION

Debugging can be a difficult and time-consuming process depending on the type of error occurring. This is particular true in situations where code runs without any SAS error, warning or note messages, but incorrect results are produced.

Successful debugging requires the user to be familiar with the potential sources of error and to be able to select the appropriate tool from those available to help them identify the problem.

When developing standard reporting applications and SAS macros, thought should be given to how users will go about the process of debugging and whether they are given appropriate information by standard tools. To begin with, it usually takes people longer to debug code they have not written themselves.

Through a review of the HARP Application and Reporting Tools, we have identified a few small enhancements that, when implemented, should significantly help the users perform debugging in the HARP environment.

REFERENCES

- [1] Frank Dilorio, The SAS® Debugging Primer, Paper 54-26, SUGI 26
- [2] Roger Staum, To Err is Human; to Debug, Divine, Paper 64-27, SUGI 27
- [3] Peter Knapp, Debugging 101, Paper 257-29, SUGI 29
- [4] Anup Patel and Lee Seymour, Design Principles & Implementation of a Reporting System, Paper AS05, PhUSE 2005

RECOMMENDED READING

SAS Institute Inc., SAS® Macro Language: Usage and Reference, Version 8

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Anthony Cooper
GlaxoSmithKline
New Frontiers Science Park
Third Avenue
Harlow
CM19 5AW
Work Phone: +44 (0) 1279 646349
Fax: +44 (0) 1279 644430
Email: Anthony.J.Cooper@gsk.com
Web: www.biometrics.com

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.

Other brand and product names are trademarks of their respective companies.