Paper 1258-2017

Testing the night away

Stephan Minnaert, PW Consulting

ABSTRACT

Testing is a weak spot in many data warehouse environments. A lot of the testing is focused on the correct implementation of requirements. But due to the complex nature of analytics environments, a change in a data integration process can lead to unexpected results in totally different and untouched areas.

We developed a method to identify unexpected changes often and early by doing a nightly regression test. The test does a (complete) ETL run, compares all output from the test to a baseline, and reports all the changes. This paper describes the process and the SAS® code needed to back up existing data, trigger ETL flows, compare results, and restore situations after a nightly regression test. We also discuss the challenges we experienced while implementing the nightly regression test framework.

INTRODUCTION

When building a data warehouse, it takes a lot of effort to create all the jobs needed to extract, transform and eventually load data. Often the development consumes a lot of time. As deadlines to go live get closer, it leaves less time to properly test the newly created functionality. Even with the best manual testing processes it is hard to find all defects.

Dedicated testing teams are not always available. And those dedicated testers still need to do a lot of work 'by hand'. The solution could be to implement an automated testing tool. Automated testing tools offer the ability to repeat multiple test scenarios over and over again in a short amount of time. Often those tools are not available at projects.

Over the past couple of years more and more companies adapted an agile working method, like Continuous Delivery (CD), SCRUM or Kanban. The focus of a team is to quickly deliver working parts of software in short cycles. Although testing is an important part of those methodologies, the small amount of time available leads to a focus of testing the end product. A big concern is, or should be, testing all the steps needed to create an end product. Especially when a data warehouse is involved. Even when most of the steps are already working properly.

When using the scrum methodology, a sprint (the time box in which the product needs to be developed and made ready for production) usually takes up 2 to 4 weeks. Within this time period the developer needs to create his software, (unit) test is, promote it to a testing environment, get it tested, maybe get some rework done, promote it to a testing environment for a second time and then it's ready to be accepted by the business. Often it happens that testing the software gets squeezed into a couple of days. And that is a luxury already.

It's good to point out that in the Netherlands it's common practice to use the DTAP (development, test, acceptance, production) methodology for separate environments. The use of multiple environments together with agile working methods makes testing even more challenging within the given time. Figure 1 shows an overview of some of the basic actions needed to move software through the DTAP street.

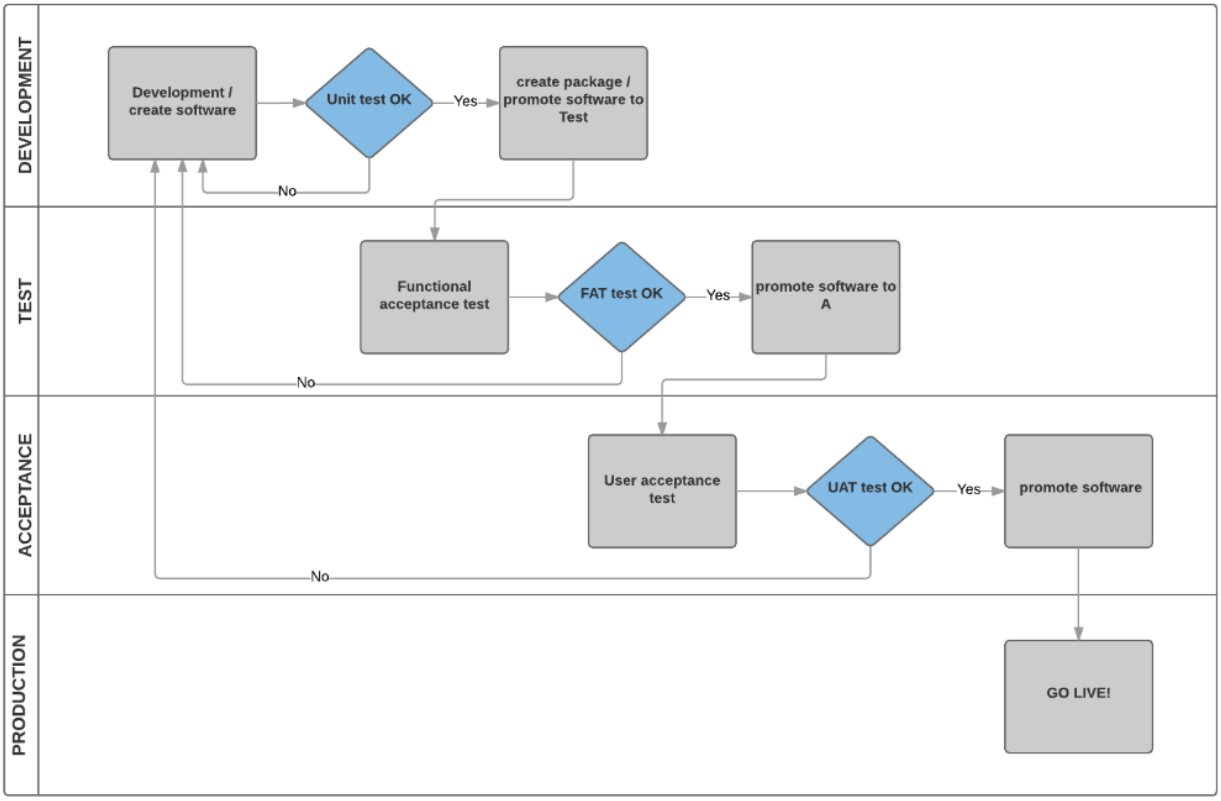


Figure 1. Simplified steps when delivering software through DTAP

Our personal experience led to the development of a test framework that runs every night on the development environment, using the same data over and over again, to detect the smallest changes in software as soon as possible in the development cycle. Even before the software is promoted to a testing environment. Most differences will be expected as a result of the changes made during the day. But sometimes those changes create unexpected differences. The sooner you find those changes the better.

This paper describes a concept for a test framework:

- Test preparation; preconditions for a successful test run
- Test framework; a description of steps taken by the framework supported by code snippits
- Limitations

INITIAL PREPARATION

When starting a project from scratch the use of a test framework is often not high on the priority list of things to do. It is advisable to start work on a framework as soon as possible. The longer you wait, the harder it gets to create a proper data set to start testing with. With each development the data can grow with it.

As soon as the first complete flows are in place and found to be working as designed, using a test framework becomes a useful addition of the work process. A test framework can't work without two crucial parts, test data and a baseline.

TEST DATA

Test data is key to a successful test run. A few considerations though. First define the purpose of the test run. When the main focus lies on the technical aspect of the jobs a simple test set will do the trick. When the test run needs to validate a lot of functional requirements the test set needs to be more detailed and realistic. This also means that it will probably be harder to create this test set, and it will be harder to maintain it as well when changes are made in the sources.

BASELINE

To be able to compare a test result it is mandatory to create a baseline. This first baseline has no reference.

The test run uses the batch flow as it runs in your production environment. It is advised not to create a separate flow in your scheduling tool to prevent differences between the production version and the test version. The test run can be used to test the batch as well.

Maintaining the baseline

As development progresses the baseline needs to be updated each time new functionality is added. Each morning a developer needs to check the outcome of the test run. Each time a difference is reported a developer, preferably the developer who was responsible for the software changes that led to the difference, should investigate the result. When the difference is not expected the developer needs to do some rework. When the results are as expected the baseline needs to be updated. The test result of the test run can be used as the new baseline.

TEST FRAMEWORK

Due to the big variety of software implementations the following considerations are merely an example of what to do. This paper should give an idea on how to implement a framework. Therefore, the process of getting the test run ready is described as a concept, sometimes supported by examples.

The test framework exists out of 3 flows containing DI Studio jobs build using LSF Platform Computing:

- Pre run; preparation of the environment
- Post run; compare the results and restore development
- Check run status; check if post run has started

Figure 2 shows an overview of the test framework.

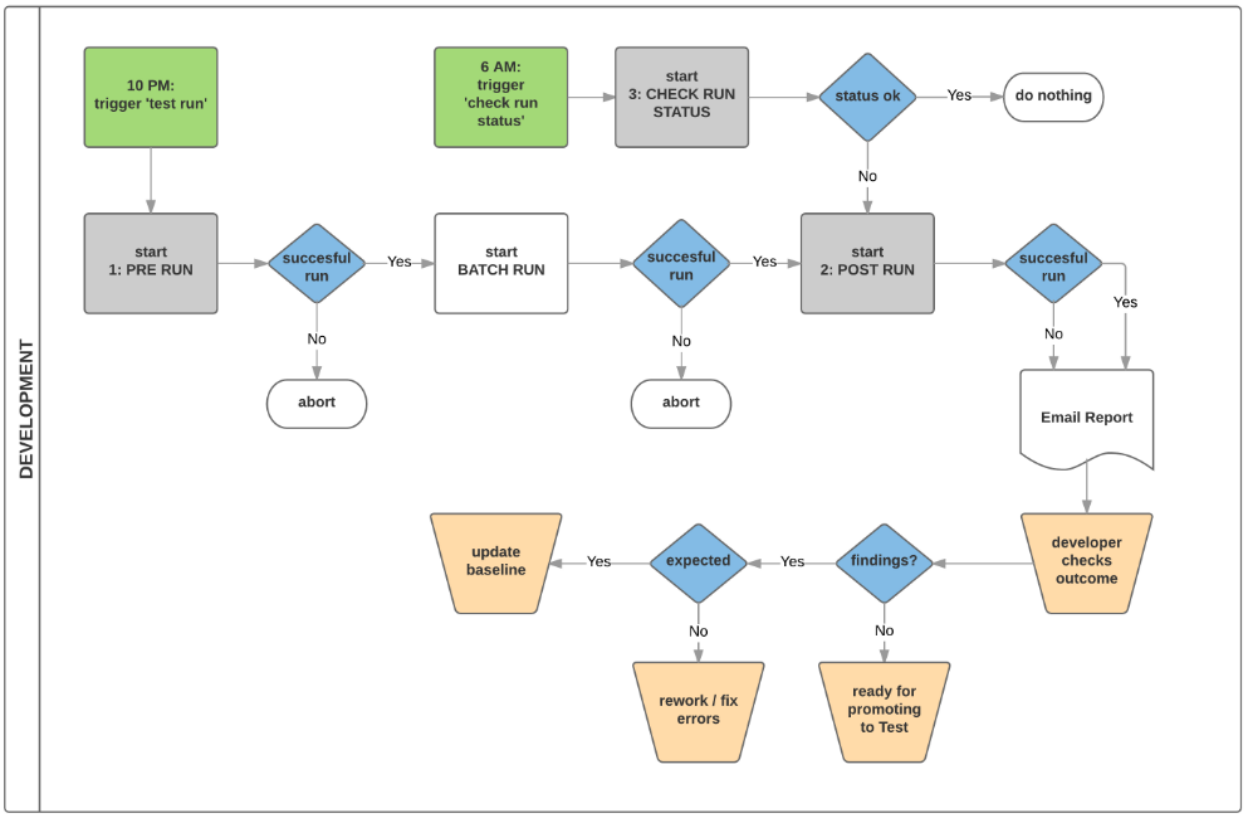


Figure 2. Conceptual overview test framework

STEP 1: PRE RUN

The first step is to prepare the development environment for the test run. To be able to compare data properly it's key to make sure that the initial state of the system is exactly equal before each test run starts, except of course for the newly developed software.

At 10.00 pm from Sunday until Thursday a time trigger starts the pre run. Running during the weekend days is only useful when a team member will be checking the results the next morning.

Figure 3. Pre run steps

Backup and initialize

All the data that is being used by the development team in all the libraries, or operating system folders is data that the team will need the following morning to continue their work. Therefore, it is crucial to store all data that will be touched by the test run in a separate folder. Those backups will be used to roll back the system after the test run is done.

Creating backups can be a time consuming task, especially when the datasets are large. The easy way to speed up this step is to move all datasets and create new, empty datasets based on the metadata. But

that would mean that testing whether new columns have been added to existing tables correctly would be skipped, as those new columns are now added by the create table script. That is an unwanted consequence of moving a complete dataset. Our advice is to truncate the datasets, after copying the data.

Copy source files

The test data is stored in a specific folder. Before the test run can start the test data needs to be copied to the folder from where the system can read the files. Each test run the same source files should be used.

Trigger test run

When all previous steps have been executed without any warnings the test run can be started by triggering the batch run. This trigger should be equal to the triggering mechanism used in your production environment.

STEP 2: POST RUN

The post run flow contains jobs that compares the test results with the baseline data, restore the development environment and report on the test results. The post run flow should be started as soon as the batch run has ended. How to determine when the batch run has finished depends completely on how your batch run is developed.

It is important that at least the restore of development will run when this flow is being executed. Therefore, all jobs in this flow will start running, even when the previous jobs exited with an exit code > 0. If the restore job won't run, the development team can't start working until the restore has been manually executed.

Figure 4. Post run steps

Compare results

After the test run has completed the results need to be compared. Using a proc compare works perfectly fine when using a couple of datasets. When comparing complete libraries which represent data warehouses the number of datasets, and amount of data, is large. A simple proc compare would take far too long to complete the test run in time. The data needs to be prepared for an efficient comparison of result with the baseline.

The compare functionality reports on columns that have been added or removed by joining the proc contents of the baseline and the test result using a full join. The following code

```
%macro compareDefinition (library=, baseDs=, compDs=);
    /* check if columns in base and compare are identical */
    /* baseDs = input data set                            */
    /* compDs = output data set                           */

    proc contents data=&baseDs.
      memtype=data
      out=_baseCols (keep=memname name type length format)
      nodetails
      noprint;
```

```
    run;

    proc contents data=&compDs.
      memtype=data
      out=_compareCols (keep=memname name type length format)
      nodetails
      noprint;
    run;

    proc sql noprint;
      create table diffColumns as
      select
      "&library." as library length=8,
      coalesce(t1.memname, t2.memname) as dsName,
      case
       when t2.name eq '' then catx(' ','Column', t1.name, 'is no longer
          available in testrun.')
       when t1.name eq '' then catx(' ','Column', t2.name, 'is added in
          this testrun.')
       when t1.type ne t2.type or
          t1.length ne t2.length or
          t1.format ne t2.format then catx(' ','Column Definition',
                t1.name, 'has changed.')
      end as reason length=100,
     "&SYSDATE9.:&SYSTIME."dt as processed_dttm format=datetime20.
      from _baseCols t1
          full join _compareCols t2
          on upcase(t1.name) = upcase(t2.name)
      where upcase(t1.name) ne upcase(t2.name) or
        t1.type ne t2.type or
        t1.length ne t2.length or
        t1.format ne t2.format
  ;
  quit;

  proc append base=TF.tf_runresults data=diffColumns force;
  run;

%mend compareDefinition;
```

The second comparison that is executed is on the contents of tables. Comparing each value in a dataset by ID would take a lot of time considering we were running a comparison against several hundred tables. We created a process that converts all character values into numeric values.

```
%macro allChar2toNum(outDS=, inDS=, numCharCols=, allCharCols=,
          allNewCols=, allNumCols=);

    data &outDS (keep=&allNewCols. &allNumCols.);
          set &inDS;
     %if &numCharCols. ne 0 %then %do;
      array chr{&numCharCols} &allCharCols;
      array newVars{&numCharCols} &allNewCols;

      do i = 1 to dim(chr);
          newVars{i} = input(put(put(md5(chr{i}),hex32.),octal64.),12.);
      end;
```

```
   %end;
run;

%mend allChar2toNum;
```

The next step is to aggregate the totals by column and compare the results using proc tabulate and report on the differences using another macro that is called from within the following code.

```
%macro compareTestRun(library=, baseDs=, compDs=, resDs=);

   %let nObs = 0;

    %let baseID = %sysfunc(open(&baseDs.));
    %let inObs = %sysfunc(attrn(&baseID.,nlobsf));
    %let rc   = %sysfunc(close(&baseID.));

    %let compID = %sysfunc(open(&compDs.));
    %let outObs = %sysfunc(attrn(&compID.,nlobsf));
    %let rc   = %sysfunc(close(&compID.));

   %if %eval(&inObs. gt 0 and &outObs gt 0) %then %do;

          %transCharCol2NumCol(inDS=&baseDs., outDS = work.transBase)
          %transCharCol2NumCol(inDS=&compDs., outDS = work.transComp)

          /*determine sums of the numeric columns and compare*/
          proc tabulate data=transBase out=work.baseSums;
                var _numeric_;

                table _numeric_, sum;
          run;

          proc tabulate data=transComp out=work.compSums;
                var _numeric_;

                table _numeric_, sum;
          run;

          proc compare base=work.basesums comp=work.compSums out=&resDs.
                                outdif outnoeq;
          run;

    %end;

    data _null_;
          call
execute('%appendResults('||"library=&library.,inObs=&inObs.,outObs=&outObs.
          ,baseDs=&baseDs.,resDs=&resDs.)");
    run;

   %compareDefinition(library=&library., baseDs=&baseDs., compDs=&compDs.);

%mend compareTestRun;
```

Appendix A provides the complete code of the custom transformation, which includes the macro's that are being called from the examples above.

Backup results

All data generated by the test run that contains results (data, reports, etc) should be stored in case the results are needed at a later point in time. For example, the results can be used to update de baseline data (as described earlier).

Restore development

The most important part of the test framework is restoring the development environment. This enables the developers to start working on new software development the minute they start their working day. All data stored in the backup step in the pre run, needs to be restored.

Delete old(est) test run

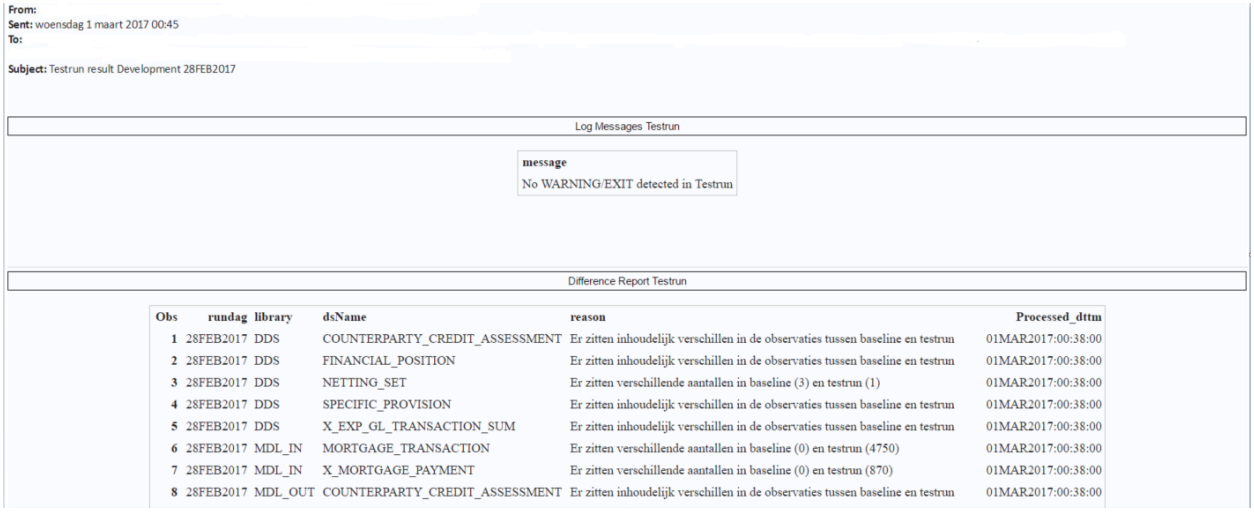
The step 'backup results' stores all test results after each test run. When testing a data warehouse this could lead to the use of a huge amount of storage. It is wise to start deleting the oldest test run data after a while. The number of days that should be stored depends on how much storage is available (and at what cost). We started deleting results after 7 days. We never had the need to look back further in time.

Report

The final step in the post run is reporting on the test results. The report is delivered to the developers by mail. Each morning when the developer starts working, feedback is presented on the condition of the test run.

When the batch run was cancelled by an abort you want to know which job is responsible for that abort. It is possible to use a log parser to extract information from the log files. Another option, when using LSF Platform Computing, is to extract data from the lsb.acct file. Both methods enable you to add jobs to the report that ran into warning or exit.

The result of the compare transformation is also added in the mailing. The following figure gives an example of what the mail could look like:



From:
Sent: woensdag 1 maart 2017 00:45
To:

Subject: Testrun result Development 28FEB2017

Log Messages Testrun

message
No WARNING/EXIT detected in Testrun

Difference Report Testrun

Obs	rundag	library	dsName	reason	Processed_dttm
1	28FEB2017	DDS	COUNTERPARTY_CREDIT_ASSESSMENT	Er zitten inhoudelijk verschillen in de observaties tussen baseline en testrun	01MAR2017:00:38:00
2	28FEB2017	DDS	FINANCIAL_POSITION	Er zitten inhoudelijk verschillen in de observaties tussen baseline en testrun	01MAR2017:00:38:00
3	28FEB2017	DDS	NETTING_SET	Er zitten verschillende aantallen in baseline (3) en testrun (1)	01MAR2017:00:38:00
4	28FEB2017	DDS	SPECIFIC_PROVISION	Er zitten inhoudelijk verschillen in de observaties tussen baseline en testrun	01MAR2017:00:38:00
5	28FEB2017	DDS	X_EXP_GL_TRANSACTION_SUM	Er zitten inhoudelijk verschillen in de observaties tussen baseline en testrun	01MAR2017:00:38:00
6	28FEB2017	MDL_IN	MORTGAGE_TRANSACTION	Er zitten verschillende aantallen in baseline (0) en testrun (4750)	01MAR2017:00:38:00
7	28FEB2017	MDL_IN	X_MORTGAGE_PAYMENT	Er zitten verschillende aantallen in baseline (0) en testrun (870)	01MAR2017:00:38:00
8	28FEB2017	MDL_OUT	COUNTERPARTY_CREDIT_ASSESSMENT	Er zitten inhoudelijk verschillen in de observaties tussen baseline en testrun	01MAR2017:00:38:00

Figure 5. Email example

The developer should be able to investigate why differences were detected using the data in the backup of the test run.

STEP 3: CHECK RUN STATUS

It is important that the post run has run successfully. Otherwise, the development team faces a problem the next morning. Therefore, at 06.00 am, a process is started that checks whether the post run has started. If not, then the post run is started afterwards in order to roll back the development environment.

The check on whether the post run was successful can be very simple, but depends (again) on how your environment has been set up. When working with triggers a simple check if a trigger has been created will work.

LIMITATIONS

During this paper we talked about datasets and source files. When using database connections some of the steps will get more complicated.

CONCLUSION

Creating an automated test framework is useful for every project. When there's no tooling for test automation available it is worthwhile considering to create a test framework with some basic SAS knowledge and smart comparison tools.

When starting as early as possible with implementing a test framework and building test data sets the amount of work can be overseen. The benefits of automated testing are huge.When the framework was running for the first time, immediately some defects where detected that were in the code for some time and had never been noticed before. It will help you to identify defects at an early stage without too much effort.

ACKNOWLEDGMENTS

The author would like to thank Laurent de Walick, Bas Marsman, John de Kroon, Yat Pang and Emma Bunte for their contribution in making the framework a success.

CONTACT INFORMATION

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

Stephan Minnaert
PW Consulting
stephan.minnaert@pwconsulting.nl
http://www.pwconsulting.nl

APPENDIX A: COMPARE TRANSFORMATION

```
%macro compareDefinition (library=, baseDs=, compDs=);
    /* check if columns in base and compare are identical */
    /* baseDs = input data set                            */
    /* compDs = output data set                           */

    proc contents data=&baseDs.
      memtype=data
      out=_baseCols (keep=memname name type length format)
      nodetails
      noprint;
    run;

    proc contents data=&compDs.
      memtype=data
      out=_compareCols (keep=memname name type length format)
      nodetails
      noprint;
    run;

    proc sql noprint;
      create table diffColumns as
      select
      "&library." as library length=8,
      coalesce(t1.memname, t2.memname) as dsName,
      case
       when t2.name eq '' then catx(' ','Column', t1.name, 'is no longer
          available in testrun.')
       when t1.name eq '' then catx(' ','Column', t2.name, 'is added in
          this testrun.')
       when t1.type ne t2.type or
          t1.length ne t2.length or
          t1.format ne t2.format then catx(' ','Column Definition',
                t1.name, 'has changed.')
      end as reason length=100,
     "&SYSDATE9.:&SYSTIME."dt as processed_dttm format=datetime20.
      from _baseCols t1
          full join _compareCols t2
          on upcase(t1.name) = upcase(t2.name)
      where upcase(t1.name) ne upcase(t2.name) or
        t1.type ne t2.type or
        t1.length ne t2.length or
        t1.format ne t2.format
 ;
 quit;

 proc append base=TF.tf_runresults data=diffColumns force;
 run;

%mend compareDefinition;

%macro allChar2toNum(outDS=, inDS=, numCharCols=, allCharCols=,
          allNewCols=, allNumCols=);
```

```
    data &outDS (keep=&allNewCols. &allNumCols.);
          set &inDS;
     %if &numCharCols. ne 0 %then %do;
      array chr{&numCharCols} &allCharCols;
      array newVars{&numCharCols} &allNewCols;

      do i = 1 to dim(chr);
          newVars{i} = input(put(put(md5(chr{i}),hex32.),octal64.),12.);
      end;
   %end;
run;

%mend allChar2toNum;

%macro transCharCol2NumCol(inDS=, outDS=, ignoreCols='processed_dttm');
    /*transform character columns to numerics*/
    /* inDS  = input data set                              */
    /* outDS = output data set               */
    /* obtain the current character variables              */
    /* create their numeric counterparts        */

   %let ignoreCols = %sysfunc(upcase(&ignoreCols.));

    data _null_;
          format tmpC tmpNewC tmpN $32000. countC 8.;
          call missing(of _all_);
          set &inDS. (obs=1);
          array num{*} _numeric_;
          array chr{*} _character_;
          countC = 0;

          do i = 1 to dim(chr);
                if upcase(vname(chr{i})) not in (&ignoreCols. 'TMPC'
                            'TMPNEWC' 'TMPN') then
                      do;
                            tmpC = catx(' ', tmpC, vname(chr{i}));
                            tmpNewC = catx(' ', tmpNewC,
                                  substr('N_'||strip(put(i,best. )),1,32));
                            countC+1;
                      end;
          end;

          do i = 1 to dim(num);
                if upcase(vname(num{i})) not in (&ignoreCols. 'COUNTC')
                then  do;
                      tmpN = catx(' ', tmpN, vname(num{i}));
                end;
          end;

      put "NOTE: For table &inDS the following info is derived";
      put "NOTE- number of character columns: " countC;
      put "NOTE- names of the character columns: " tmpC;
      put "NOTE- names of the new columns: " tmpNewC;
      put "NOTE- names of the numeric columns: " tmpN;
```

```
    call execute(cats('%allChar2toNum(outDS=',"&outDS., inDS=&inDS.,
                numCharCols=",countC,', allCharCols=',tmpC,',
                allNewCols=',tmpNewC,', allNumCols=',tmpN,')'));
    run;

%MEND transCharCol2NumCol;


%macro appendResults(library=, inObs=, outObs=, baseDs=, resDs=);

    %let nObs = 0;

    %if %sysfunc(exist(&resDs., DATA)) %then %do;

          %let resID = %sysfunc(open(&resDs.));
          %let nObs = %sysfunc(attrn(&resID.,nlobsf));
          %let rc   = %sysfunc(close(&resID.));

          proc sql;
                drop table &resDs.;
          quit;

    %end;

    %if %eval(&nObs. gt 0 or &inObs. ne &outObs.) %then %do;

          %if %eval(&nObs. gt 0) %then %do;

                data unequal;
                      format library $8. dsName $32. reason $100.
                            processed_dttm datetime20. ;
                      library = "&library.";
                      dsName = scan("&baseDs.",2,".");
                      reason = "Differences in content between baseline and
                                  test run ";
                      processed_dttm = "&SYSDATE9.:&SYSTIME."dt;
             run;
          %end;

          %if (&inObs. ne &outObs.) %then %do;

                data unequal;
                      format library $8.  dsName $32. reason $100.
                            processed_dttm datetime20. ;
                      library = "&library.";
                      dsName = scan("&baseDs.",2,".");
                      reason = "Differences in number of observations in
                                  baseline (&inObs.) and test run
                                    (&outObs.)";
                      processed_dttm = "&SYSDATE9.:&SYSTIME."dt;
                run;

          %end;

          proc append base=TF.tf_runresults data=unequal force;
       run;
    %end;
```

```
%macro compareTestRun(library=, baseDs=, compDs=, resDs=);

   %let nObs = 0;

    %let baseID = %sysfunc(open(&baseDs.));
    %let inObs = %sysfunc(attrn(&baseID.,nlobsf));
    %let rc   = %sysfunc(close(&baseID.));

    %let compID = %sysfunc(open(&compDs.));
    %let outObs = %sysfunc(attrn(&compID.,nlobsf));
    %let rc   = %sysfunc(close(&compID.));

   %if %eval(&inObs. gt 0 and &outObs gt 0) %then %do;

          %transCharCol2NumCol(inDS=&baseDs., outDS = work.transBase)
          %transCharCol2NumCol(inDS=&compDs., outDS = work.transComp)

          /*determine sums of the numeric columns and compare*/
          proc tabulate data=transBase out=work.baseSums;
                var _numeric_;

                table _numeric_, sum;
          run;

          proc tabulate data=transComp out=work.compSums;
                var _numeric_;

                table _numeric_, sum;
          run;

          proc compare base=work.basesums comp=work.compSums out=&resDs.
                                outdif outnoeq;
          run;

    %end;

    data _null_;
          call
execute('%appendResults('||"library=&library.,inObs=&inObs.,outObs=&outObs.
          ,baseDs=&baseDs.,resDs=&resDs.)");
    run;

   %compareDefinition(library=&library., baseDs=&baseDs., compDs=&compDs.);

%mend compareTestRun;

ods _all_ close;

data _null_;
 set &_input;
 call execute(cats('%compareTestRun(library=',&libref.,",
baseDs=",&baseTable.,", compDs=",&compTable.,", resDs=",&resTable.,")"));
run;
```