Paper 371-2011

Regression Testing: the SAS Programmer's Friend

Phil Busby, Live Data Systems, Inc., Ewen, MI USA

ABSTRACT

Consultants are often assigned to fix systems that were developed ad hoc over many years. The code needs consolidation, error checking, improvements for ease of use, and lots of comments. Above all, the consultant must ensure that the resulting system produces correct results. This paper shows how a reporting system SAS® macro was made easy to use and easy to maintain by developing a regression test to ensure software quality. A regression test for a SAS/AF® software system is described, using a single global SAS macro variable to make the testing of an interactive system run without any user intervention.

INTRODUCTION

Regression testing means testing a change to a software system by comparing the "before" and "after" code output. The "before" code produces known results that are presumed to be correct: the "benchmark" output. After a change to the code is made, we run the test version of the code, and compare the test output to the benchmark output. If the results are identical (except for what the code change is supposed to change), we know that the code change did not break anything. If not, we examine the differences, and either revise the change to omit the differences, or accept them. If we accept the differences, then the test output becomes the new benchmark output.

Nobody wants incorrect results. But trying to test for every possible input value is usually impossible. The first time I was given a system to write where every possible input value could be tried was a date routine. I programmatically generated every possible date from the Gregorian calendar from 1582 to the year 9999 and verified the result of the function using that date as an argument. Aha! I slept well knowing that my software was proven 100% correct.

The SAS system itself faces the same problem: so many users with such varied applications that SAS Institute cannot possibly know in advance that a new release will continue to give correct results for all users. With millions of lines of code, a regression test is a must. The Institute has benchmarked all of the examples of how the software is documented to be used by running a set of programs on the earlier release. When those same programs are run on the new release, the outputs are compared, and if they match, then SAS Institute knows that what used to work, still works. With a command-driven, procedural system like SAS Software, all of those programs can be put into a single job, and the output comparisons automated. The QA manager sleeps well knowing that the new release passes the regression test.

In 1993, I was working on a real-time system (SAS/Trader™) at SAS Institute where the input is coming in from a satellite dish on the roof as a broadcast. The stream of stock and commodity prices, interlaced with news stories, is asynchronous, unpredictable. If your software is off doing something else, like printing, then you might lose input data. SAS/Trader™ had to detect this buffer-chopping condition and repair it on the fly. How to test this? One way is to capture the input feed for a while (the baseline) and play it back later (the regression test) to see if the resulting portfolios match. Not only must the data stream be replicated, but also the timing of its transmission must be the same. But a data-driven, real-time software system stood the SAS command-driven, procedural model on its head. In subsequent work as a contract programmer, I saw that the testing problem didn't go away. How can the programmer know that his code is correct? A regression test benchmark is needed. But usually it does not exist.

Seventeen years and as many coding gigs later, my manager hands me a 4400-line SAS macro, saying that it has become unmanageable; please improve it. The macro evolved over ten years. Each quarter it had been run, and the output saved. Aha! I knew what the output was supposed to be. My benchmark data sets were already generated. Or so I thought.

THE BATCH APPLICATION

Medical laboratories need to verify that their equipment and procedures are calibrated and accurate, so that the doctor's diagnoses are based on correct lab measurements. My client provides this service by sending out test kits and surveys. The lab runs the test kit samples just like they would for a patient. They report their results in the survey, and get back a report saying if their readings are accurate, and also showing how well they are doing compared to other labs using the same equipment and method.

My customer's customers want to know how well they are doing today, but also how well they have done over time, so a reporting system was developed to extract a data item of interest, called a "monitor," from all of the history data, and to produce a report for each customer. At first there were just six monitors, but in subsequent years, more were added. After several years, some monitors were no longer relevant, and were discontinued. To be reported in this system, a customer had to have at least eight quarters of non-missing data for the monitor requested. An observation with a missing value in the output data set is required when a customer has no record for any quarter between his first quarter and his last. All of the input data exists in separate folders by monitor, year, and quarter. The SAS macro I was to revise was called HDRSETS, and its function was to generate all of the report data sets for the specified monitor, year, and quarter.

THE IMPORTANCE OF BACKUPS

Every programmer seems to learn this lesson the hard way, sometimes more than once. I should have known better, but I just took a copy of the macro and made a few obvious improvements. Of course, with a piece of software this size, one gets uneasy after a short while. My coding changes seemed straightforward, but did I break anything? To find out, I wrote a driver and invoked the macro for the first year and quarter for which we had any data: 1999 quarter 1. But in 2001, unknown to me, the name of the input data sets changed. What was output now was input in 1999. So I had just overwritten my input for the first year and quarter. The original data was on my supervisor's C drive, and she was on vacation.

THE REGRESSION TEST IS BORN

Well I benefited from this blunder by proceeding much more carefully. There was output for a certain monitor, year, and quarter that seemed impossible for the existing macro to have produced. So I wrote another driver to invoke the macro for every valid monitor, year, and quarter. This time, I changed the output data set name from "hdr" to "tst" and wrote a PROC COMPARE step to verify that I was really getting the right output. Sure enough, over the years, the macro had been changed, and only some recently-generated output data sets matched my test output. What was originally produced could not be recreated. I could not help that! The benchmark my revised macro would be measured by would be the tst data sets. I generated them from the original code, changing only the path to the data and the name of the output data set.

```
/* test just mmail d for monitor and year. */
%macro testqtd(mon,yy) ;
   %let monnum = %substr(&mon,3) ;
   %hdrsets (&mon, &yy, d) ;
   proc compare data=&mon.&yy.d.q&monnum.tst&yy.d
                compare=&mon.&yy.d.q&monnum.new&yy.d ;
   run ;
   %let hdrsetsrc = &sysinfo ;
   %if &sysinfo >= 64 %then
         %put &mon.&yy.d.q&monnum.new&yy.d DIFFERENT. ;
   %else %put &mon.&yy.d.q&monnum.new&yy.d ALLOK. ;
%mend testqtd ;
```

Now the PROC COMPARE code needed to use the tst data sets and the result of any changes I made. So the revised macro was changed to write data sets named "new." If the "tst" data sets matched the "new" data sets, I could sleep peacefully. But they didn't! Looking closer, the original code was including some extraneous variables in the output for certain monitors and years. Clearly the macro *ought* to omit them. But I had to keep in mind that these differences from PROC COMPARE were OK.

Correct Input is not a given

Well, the boss came back and graciously restored my input data for 1999. But I realized that 1999 was not even a valid year to use as a parameter value, since two years of data are needed for a customer to be included in the output. So I added checks to make sure the macro's parameter values were valid, and if not, to return immediately with a nice Log message.

```
 /* Validate: mon between qt1-qt17, myear 01-98, mmail a,b,c,d. */
%let monnum = %substr(&mon,3) ;
%let ds=q%substr(&mon,3) ;
%let monpre = %substr(&mon,1,2) ;
%put monnum=&monnum ds=&ds monpre=&monpre maxmon=&maxmon ;
%if (%eval(&monnum) < 1) or (%eval(&monnum) > &maxmon) %then %do ;
   %put hdrsets: monnum=&monnum invalid mon: &mon ;
   %return ;
%end ;
%if "&monpre" ne "qt" %then %do ;
   %put hdrsets: invalid mon: &mon ;
   %return ;
%end ;
/* year 01 logically possible, but no output seen in originals. */
%if not (("02" le "&myear") and ("&myear" le "98")) %then %do ;
   %put hdrsets: invalid myear: &myear ;
   %return ;
%end ;
%if not (("a" le "&mmail") and ("&mmail" le "d")) %then %do ;
   %put hdrsets: invalid mmail: &mmail ;
   %return ;
%end ;
```

Surprise! New system requirements!

Software systems are rarely static; they need changing over time as business requirements change. That is why comments in the code and clear documentation are so important. Write copious comments saying what, why, and how your code functions, and the next programmer tasked with modifying your code will thank you. My boss said, "Please add a 4th parameter specifying how many years to look back for inclusion in the report. Limit the results to just that many years of data, to the nearest quarter." Now lots of new logic was needed, and nothing could be hard-coded anymore for any monitor or year.

Limiting the look back period to just a few years of data meant that there was no comparison with the original benchmark data sets possible. I had to invoke the macro for each possible look back number of years, and examine the output data sets manually. Once was enough! After those output data sets checked out, they became the benchmarks for that number of look back years.

So it went: optimize the macro by putting in loops where we had %Ifs for each year. Consolidate, simplify, clarify with comments, delete redundant code, and use macro variables rather than hard-coded constants. Avoid quoted strings as macro variable values. This went on for many days, since I had to work on this only when other projects were not demanding attention. Each time, I would reach a stopping point, and run the regression test. If something was broken, I knew it at once! I could go back and review what just got changed, and fix it! Incrementally the code got better. After each iteration, I was sure it still worked. Here is typical revised code:

```
/* 4. Create work.test data set. */
/* Assign qtrint sequence number depending on */
/*    values of &mon. and qtr.                */
data work.test ;
   set work.fintotals ;
   by capno ;

   /* Assign QTRINT as the sequence number of the quarter in the */
   /* possible span of quarters thru the lookback period. */
   /* Do not consider lookback years since qtrint is based */
   /* on starting year of data. */
   /* Assign pushqtrint depending on start year of data for monitor: */
   /* how many quarters to push the start integer forward in time, */
   /* for assigning quarter sequence numbers for sorting.  */
```

```
      /* Absolute integer value does not matter, just for sorting, */
      /* so lookback is not affected. But regression test must     */
      /* match originals! Since qtrint is in output data set,      */
      /* it must be as it was. */
      %if &startyy=99 %then %let pushqtrint=0 ;
      %else %let pushqtrint = %eval((&startyy + 1) * 4) ;
      %put pushqtrint=&pushqtrint ;

      if qtr="99-A" then qtrint=1 ;
      if qtr="99-B" then qtrint=2 ;
      if qtr="99-C" then qtrint=3 ;
      if qtr="99-D" then qtrint=4 ;
      %let nextqtr = 5 ;
      %do qtryy = 0 %to &myear ;
         %let tyy = &qtryy ;
         /* Prepend leading 0 to year value if needed. */
         %if %eval(&tyy) < 10 %then %let tyy = 0&tyy ;
         if qtr="&tyy.-A" then qtrint= &nextqtr-&pushqtrint ;
         if qtr="&tyy.-B" then qtrint=(&nextqtr-&pushqtrint)+1 ;
         if qtr="&tyy.-C" then qtrint=(&nextqtr-&pushqtrint)+2 ;
         if qtr="&tyy.-D" then qtrint=(&nextqtr-&pushqtrint)+3 ;
         %let nextqtr = %eval(&nextqtr + 4) ;
      %end ;
   run ;

   proc sort data=work.test ;
      by capno qtrint ;
   run ;
```

Gradually the underlying algorithm was distilled into just three parameters per monitor: What year did it start? What year did it end, if any? And what is the variable name of the data point of interest? Well, no need to put this in the code. I created a little control data set, so any changes to the system could be done in the control data set which the macro would read, and the macro code would not need to be changed at all. Well, not quite: the client wanted to keep a 2-digit year, so it all breaks in 2099. Call me back then.

```
data _null_ ;
   set qtctrl.qtcontrol(where=(mon="&mon")) ;
   put "QTCONTROL:" mon= pi= startyy= endyy= ;
   call symput("pi", trim(pi)) ; /* data point of interest */
   call symput("startyy", substr(put(startyy,4.),3)) ;
   if endyy = . then call symput("endyy", " ") ;
   else call symput("endyy", substr(put(endyy,4.),3)) ;
run ;
```

A REGRESSION TEST FOR A SAS/AF® SYSTEM

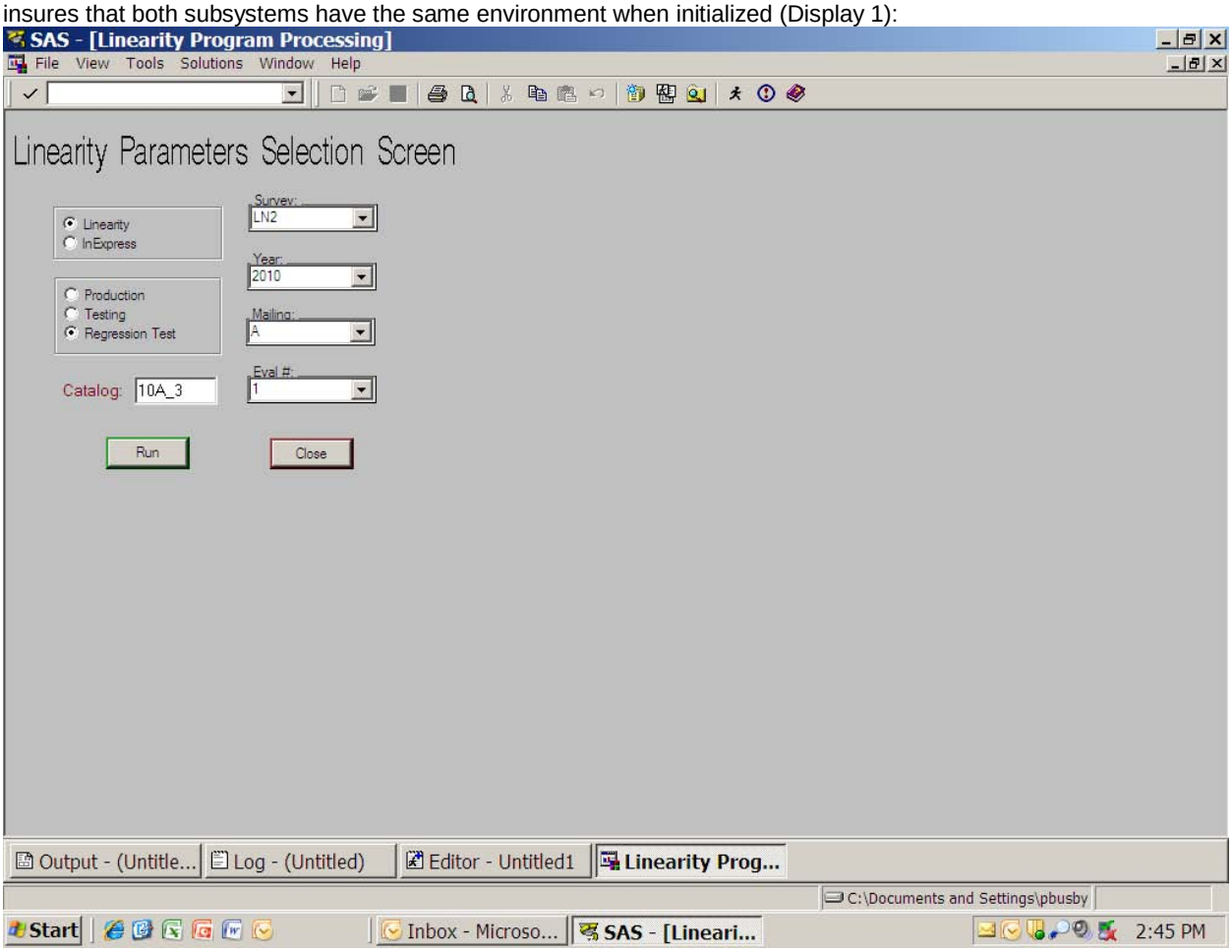
The client has a data management system using SAS/AF® Software to control execution of multiple SCL programs. To write a regression test for this system, all of the popup windows and PDF output must be suppressed to avoid having to repeatedly click "OK" and close the popups so the test execution can continue. There are several third-party screen capture products we could use, but this solution uses a single global SAS macro variable whose value of "regtest" is set in the regression test driver:

```
/* comment pfb 16aug2010: tell subprograms not to halt on popups. */
call symput('runmode','regtest');
```

The data management code copied for the regression test then queries the runmode macro variable value to suppress the popups, such as in this SCL module:

```
/* comment pfb 17aug2010: do not ask if runmode regtest */
tstr = symget('runmode');
put 'input.input_varlist runmode= ' tstr;
if tstr ne 'regtest' then do;
  rc=setitemc(msgList,'Do you wish to use the previous TRF settings?',1);
  callVar=messagebox(msgList,'?','YN',' ');
end;
else callVar='Yes';
```

Both the production system and the regression test are invoked from the same initial system entry screen. This insures that both subsystems have the same environment when initialized (Display 1):

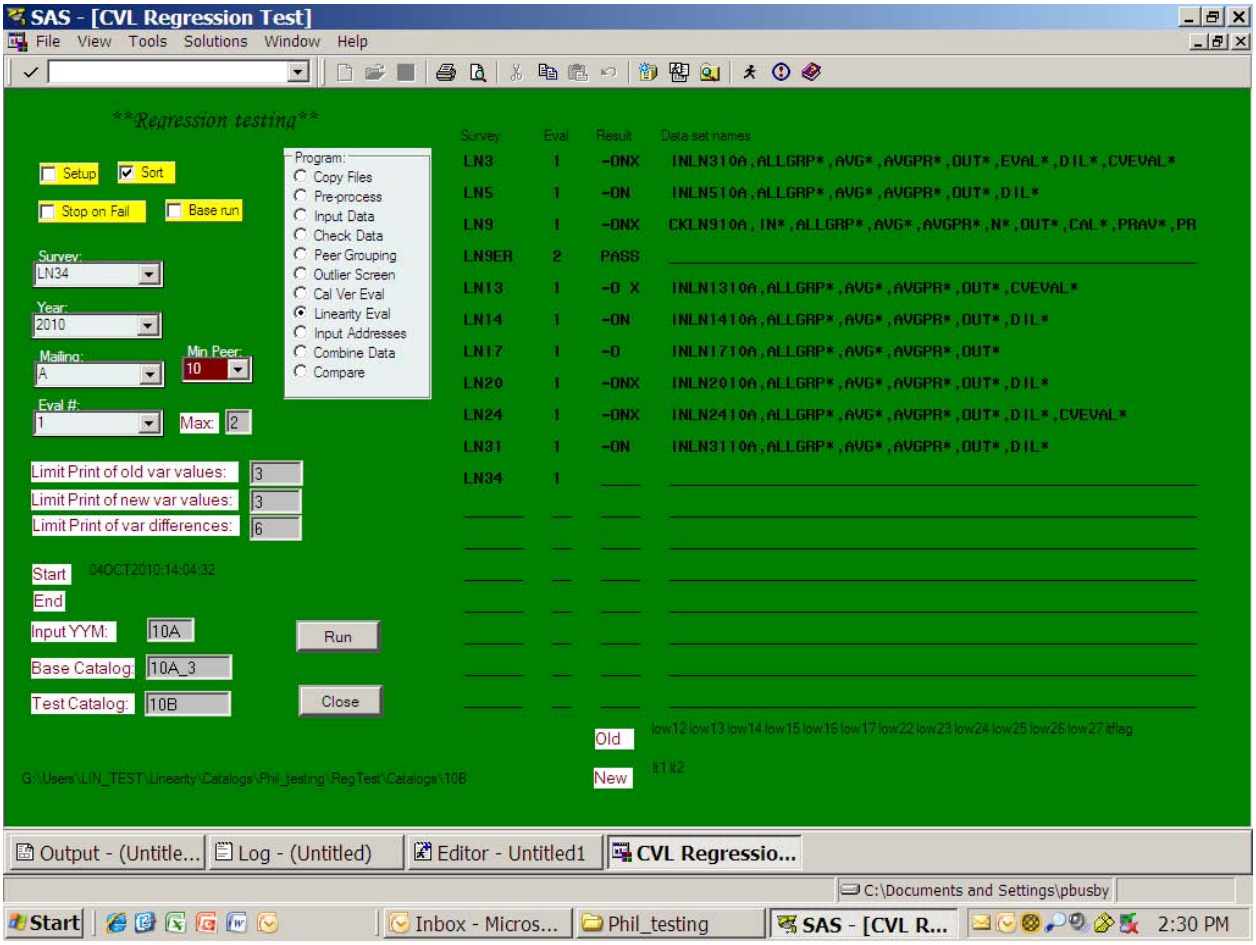


Display 1. Regression Test invocation using same environment as production system

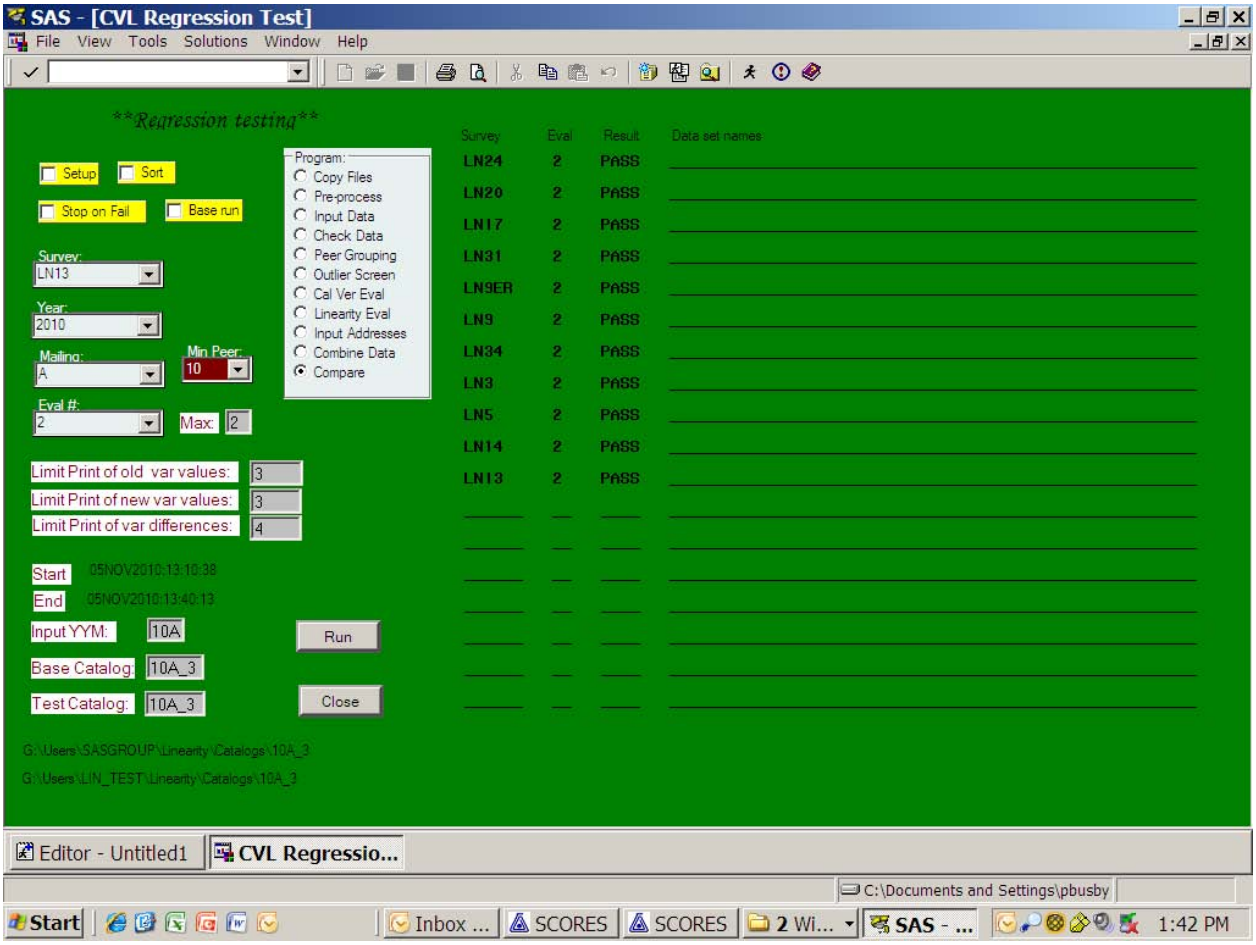
The SAS macro variable 'runmode' was later changed to a SCL list variable, but the functionality was the same. Note that this method of doing a regression test has the disadvantage of having to modify the system being tested. In this case it is important not to introduce any logic changes that depend on whether or not the regression test is running. The only thing we want to do is suppress the screen displays that would halt execution. Since the only thing we are depending on is a test for differences between benchmark and test output data sets, other output can be omitted during the regression test. But any code which could possibly alter a SAS data set must run the same way. The production system runs from a menu of processing steps. The operator selects a step and clicks Run. The regression test control screen simply runs *all* of the steps when the operator clicks Run. The screen is refreshed to show what step is currently executing.

The regression test tells the system administrator in less than an hour when a change made for one survey causes some differences in the output for any other critical surveys. In Display 1 below, the Result column shows up to four types of failures: Old variables are now gone (O), new variables found which did not exist in the benchmark (N),

variable value differences (X), and minimum observations not found (M). At the bottom of the screen we show old and new variable name differences for the data set being compared (Display 2):



Display 2. Original CVL Regression test Control Screen during execution



Display 3. Original CVL Regression test Control Screen after successful execution

Note the added features in Display 3 above: We can generate input data by copying production data for any year using the Setup checkbox; we can use any existing Base or Test catalog; we can execute the Base run once and use that benchmark for multiple Test Runs; we can sort the surveys or use the control data set order; and we show the start and end times for performance evaluation. The "Limit Print" fields control how much detail goes into the output report written at the end of the regression test. This summary (Output 1) is more useful than the copious output generated by all of the PROC COMPARE executions. The analyst ignores the value differences when there is an observation count difference for a test data set:

CVL regression test results at 06DEC2010:12:34:57 for pbusby
Data YYM: 10A minPeer:10 maxEval:2
Input folder: G:\Users\LIN_TEST\Linearity\regTest\Input
Base folder: G:\Users\LIN_TEST\Linearity\regTest\BaseOutput
Test folder: G:\Users\LIN_TEST\Linearity\regTest\CompareOutput
Base catalogs: G:\Users\SASGROUP\Linearity\Catalogs\10B
Test catalogs: G:\Users\LIN_TEST\Linearity\Catalogs\10B

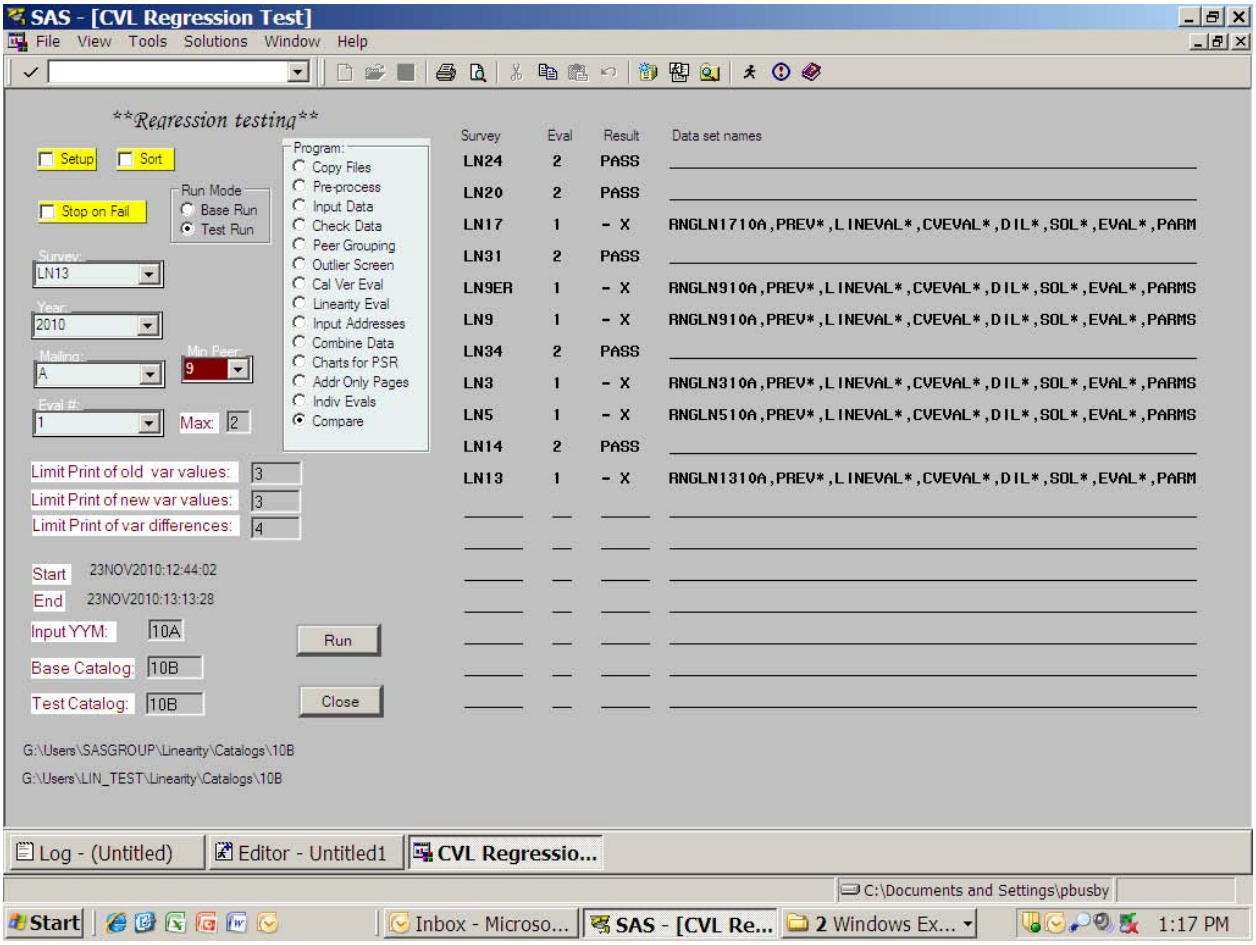
Obs	Survey	Eval Number	Data Set	passfail	Base Observations	Test Observations	Observation Count Difference	Variables on Base but not on Test	Variables on Test but not on Base
1	LN24	1	CKLN2410A	- X	0	4	4		
2	LN20	1	CKLN2010A	- X	2	14	12		
3	LN17	1	CKLN1710A	- X	20	38	18		
4	LN31	1	CKLN3110A	- X	2	10	8		
5	LN9ER	1	CKLN910A	- X	68	96	28		
6	LN9	1	CKLN910A	- X	47	82	35		
7	LN34	1		PASS					
8	LN34	2	CKLN3410A	- X	4	8	4		
9	LN3	1	CKLN310A	- X	56	90	34		
10	LN5	1	CKLN510A	- X	18	68	50		
11	LN14	1		PASS					
12	LN14	2		PASS					
13	LN13	1	CKLN1310A	- X	100	152	52		

Obs	Variables with value differences	Corresponding value differences	FAIL: Variables on Base but not on Test	FAIL: Variables on Test but not on Base	FAIL: Variable Values Differ	FAIL: Data set Incomplete
1	Pg, Seq, analyte, kitRef	(Base: Test:01)(Base: Test:0070)(Base: Test:Creatinine)(Base: Test:20009010101)			Yes	
2	Seq, kitRef, kit_id, newName	(Base:0080 Test:0060)(Base:70985010101 Test:19927010801)(Base:22800323 Test:22586812)(Base:DIL12 Test:DIL11)			Yes	
3	Seq, newName, solnLbl, value	(Base:0180 Test:0140)(Base:DIL15 Test:DIL11)(Base:LN1705 Test:LN1701)(Base: 350 Test: 29)			Yes	
4	Seq, kitRef, kit_id, newName	(Base:0040 Test:0030)(Base:32376010601 Test:13192010101)(Base:22682180 Test:22514709)(Base:DIL12 Test:DIL11)			Yes	
5	Pg, Seq, analyte, kitRef	(Base:02 Test:01)(Base:0100 Test:0040)(Base:Hemoglobin Test:Red Blood Cell Count)(Base:12488010301 Test:34097010101)			Yes	
6	Pg, Seq, analyte, kitRef	(Base:02 Test:01)(Base:0100 Test:0040)(Base:Hemoglobin Test:Red Blood Cell Count)(Base:12488010301 Test:34097010101)			Yes	
7		(Base:02 Test:01)(Base:0100 Test:0040)(Base:Hemoglobin Test:Red Blood Cell Count)(Base:12488010301 Test:34097010101)				
8	Seq, newName, solnLbl, value	(Base:0080 Test:0070)(Base:DIL14 Test:DIL13)(Base:LN3404 Test:LN3403)(Base: 273.8 Test: 150.8)			Yes	
9	Pg, Seq, analyte, newName	(Base:06 Test:04)(Base:0340 Test:0290)(Base:Theophylline Test:Phenobarbital)(Base:DIL14 Test:DIL11)			Yes	
10	Seq, kitRef, kit_id, newName	(Base:0260 Test:0250)(Base:11342010101 Test:12863020601)(Base:22767440 Test:22581016)(Base:DIL13 Test:DIL12)			Yes	
11		(Base:0260 Test:0250)(Base:11342010101 Test:12863020601)(Base:22767440 Test:22581016)(Base:DIL13 Test:DIL12)				

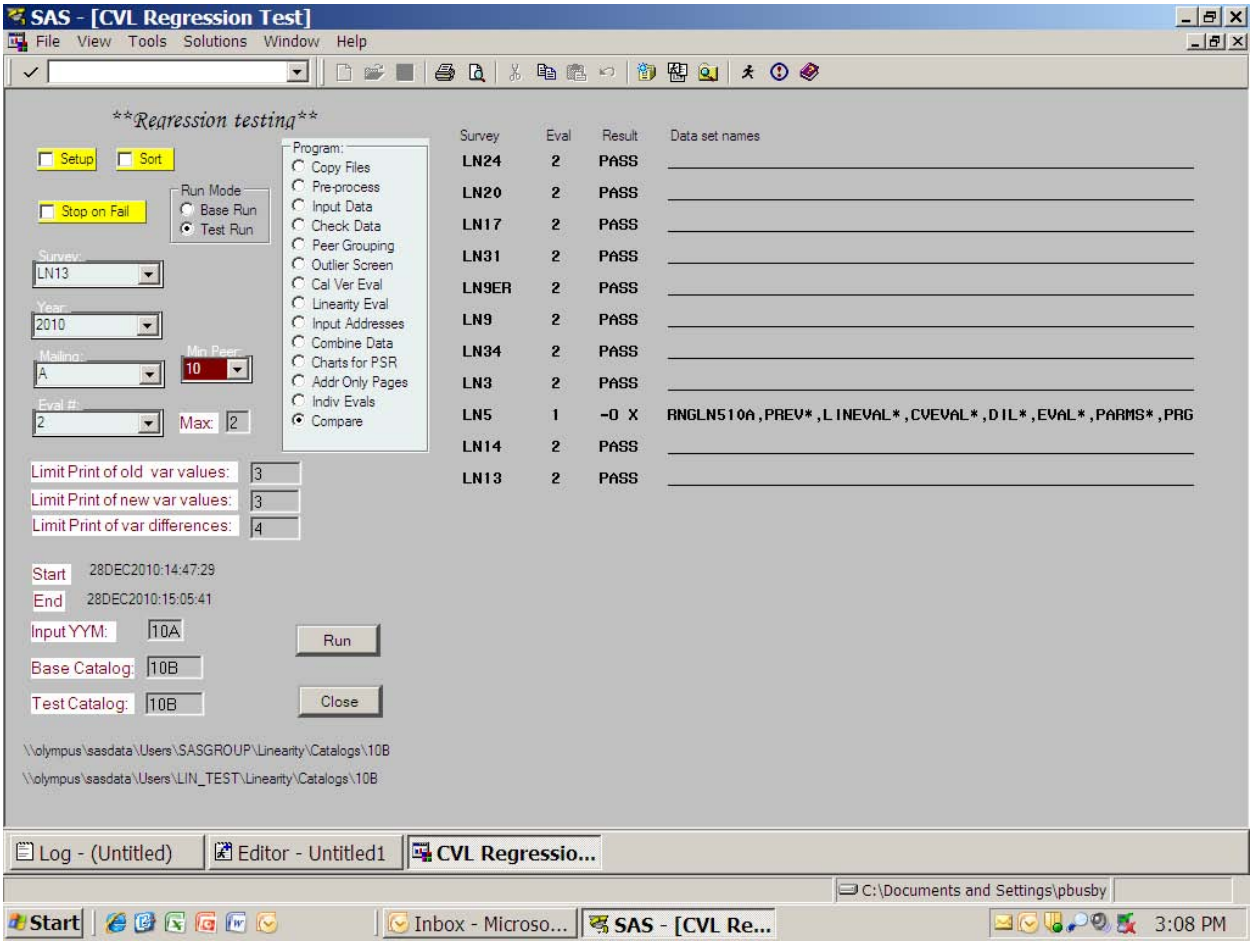
12		(Base:0260 Test:0250)(Base:11342010101 Test:12863020601)(Base:22767440 Test:22581016)(Base:DIL13 Test:DIL12)				
13	Pg, analyte, kitRef, kit_id	(Base:01 Test:03)(Base:pH Test:Chloride)(Base:14652010101 Test:14770020101)(Base:22459022 Test:22459020)			Yes	

Output 1. Regression Test Execution Summary Report

In the final version of this regression test, more processing steps were added (since they were changing output data set values!), and the user requested a different screen background color. (Display 4):



Display 4. Subsequent CVL Regression test Control Screen after execution



Display 5. Final CVL Regression test Control Screen after execution: the test pays off

When the test went live, it detected a change nobody had expected (Display 5). Better to find out early!

CONCLUSION

A SAS macro of 4,400 lines needed to be reworked. After two months of intermittent development, the macro was under 700 lines of code; 200 of those were comments; and the new functionality was working fine--thanks to a robust regression test. A mission-critical interactive reporting system was regression tested using a single SAS macro control variable for unmonitored execution, providing a black-box confirmation of correct system execution after any coding changes. To insure correct code, a regression test is the SAS programmer's best friend.

CONTACT INFORMATION

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

Name: Phil Busby
Enterprise: Live Data Systems, Inc.
Address: 15489 Ewen Airport Road, suite 201
City, State ZIP: Ewen, MI 49925
Work Phone: 919/523-0289
E-mail: fransioli@hotmail.com
Web: www.cdbaby.com/fransioli