Paper 701-2017

The Future of Transpose: How SAS® Is Rebuilding Its Foundation by Making What Is Old New Again

Scott Mebust, SAS Institute Inc., Cary, NC

ABSTRACT

As computer technology advances, SAS® continually pursues opportunities to implement state-of-the-art systems that solve problems in data preparation and analysis faster and more efficiently. In this pursuit, we have extended the TRANSPOSE procedure to operate in a distributed fashion within both Teradata and Hadoop, using dynamically generated DS2 executed by the SAS® Embedded Process and within SAS® Viya™, using its native transpose action. With its new ability to work within these environments, PROC TRANSPOSE provides you with access to its parallel processing power and produces results that are compatible with your existing SAS programs.

INTRODUCTION

In a continuation of efforts to improve performance, process larger amounts of data, and take advantage of existing customer hardware and third-party software, SAS has enhanced the TRANSPOSE procedure to run in a distributed fashion within massively parallel processing (MPP) environments. These environments include the Teradata relational MPP database management system (DBMS), commodity computer clusters running Apache Hadoop, and SAS Cloud Analytic Services (CAS). CAS is a part of SAS Viya — the new and modern platform on which we are building the next generation of SAS analytics.

Operation within the Teradata DBMS and Hadoop, both known as in-database operation, is accomplished through the generation of DS2 program code submitted to the SAS Embedded Process (EP) for execution. Operation inside SAS Cloud Analytic Services, referred to here as in-CAS operation, is accomplished through a simple client request by PROC TRANSPOSE to invoke the new CAS transpose action on the CAS server. An understanding of the details of in-database transposition can provide you with an introduction to coding custom DS2 for in-database execution with the SAS® In-Database Code Accelerator. Understanding in-CAS operation can provide exposure to coding actions for execution by CAS. The DS2 generated for in-database operation can be examined in the SAS log and used to produce a stand-alone accelerated DS2 program that you can modify and extend. Likewise, the transpose action code generated for in-CAS operation can be examined and submitted to a CAS server, independently of PROC TRANSPOSE, using various clients and languages.

Even though the DBMS and CAS systems are very different from the traditional MultiVendor Architecture (MVA) system in which PROC TRANSPOSE usually runs, we have made great effort to ensure that in-database and in-CAS operation produce results that are compatible with the traditional system. If you've used PROC TRANSPOSE, then use of these new operational capabilities and their results should be familiar to you.

We encourage you to provide feedback on these new capabilities of PROC TRANSPOSE and on the new CAS transpose action. Your suggestions can help guide code development and make these tools more useful to you. One change that we are making in an upcoming release of SAS is the ability for the TRANSPOSE procedure to directly produce a stand-alone DS2 program that you can readily study and change. We hope that you find this change useful for exploration of DS2 and the In-Database Code Accelerator.

REQUIREMENTS

To take advantage of the new capabilities of PROC TRANSPOSE to run in-database with Teradata, when using Base SAS®, you need licenses for the SAS/ACCESS® Interface to Teradata, as well as SAS® In-Database Code Accelerator for Teradata. Similarly, for in-database operation on Hadoop, you need licenses for the SAS/ACCESS® Interface to Hadoop, as well as SAS® In-Database Code Accelerator for Hadoop. For operation in SAS Cloud Analytic Services, you need a license for SAS Viya 3.1 or a later release.

For the TRANSPOSE procedure to operate in-database, its input data set must be read from the DBMS and the results must be written to the DBMS. Likewise, for the TRANSPOSE procedure to operate in-CAS, it is necessary that the input resides in the server and the output data set is directed to the server. Other conditions, stated in sections below detailing the in-database and in-CAS examples, are also necessary for such operation. For SAS Viya, the transpose action operates only on input tables that are available in the CAS server and writes results to output tables in the same server.

GENERAL INTERNAL OPERATION

Transposition, as performed by SAS, is a dynamic transformation of an input data set, producing an output data set that is dependent not only on the transposition specification (the statements, variable lists, options, parameters, etc., that are coded by the user), but also on the characteristics of the variables to be transposed and the values of those variables within the input data. In particular, the types of variables to be transposed determine the types of the output variables. This determination is made with a type promotion scheme in which mixed input variable types result in an output variable type that is capable of representing the full range and precision of input values. The simplest example of this is that the transposition of both a numeric variable and a character variable results in character output variables. It is the transposition's dependence on the values of the input data, though, that makes it dynamic, not just statically defined by the transposition specification, and relieves the user from having to thoroughly know the input data prior to the transposition. A user does not have to specify an explicit list of variables to be created in the output data set or stipulate how values from input rows are assigned to those variables.

In the traditional MVA implementation of PROC TRANSPOSE, the in-database implementation, and the CAS transpose action implementation, the general process of transposition works similarly. All three implementations perform two read passes over the input data. The shape and characteristics of the output data set (the number of variables and their names) are determined during the first pass, while the actual transposition occurs during the second pass.

A SIMPLE TEST CASE

To demonstrate and focus on the details of in-database execution, in-CAS execution, and direct execution of the CAS transpose action, we transpose a very simple data set. The data set is a theoretical register or log of manufactured item measurements taken for quality control purposes, with three recorded values for each of two measurements within each batch:

```
data register;
  length property $ 20;
  input batch property value1 value2 value3;
  cards;
  1000 length    86.1 86.0 86.1
  1000 width     55.2 55.1 55.0
  1001 length    84.3 84.5 84.4
  1001 width     57.3 57.4 57.4
  ;
run;
```

Table 1 Shows the data to be transposed

Obs	batch	property	value1	value2	value3
1	1000	length	86.1	86.0	86.1
2	1000	width	55.2	55.1	55.0
3	1001	length	84.3	84.5	84.4
4	1001	width	57.3	57.4	57.4

Table 1: Register data set

For the sake of preparation or analysis, we transpose this data so that the two measurements (length and width) are each represented by one variable in the transposed data set:

```
proc transpose data=register out=measurements;
 by batch;
 id property;
 var value1-value3;
run;
```

Table 2 Shows the data after transposition

Obs	batch	_NAME_	length	width
1	1000	value1	86.1	55.2
2	1000	value2	86.0	55.1
3	1000	value3	86.1	55.0
4	1001	value1	84.3	57.3
5	1001	value2	84.5	57.4
6	1001	value3	84.4	57.4

Table 2: Measurements data set

IN-DATABASE EXAMPLE

To operate within a DBMS or Hadoop, we must start with the data set residing in the remote system, so we first place it there using a DATA step. For this example, the GRIDLIB library has been established using a SAS/ACCESS engine (either to Teradata or to Hadoop). To run in-database, even though in-database transposition is largely performed by running DS2 code, it is also necessary that the system SQLGENERATION option is properly set. The SQLGENERATION option remains relevant because in-database operation is traditionally controlled with this option and the same underlying infrastructure, as well as some generated SQL, is used to run the DS2. Use of the MSGLEVEL and SQL_IP_TRACE options provide more feedback in the SAS log regarding the process and internal operations. Note that both the DATA= and OUT= options reference data sets in the GRIDLIB library. Furthermore, note the use of the BY statement, the ID statement, the INDB=YES option setting, and the LET option. These are all required for in-database operation:

```
data gridlib.register;
 set register;
run;

options sqlgeneration=dbms;
options msglevel=i;
options sql_ip_trace=source;

proc transpose
    data=gridlib.register
    out=gridlib.measurements
    INDB=YES
    LET;
  by batch;
  id property;
  var value1-value3;
run;
```

The specification of at least one BY variable in a BY statement is required to define groups of observations that are transposed in parallel. One or more ID variables are required to define assignment of input rows to output columns and the names of those output columns. This assignment is made by associating a particular output column with the column name that was formed from the formatted values of the ID variables of an input observation. Without an ID variable, the assignment of values from input observations to output variables would be nondeterministic, with such assignments likely varying among BY groups, because of the nondeterminism inherent in MPP DBMSs and similar parallel processing systems. Such nondeterministic assignments would produce output that is difficult to validate and replicate. Because of this, we limit transpositions to ones requiring a mapping between input rows and output columns defined by one or more ID variables.

Finally, the LET option is required because PROC TRANSPOSE does not detect data loss, caused by more than one input observation within a BY group being transposed to a single output variable, during in-database operation. The TRANSPOSE procedure, when running in its traditional manner, stops with an error message during its first pass if the ID variables from two or more input observations within a BY group produce the same output variable name. The error means that two or more input observations within the group are assigned to the same output variable and only the values from the last transposed observation are output if execution continues. However, for efficiency of processing in the first pass, PROC TRANSPOSE does not attempt to detect this situation when running in-database. To protect you from data loss, PROC TRANSPOSE will not run in-database unless the LET option is specified. The LET option allows (or lets) this data loss to occur. Furthermore, because data loss is not detected, PROC TRANSPOSE does not issue any warning messages if data loss does occur when running in-database. Use the LET option to allow in-database operation if you understand your data and are certain that the specified ID variables ensure only one input observation is assigned to any one output variable or if you accept the possibility of data loss due to many-to-one assignments.

IN-DATABASE INTERNAL OPERATION

Two separate DS2 programs are generated and executed within the EP, one for each pass over the input data. The first program determines the shape of the output transposition by discovering the set of unique output column names generated from the ID variable values for each observation of the input data set. This DS2 program, indicated in the SAS log as the SHAPE QUERY, achieves a high degree of parallelism by performing the determination and discovery within its thread program using a sequential scan, with each thread reading a portion of data. The sets of column names generated by each thread are aggregated by a second stage data program that performs the same task. The DS2 code for this program is revealed by setting the SQL_IP_TRACE system option to SOURCE or ALL. The code for the above test case, when formatted, looks like this:

```
data sasep.out;

  retain "property" "_NAME_" ;
  declare double TPS_RC;
  declare package hash TPS_HASH();
  declare package hiter TPS_HITER('TPS_HASH');
  declare package tpsn TPS_TPSN('','','','V7');
  declare nchar(32) "_NAME_";
  keep "property" ;

  method init();
    TPS_HASH.defineKey('"_NAME_"');
    TPS_HASH.defineData('"_NAME_"');
    TPS_HASH.defineData('"property"');
    TPS_HASH.defineDone();
  end;

  method run();
    declare char(20) TPS_RAWVAL1;
```

```
    set sasep.in;
    if not ( missing( "property" )  );

    TPS_TPSN.reset();
    TPS_TPSN.build(["property"],'$',20,0);
    TPS_TPSN.finalize();

    "_NAME_" = TPS_TPSN.copyname();

    TPS_RAWVAL1 = "property";
    TPS_RC = TPS_HASH.find();
    if ( TPS_RC = 0 )
    then
      do;
        if ( TPS_RAWVAL1 < "property" )
        then "property" = TPS_RAWVAL1;
        TPS_RC = TPS_HASH.replace();
      end;
    else
      do;
        TPS_RC = TPS_HASH.add();
      end;
  end;

  method term();
    TPS_RC = TPS_HITER.first();
    do while ( TPS_RC = 0 );
      output;
      TPS_RC = TPS_HITER.next();
    end;
  end;

enddata;
```

This first program creates variable names using a DS2 package called TPSN, the sole purpose of which is to form output variable names from the formatted values of one or more ID variables that conform to SAS variable naming conventions, as determined by the VALIDVARNAME system option. The TPSN package is not documented and is for internal use by PROC TRANSPOSE and related code only; however, you can write your own DS2 package that performs a similar function, if you want to implement a similar transformation or use a different method for associating input observations to output variables. For every input observation, an output variable name is created and inserted into a Hash object, which maintains the list of unique names, along with representative raw values of the ID variables.

Note that, because this program is in a format that is suitable for execution by the EP, it is not directly executable as a stand-alone DS2. However, it can be transformed into a stand-alone program. Although the listed code appears to be a data program, it serves as both the DS2 thread program and second-stage aggregation data program. To transform this code into a stand-alone DS2 program, use the program structure below and follow the instructions within the comments:

```
proc ds2 indb=yes;
  thread th_pgm / overwrite=yes;

    /************************************************
      1. Insert code between DATA and ENDDATA here.
      2. Replace SASEP.IN with input data set name.
     ************************************************/
```

```
  endthread;
  run;

  /*************************************************
    1. Insert code, from DATA to ENDDATA inclusive, here.
    2. Replace SASEP.OUT with temporary data set name
    3. Declare a thread th_pgm M at top of data program
    4. Replace SET SASEP.IN with SET FROM M
   *************************************************/
 run;
quit;
```

The resulting stand-alone program is:

```
proc ds2 indb=yes;

  thread th_pgm / overwrite=yes;

    retain "property" "_NAME_" ;
    declare double TPS_RC;
    declare package hash TPS_HASH();
    declare package hiter TPS_HITER('TPS_HASH');
    declare package tpsn TPS_TPSN('','','','V7');
    declare nchar(32) "_NAME_";
    keep "property" ;

    method init();
      TPS_HASH.defineKey('"_NAME_"');
      TPS_HASH.defineData('"_NAME_"');
      TPS_HASH.defineData('"property"');
      TPS_HASH.defineDone();
    end;

    method run();
      declare char(20) TPS_RAWVAL1;

      set gridlib.register;
      if not ( missing( "property" )  );

      TPS_TPSN.reset();
      TPS_TPSN.build(["property"],'$',20,0);
      TPS_TPSN.finalize();

      "_NAME_" = TPS_TPSN.copyname();

      TPS_RAWVAL1 = "property";
      TPS_RC = TPS_HASH.find();
      if ( TPS_RC = 0 )
      then
        do;
          if ( TPS_RAWVAL1 < "property" )
          then "property" = TPS_RAWVAL1;
          TPS_RC = TPS_HASH.replace();
        end;
      else
        do;
```

```
        TPS_RC = TPS_HASH.add();
      end;
  end;

  method term();
    TPS_RC = TPS_HITER.first();
    do while ( TPS_RC = 0 );
      output;
      TPS_RC = TPS_HITER.next();
    end;
  end;

endthread;
run;

data gridlib.metadata;

  declare thread th_pgm m;

  retain "property" "_NAME_" ;
  declare double TPS_RC;
  declare package hash TPS_HASH();
  declare package hiter TPS_HITER('TPS_HASH');
  declare package tpsn TPS_TPSN('','','','V7');
  declare nchar(32) "_NAME_";
  keep "property" ;

  method init();
    TPS_HASH.defineKey('"_NAME_"');
    TPS_HASH.defineData('"_NAME_"');
    TPS_HASH.defineData('"property"');
    TPS_HASH.defineDone();
  end;

  method run();
    declare char(20) TPS_RAWVAL1;

    set from m;
    if not ( missing( "property" )  );

    TPS_TPSN.reset();
    TPS_TPSN.build(["property"],'$',20,0);
    TPS_TPSN.finalize();

    "_NAME_" = TPS_TPSN.copyname();

    TPS_RAWVAL1 = "property";
    TPS_RC = TPS_HASH.find();
    if ( TPS_RC = 0 )
    then
      do;
        if ( TPS_RAWVAL1 < "property" )
        then "property" = TPS_RAWVAL1;
        TPS_RC = TPS_HASH.replace();
      end;
    else
      do;
```

```
          TPS_RC = TPS_HASH.add();
        end;
    end;

    method term();
      TPS_RC = TPS_HITER.first();
      do while ( TPS_RC = 0 );
        output;
        TPS_RC = TPS_HITER.next();
      end;
    end;

  enddata;
  run;
quit;
```

You can run this first program within the local SAS session if you change INDB=NO. As is, with INDB=YES, the code runs within the EP using DS2 code acceleration. In this case, you should see the following notes in the SAS log:

```
NOTE: Running THREAD program in-database
NOTE: Running DATA program in-database
```

The second program, generated with the aid of the output variable metadata that was returned from the execution of the first program, actually performs the transposition. This DS2 program, indicated in the SAS log as the TRANSPOSE QUERY, achieves parallelism through the transposition of independent BY groups. BY-group processing, triggered by the presence of the BY statement within a thread program, causes the underlying system (Teradata or Hadoop) to shuffle and order the observations so that they are grouped before processing. For Teradata, this involves the use of the BY variables in HASH BY and ORDER BY clauses on the SELECT statement of the SQL query that provides data to the EP. For Hadoop, the BY variables are used as the Map/Reduce keys, which results in a shuffle and sort.

Partitioning of the data can be costly, in terms of computational resources. Pre-partitioning of the input tables within those systems, in preparation for such group processing, can eliminate the need to shuffle the data when running the transposition. You might choose to pre-partition the input if partitioned access is required for more than one purpose. For a more detailed description of how DS2 code can be accelerated within the EP, see the recommended reading material listed at the end of this paper. Again, the DS2 code for this program can be revealed by setting the SQL_IP_TRACE system option to SOURCE or ALL. The formatted code for the above test case is:

```
data sasep.out;
  retain "batch" "_NAME_" "_LABEL_" ;
  retain "width" "length" ;
  declare package hash TPS_HASH();
  declare package tpsn TPS_TPSN('','','','V7');
  declare integer TPS_COL;
  declare nchar(32) "_NAME_" having label 'NAME OF FORMER VARIABLE';
  declare nchar(32) TPS_INVNAM[ 3 ];
  declare nchar(40) "_LABEL_";
  declare nchar(40) TPS_INVLAB[ 3 ];
  vararray double TPS_INVARS[ 3 ] "value1" "value2" "value3" ;
  vararray double TPS_OUTVARS[ 2 ] "width" "length" ;
  declare double TPS_TPOSETMP[ 3, 2 ];
  keep "batch" "_NAME_" "_LABEL_" "width" "length" ;

  method init();
    TPS_HASH.defineKey('"_NAME_"');
```

```
  TPS_HASH.defineData('"_NAME_"');
  TPS_HASH.defineData('"TPS_COL"');
  TPS_HASH.defineDone();

  "_NAME_" = 'width';
  TPS_COL = 1;
  TPS_HASH.add();

  "_NAME_" = 'length';
  TPS_COL = 2;
  TPS_HASH.add();

  TPS_INVNAM := ( 'value1' 'value2' 'value3'  );
  TPS_INVLAB := ( 'value1' 'value2' 'value3'  );
end;

method run();
  declare integer TPS_ROW;
  declare double TPS_RC;
  set sasep.in ;
  by "batch" ;
  if not ( missing( "property" )  )
  then
    do;
      TPS_TPSN.reset();
      TPS_TPSN.build(["property"],'$',20,0);
      TPS_TPSN.finalize();
      "_NAME_" = TPS_TPSN.copyname();

      TPS_RC = TPS_HASH.find();
      if ( TPS_RC )
      then
        do;
          put 'ERROR: Unexpected error, _NAME_ not found.';
          stop;
        end;

      do TPS_ROW = 1 to 3;
        TPS_TPOSETMP[ TPS_ROW, TPS_COL ] = TPS_INVARS[ TPS_ROW ];
      end;
    end;

  if last."batch"
  then
    do;
      do TPS_ROW = 1 to 3;
        "_NAME_" = TPS_INVNAM[ TPS_ROW ];
        "_LABEL_" = TPS_INVLAB[ TPS_ROW ];
        do TPS_COL = 1 to 2 ;
          TPS_OUTVARS[ TPS_COL ] = TPS_TPOSETMP[ TPS_ROW, TPS_COL ];
        end;
        output;
      end;
      TPS_TPOSETMP := ( 3 * ( 2 * NULL ) );
    end;
end;
```

```
  method term();
  end;
enddata;
```

Although the generated code is enclosed in a DATA … ENDDATA block, this code is really the DS2 thread program, which executes independently, but in parallel, with each thread processing one or more BY groups. The DS2 code begins with declarations of variables, both those used to accomplish the transposition, but not included in the output, and those results of the transposition kept for the output. The type of variable elements in the transposition matrix, TPS_TPOSETMP, as well as the types of the output variables WIDTH and LENGTH, are determined by the type of those input variables listed on the VAR statement. In this case, the variables VALUE1, VALUE2, and VALUE3 are all of type double, thus the transposition matrix and output variables are doubles as well. The output type is simple to determine for this example, but the type of promotion scheme used to determine the output variable type supports the full complement of DS2 variable types. These declarations are followed by the initialization of a Hash Object with key/value pairs that map names produced by the formatted values of ID variables, for each observation to a corresponding column in the transposition matrix. The keys of the key/value pairs used to initialize the Hash Object are the output variable names determined during the first pass from the execution of the first DS2 program.

The code then processes a group of observations by reading each observation of the group, transposing the values for the variables of interest of that observation into a column of the transposition matrix. The ordinal of the column into which the observation is transposed is identified by searching the Hash Object for the name produced from the ID variable values of that observation. After all observations of a group are processed in this way, the transposition is completed by outputting each row of the transposition matrix. After output is complete, the transposition matrix is cleared so that another observation group can be processed.

This code will not execute as a stand-alone DS2 program, because it is generated to be executed by the EP. For the purposes of learning, modifying, or extending the code, you can easily transform the code into a stand-alone DS2 program by using the program structure below and following the instructions within the comments:

```
proc ds2 indb=yes;

  thread th_pgm / overwrite=yes;

    /*************************************************
      1. Insert code between DATA and ENDDATA here.
      2. Replace SASEP.IN with input data set name.
     *************************************************/

  endthread;
  run;


  /*************************************************
    1. Create a new data program here that does nothing
       but declares and sets from a thread.
    2. Direct output of the data program to a data set
       on the grid.

    For example:
   *************************************************/

  data LIBNAME.DATASETNAME(overwrite=yes);

    declare thread th_pgm m;
```

```
    method run();
     set from m;
    end;

  enddata;
  run;

quit;
```

Arranged this way, with the data program doing nothing but invoking the thread program, the threads perform parallel output of the transposed data.

The resulting stand-alone program for the above test case is:

```
proc ds2 indb=yes;

  thread th_pgm / overwrite=yes;

    retain "batch" "_NAME_" "_LABEL_" ;
    retain "width" "length" ;
    declare package hash TPS_HASH();
    declare package tpsn TPS_TPSN('','','','V7');
    declare integer TPS_COL;
    declare nchar(32) "_NAME_" having label 'NAME OF FORMER VARIABLE';
    declare nchar(32) TPS_INVNAM[ 3 ];
    declare nchar(40) "_LABEL_";
    declare nchar(40) TPS_INVLAB[ 3 ];
    vararray double TPS_INVARS[ 3 ] "value1" "value2" "value3" ;
    vararray double TPS_OUTVARS[ 2 ] "width" "length" ;
    declare double TPS_TPOSETMP[ 3, 2 ];
    keep "batch" "_NAME_" "_LABEL_" "width" "length" ;

    method init();
      TPS_HASH.defineKey('"_NAME_"');
      TPS_HASH.defineData('"_NAME_"');
      TPS_HASH.defineData('"TPS_COL"');
      TPS_HASH.defineDone();

      "_NAME_" = 'width';
      TPS_COL = 1;
      TPS_HASH.add();

      "_NAME_" = 'length';
      TPS_COL = 2;
      TPS_HASH.add();

      TPS_INVNAM := ( 'value1' 'value2' 'value3'  );
      TPS_INVLAB := ( 'value1' 'value2' 'value3'  );
    end;

    method run();
      declare integer TPS_ROW;
      declare double TPS_RC;
      set gridlib.register;
      by "batch" ;
      if not ( missing( "property" )  )
      then
```

```
        do;
          TPS_TPSN.reset();
          TPS_TPSN.build(["property"],'$',20,0);
          TPS_TPSN.finalize();
          "_NAME_" = TPS_TPSN.copyname();

          TPS_RC = TPS_HASH.find();
          if ( TPS_RC )
          then
            do;
              put 'ERROR: Unexpected error, _NAME_ not found.';
              stop;
            end;

          do TPS_ROW = 1 to 3;
            TPS_TPOSETMP[ TPS_ROW, TPS_COL ] =
              TPS_INVARS[ TPS_ROW ];
          end;
        end;

      if last."batch"
      then
        do;
          do TPS_ROW = 1 to 3;
            "_NAME_" = TPS_INVNAM[ TPS_ROW ];
            "_LABEL_" = TPS_INVLAB[ TPS_ROW ];
            do TPS_COL = 1 to 2 ;
              TPS_OUTVARS[ TPS_COL ] =
                TPS_TPOSETMP[ TPS_ROW, TPS_COL ];
            end;
            output;
          end;
          TPS_TPOSETMP := ( 3 * ( 2 * NULL ) );
        end;
    end;

    method term();
    end;

  endthread;

  run;

  data gridlib.measurements(overwrite=yes);

    declare thread th_pgm m;
    method run();
     set from m;
    end;

  enddata;

  run;

quit;
```

Like the stand-alone DS2 program for the first-pass, you can run this program within the local SAS session if you change INDB=NO. With INDB=YES, the code runs within the EP, using DS2 code acceleration. When run within the EP, you should see the following notes in the SAS log:

```
NOTE: Running THREAD program in-database
NOTE: Running DATA program in-database
```

IN-CAS EXAMPLE

To operate within the CAS server, we must also start with the data set residing in the server. For this example, having established a SASCAS1 libref using the SAS/ACCESS engine to CAS, we copy the data from the WORK library to the server using a DATA step. Unlike in-database operation, no DS2 code is generated and no EP is involved. Instead, PROC TRANSPOSE on the SAS client simply requests that the CAS transpose action is invoked on its behalf within the CAS server. Other than both the DATA= and OUT= options referencing data sets in the SASCAS1 library, the only requirement here for in-CAS operation is the use of an ID statement and the specification of one or more ID variables to determine the transposition:

```
data sascas1.register;
  set register;
run;

proc print data=sascas1.register;
  var batch property value1-value3;
run;

options msglevel=i;

proc transpose
    data=sascas1.register
    out=sascas1.measurements;
  by batch;
  id property;
  var value1-value3;
run;
```

While not strictly required, using a BY statement with the specification of one or more BY variables is suggested because, like in-database operation, parallelism in the transposition is achieved through the distribution of different BY groups across the computing nodes in the system and the simultaneous processing of BY groups on those nodes. A transposition with no BY statement implies a single BY group and the processing of all data on a single node.

The LET option is not required for in-CAS operation, because the underlying CAS transpose action detects during the first pass when the values of two or more observations within an input group are assigned to a single output variable. If the LET option has not been specified, like traditional PROC TRANSPOSE, the action stops with an error when data loss is detected. If you encounter such an error, you can use the LET option to ignore the data loss and allow the transposition to complete.

IN-CAS INTERNAL OPERATION

For execution within the CAS server, the TRANSPOSE procedure on the SAS client issues a request to the server to run the CAS transpose action. That request can be revealed by recalling the history of client and server interactions, using the HISTORY statement with the CAS procedure:

```
proc cas;
  session sascas1;
  run;
```

```
  history verbose first=-4 last=-4;
  run;
quit;
```

The FIRST and LAST options are used to extract the specific interaction containing the transpose action request. After requesting the history, the output of the action code can be extracted from the SAS log. For this test case, that action code is:

```
action transpose.transpose /
    table={name='REGISTER', groupBy={{name='batch'}}},
    id={'property'},
    casOut={name='MEASUREMENTS', replace=true},
    validVarName='v7',
    transpose={'value1', 'value2', 'value3'};
```

While the syntax of the transpose action is obviously different from that of the TRANSPOSE procedure, the meaning of the action parameters is likely reasonably clear, because most parameters can be directly related to procedure statements and options. The action's TABLE parameter corresponds to the DATA option of the TRANSPOSE procedure statement. Likewise, the CASOUT parameter corresponds to the OUT option of the procedure statement. The ID parameter is the equivalent of the procedure's ID statement. The GROUPBY parameter, a sub-parameter of the action's TABLE parameter, contains the list of variables that appear on the BY statement. The VALIDVARNAME parameter is set for compatibility with the client, based on the SAS client setting of the VALIDVARNAME system option. Finally, the action's transpose parameter takes the list of variables to be transposed that appear on the VAR statement. Operation of the transpose action within CAS is described below.

INVOKING TRANSPOSE ACTION DIRECTLY

This action code, revealed in the SAS log by the CAS procedure's HISTORY statement and presented in the CAS language, can be used directly within PROC CAS. Running PROC TRANSPOSE is familiar to SAS programmers and integrates neatly into standard SAS code. Running the action code directly from PROC CAS allows expression of a work and data flow, using the power of the new CAS language:

```
proc cas;
  session sascas1;
  run;

  loadactionset "transpose";
  run;

  action transpose.transpose /
    table={name='REGISTER', groupBy={{name='batch'}}},
    id={'property'},
    casOut={name='MEASUREMENTS', replace=true},
    validVarName='v7',
    transpose={'value1', 'value2', 'value3'};
  run;
quit;
```

In addition to the CAS language, all CAS actions can be invoked using other CAS client languages, such as Lua and Python, as well as through a web service using HTTP with a REST API*.

*Using the Hypertext Transfer Protocol (HTTP) with a Representational State Transfer (REST) application programming Interface (API).

TRANSPOSE ACTION INTERNAL OPERATION

An action in CAS is roughly the equivalent of a procedure in the traditional SAS system. The transpose action is a native part of CAS and does not generate any intermediate code for execution. Because it is coded at a lower level and can open its input table in two-pass mode, it first requests that the input table be partitioned according to the GROUPBY variables and then performs two passes: the first to determine the shape of the output table and the second to perform the transposition. Making two passes over the grouped data allows the action to check for potential data loss, like traditional PROC TRANSPOSE does. If data loss is detected and the LET option is not specified, the action issues one or more error messages and stops execution. If the LET option is specified and data loss is detected, the action issues one or more warning messages, but continues execution.

Similar to traditional and in-database operation, the action determines the shape and characteristics of the output table during its first pass over the input. It ascertains the type of output variables based on the input variables specified for transposition and supports the new CAS variable types. Like the TRANSPOSE procedure, if no variables are specified for transposition, then all double-precision, floating-point variables that are not used for other purposes (as ID or GROUPBY variable, for instance) are transposed.

COMPATIBILITY

With any changes that SAS makes in new releases of software in use by customers, whether adding new features to existing procedures or adding whole new methods of operation, we take great care to maintain compatibility with the behavior and results to which customers are accustomed. While it is not always possible to build upon the existing system and maintain 100% compatibility with prior releases, it is certainly the goal for which we strive. The creation of the TRANSPOSE procedure's new in-database and in-CAS modes of operation are no exception. We have tried to ensure that the results you obtain from PROC TRANSPOSE and the new CAS transpose action are consistent with traditional results that are produced by the SAS system. For these new capabilities, we have paid particular attention to the treatment of missing values, characteristics of output data sets, formatting of variables in output, and use of formats in the formation of variables names. In doing so, we hope to provide new power and performance, while preserving your investment in our software.

FUTURE DIRECTIONS

We encourage you to try the new capabilities of PROC TRANSPOSE to perform its work in remote systems, such as Teradata and Hadoop, using in-database operation and SAS Viya using in-CAS operation. We hope that you find the DS2 code generated for in-database operation useful for building upon or simply learning about DS2 and the SAS In-Database Code Accelerator. We also encourage you to investigate the new SAS Viya platform as we introduce it. We welcome feedback regarding the changes in PROC TRANSPOSE, as well as the introduction of the CAS transpose action. In addition to continued focus on efficiency, quality, and compatibility of this new software, we intend to make it easier to run PROC TRANSPOSE jobs in-database and to produce stand-alone DS2 programs from the generated code used for in-database operation. To produce a stand-alone program, we envision a new procedure option that writes the DS2 code to a text file.

RECOMMENDED READING

Secosky, Jason, et al. 2014. "Parallel Data Preparation with the DS2 Programming Language." *Proceedings of the SAS Global Forum 2014 Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings14/SAS329-2014.pdf.

Ghazaleh, David. 2016. "Exploring SAS® Embedded Process Technologies on Hadoop®." *Proceedings of the SAS Global Forum 2016 Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings16/SAS5060-2016.pdf.

De Capite, Donna. 2014. “Techniques in Processing Data on Hadoop®.” *Proceedings of the SAS Global Forum 2014 Conference*. Cary, NC: SAS Institute Inc. Available at https://support.sas.com/resources/papers/proceedings14/SAS033-2014.pdf

Ray, Robert, et al. 2016. “Data Analysis with User-Written DS2 Packages.” *Proceedings of the SAS Global Forum 2016 Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings16/SAS6462-2016.pdf

Secosky, Jason, et al. 2007. “Getting Started with the DATA Step Hash Object.” *Proceedings of the SAS Global Forum 2007 Conference*. Cary, NC: SAS Institute Inc. Available at http://www2.sas.com/proceedings/forum2007/271-2007.pdf

SAS Institute Inc. 2015. *SAS 9.4 In-Database Products, User’s Guide, 5th ed.* Cary, NC: SAS Institute Inc. Available at https://support.sas.com/documentation/cdl/en/indbug/68170/PDF/default/indbug.pdf

SAS Institute Inc. 2016. *SAS 9.4 DS2 Language Reference, 6th ed.* Cary, NC: SAS Institute Inc. Available at

http://support.sas.com/documentation/cdl/en/ds2ref/69739/PDF/default/ds2ref.pdf

SAS Institute Inc. 2016. *Base SAS 9.4 Procedures Guide, 6th ed*. Cary, NC: SAS Institute Inc. Available at

http://support.sas.com/documentation/cdl/en/proc/69850/PDF/default/proc.pdf

ACKNOWLEDGMENTS

Thanks to all those who reviewed this paper and provided feedback on its content. In particular, thanks to Robert Ray, Laura Gold, Mark Freskos, and John West.

CONTACT INFORMATION

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

Scott Mebust
100 SAS Campus Drive
Cary, NC 27513
SAS Insitute Inc.
Scott.Mebust@sas.com
http://www.sas.com