PhUSE 2012

Paper PP18

Tricks for testing Standard macro Components in SAS Software

Lina Ulkjær Jørgensen, Novo Nordisk A/S, Bagsværd, Denmark

ABSTRACT

The technical process to update – **hot fixing** - a standard component is cumbersome and time consuming. **This poster** gives suggestions on how to make the hot fixing process smoother. It shows that you can work faster by saving outputs and logs from test cases, transform them into datasets and use SAS Software® `proc compare` to test.

It also adds a suggestion to use SAS parameters on execution in automatic validation and shows a branch testing method to secure that standard components are robust.

INTRODUCTION

Novo Nordisk Portfolios of standard components in production exceeds 130. They are used to produce derived data and TFL outputs.

The technical process to test standard components is:

- Make many test cases
- Every single *if*-, *case*-, *do while*- statement in a component must be hit by a test case to cover TRUE and FALSE statements
- User acceptance test – to get commitment from the end user
- God validation process on the SAS source

…and then test and test and test again…☺

Finally the program is verified as a standard program and migrated into the production environment.
Facing reality you could after a while experience **Errors** or the end users asking "Why is it not possible to…….!"

This poster offers suggestions on how to make the **hot fixing** process smoother.

SAVE TIME – SAS OUTPUT

If one imagines a test case and output from this test case as 2 pieces of a puzzle the essential in the hot fixing process is to reuse these pieces. If you hold the source pieces in your left hand and the output pieces in the right hand after an update of the source it must be possible to put the 2 pieces together again and solve the puzzle. You do this as a check between the old and the new output piece. These must not be changed.

This is very simple, but if a test program has 75 test cases, you will have 75 pieces of code and 75 pieces of output. In this case you do exactly the same, merge and check that the new 75 outputs of a hot fixing process are exactly the same as the old 75 outputs.

If a new test case is added, you only have to test this one test case manually.
Note that in some update situations, you can expect differences caused by the reason for the update.

To compare the old pieces of the puzzle with the new ones use SAS Software procedure: `proc compare`.

Let see how simple it is in SAS:

Save output from Version_v01, test case 1 permanently by writing this:

```
libname run_v01 "C:\users\xxx\run_v01";
   data run_v01.events_case1;
      set events_case1;
   run;
```

Save output from Version_v02, test case 1 the same way:

```
libname run_v02 "C:\users\xxx\run:v01";
   data run_v02.events_case1;
      set events_case1;
   run;
```

Use `proc compare` as this:

```
proc compare data=run_v01.events_case1
          compare =run_v02.events_case2
          noprint
          out=work.comp outnoequals;
run;
```

Note that output from `proc compare` is sent to a dataset with options outnoequals, only to write differences.

Dataset: work.comp must have no observations.

If you still imagine the 75 test cases, you must do like this for all of them. Use `proc append` to stack out datasets (work.com) and if you end up with a total dataset with no observations, the hot fixing is home safe.

Next time you find a small error or customers have requests for small changes, you can do the same and **save time**!

SAVE TIME - SAVE THE LOG

To make the update process even more secure here is another trick: Save the log

Execute test program version_v01 -> save log
Execute test program version_v02 -> save log

We want to compare these logs but they have to be manipulated up front:

SAS log contains a lot of timestamps and information about memsize and CPU used.
This information will always be unique when you execute a program.

To compare 2 logs the trick is:

1. Save log in a file -> use SAS procedure `Proc Printto`
2. Infile this log
3. Remove these text strings: real time, CPU time, options, macvars, workmems and datatime stamp.
4. Transform log into a dataset
5. Do this for both Version_01 and version_v02
6. Use `proc compare` as this:

```
proc compare data=run_v01.events_log1
          compare =run_v02.events_log2
          noprint
          out=work.comp_log outnoequals;
```

If dataset `work.comp_log` has no observations, then the hot fixing is home safe.

EKSTRA TRICKS

In my poster I mention 2 other tricks we use when we test and develop components.

BRANCHTESTING

When we develop new component we produce a lot of test cases, so many that we hit every single *if*-, *case*-, *do while*- statement in a program. These must be hit to cover both TRUE and FALSE statement. This makes the program robust.

By branch testing we know that **all lines** in a component can actually execute.

To help us in this seeking for robustness, we have a utility.

1. It reads all *if/then/else* statement in a SAS program and exports this information to Excel.
2. It writes the line number and part of the *if/then/else* source text to Excel.
3. It reads the log from execution, checks for TRUE execution and puts a check mark in Excel.

Unfortunately we have to find all FALSE executions manully.

By running this utility we can see if a line in the program is not hit by a test program and we can add more test cases. The source will become more and more robust as we go along.

AUTOMATIC VALIDATION

It sounds “dream-like” and is a tool for helping validation of standard programs.
When a standard program is executed, it often runs with a lot of parameters. These can be set up when you call the program or be global, local or automatic macros.

SAS Software keeps the information about macros in the dataset sashelp.vmacro.

To get this information create a small macro like this:

```
%macro save_parms_v01(
       pgmname =
);
       data work.input_parms;
          set sashelp.vmacro;
          pgmname=”%nrbquote(&pgmname)”;
       run;
%mend save_parms_v01;
```

and put a call to this macro in all standard components like this:

```
%save_parms_v01(pgmname=my_program_for_PhUSE);
```

You can, for example, drag information about all title statements used in all standard components and compare this with your programming plan. It is also possible to find and check if all standard components are named correctly, treatment pattern e.g. parameters as these: “new drug” “comparator”.

Try yourself, just put these lines in a standard macro and observe all the useful information.

CONCLUSION

When hot fixing standard components use these tricks:

Save output and save the log from old version and new version of the component and convert this information to datasets.

Use proc compare between datasets and your home safe.

Save time – Save output – Save the log.

ACKNOWLEDGMENTS

I would like to thank the standard development programming team at Novo Nordisk for input and for the dynamic and inspiring working environment. I also extend my thank you to the consultants we have had – from various companies.

CONTACT INFORMATION

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

Author Name	Lina Ulkjær Jørgensen
Company	Novo Nordisk A/S
Address	Vandtårnsvej 83
City / Postcode	DK / 2860 Soeborg
Work Phone:	4530792860
Fax:	
Email:	LiUJ@novonordisk.com
Web:	