PhUSE EU Connect 2018

Paper CT09

Please put your data in an upright position – or how to tackle verticalisation

Kristina Zweier, Clinipace, Eschborn, Germany

ABSTRACT

When preparing data according to CDISC[1] standards, programmers are often challenged with the task of displaying horizontal source data in a normalized structure, as this is the general data structure of CDISC compliant domains. Choosing the most feasible and efficient way to do so can be a challenge in itself.
This paper provides several programming options for verticalisation as well as suggestions regarding their feasibility. Furthermore, the paper provides an example on merging data in horizontal data structure with data in vertical structure.

INTRODUCTION

Complying with and implementing CDISC standards can – in some circumstances - lead to special challenges when it comes to the verticalisation of data. This will be shown in this paper on the basis of typical CDISC mapping that is applied regularly.

Often times the implementation of CDISC specifications happens by defining those specifications in an Excel table and then, by reading the table into SAS, creating templates for the individual CDISC domains that are going to be submitted. The creation of the domains is then realised by mapping the templates with study data and arranging the data in the correct order[2]. This is of course easier said than done.

Using this approach, this paper will show two possible ways (PROC TRANSPOSE and the OUTPUT[3] statement in a data step) of rearranging horizontal data into CDISC – or more precisely: an SDTM[4] – compliant structure, using both simple and more complicated constructed raw data. In the process the paper will explain the difficulties and feasibilities of each verticalisation approach, especially in regard to the standard strategy of merging SDTM domain templates with raw study data to create CDISC/SDTM compliant domains.

PROC TRANSPOSE VS. OUTPUT STATEMENT

Let's start with a simple dataset. Table 1 shows variables of a raw vital signs dataset. Vital signs were assessed for each patient at Screening and at the Final Visit. The results of pulse rate (PULSE) and respiratory rate (RESP) are presented as two different variables. The table shows the results of three subjects from two different sites. As you can see, subject 102_001 has no observation for the Final Visit, therefore we can conclude that the respective CRF[5] page has not (yet) been filled out.

Table 1: Original vital signs dataset

	STUDYID	SITEID	SUBJID	VISIT	VISITNUM	VSDT	PULSE	RESP
1	PHUSE2018	101	101_001	Screening	0	11DEC2017	92	12
2	PHUSE2018	101	101_001	Final Visit	99	16APR2018	79	16
3	PHUSE2018	101	101_002	Screening	0	11DEC2017	86	14
4	PHUSE2018	101	101_002	Final Visit	99	23FEB2018	85	15
19	PHUSE2018	102	102_001	Screening	0	12DEC2017	68	16

The goal is to arrange the data from this vital signs dataset according to the SDTM specifications which are already set in an existing SDTM.VS template dataset. This template consists of the SDTM variables (without actual data) that are required to be used in accordance with the SDTM guidelines to create an SDTM-compliant vital signs domain.
Table 2 presents an example of an SDTM.VS template with some of the possible SDTM.VS variables.

Table 2: Template with some of the possible variables for the SDTM domain VS – vital signs

STUDYID	DOMAIN	USUBJID	VSTESTCD	VSTEST	VSORRES	VSORRESU	VSSTAT	VSREASND	VISITNUM	VISIT

By re-arranging the raw data VS dataset and merging it with the SDTM.VS template the desired SDTM.VS domain is created. At first, the PROC TRANSPOSE approach will be used to get the desired result.

```
PROC TRANSPOSE DATA = vs_raw OUT = vs_raw_t (rename = (COL1 = vsres _NAME_ = testcd));
  BY studyid siteid subjid visitnum visit vsdt;
  VAR pulse resp;
RUN;
```

The variables listed in the VAR-statement will be transposed within the variables listed in the BY-statement. Those variables will not be transposed[6].
In the output dataset the variable that stores the results of the transposed variables is re-named from "COL1" (default SAS name) to "VSRES" and the name of the variable that stores the names of the transposed variables is changed from "_NAME_" (default SAS name) to "TESTCD". Table 3 shows an extract of the transposed dataset:

Table 3: Transposed vital signs dataset for subject 101_001

	STUDYID	SITEID	SUBJID	VISITNUM	VISIT	VSDT	TESTCD	_LABEL_	VSRES
1	PHUSE2018	101	101_001	0	Screening	11DEC2017	PULSE	Pulse Rate	92
2	PHUSE2018	101	101_001	0	Screening	11DEC2017	RESP	Respiratory Rate	12
3	PHUSE2018	101	101_001	99	Final Visit	16APR2018	PULSE	Pulse Rate	79
4	PHUSE2018	101	101_001	99	Final Visit	16APR2018	RESP	Respiratory Rate	16

After the creation of the transposed dataset that is now in the desired vertical structure, it is then appended to the template dataset for the SDTM.VS domain.

The SELECT statement (see code below) is used to go through the available values of the variable TESTCD to assign SDTM compliant values to the SDTM.VS variables VSTESTCD, VSTEST[7] and VSORRESU. Furthermore the values of the test results storing variable VSRES, which was created through transposing, are assigned to the SDTM.VS variable VSORRES.
Based on the same original dataset, using the OUTPUT statement in a data step would lead to the same result.

Using the OUTPUT statement, one observation per subject and assessment for every non null entry of the vital signs variables in the dataset – PULSE and RESP – is created. The individual observations will be written into the new dataset after the execution of the OUTPUT statement.

As in this case the data is complete without any missing values, adding the original dataset to the SDTM.VS template and using the OUTPUT statement to create SDTM.VS variables in one data step can be done without complications. This is not always the case as will be demonstrated.

```
DATA vs_transpose;
   SET sdtmtmpl.vs
       vs_raw_t ;
   studyid = _studyid;
   usubjid = trim(studyid)||"-"||trim(subjid);
   domain = ;

   SELECT (testcd);
       WHEN ("PULSE") DO;
         vstestcd = "PULSE";
         vstest = "Pulse Rate";
         vsorresu = "beats/min";
       END;
       WHEN ("RESP") DO;
         vstestcd = "RESP";
         vstest = "Respiratory Rate";
         vsorresu = "breaths/min";
       END;
       OTHERWISE
       call missing (vstestcd,vstest,vsorresu);
   END;
   vsorres = strip(put(vsres, best.));
RUN;
```

```
DATA vs_output;
   SET sdtmtmpl.vs
       vs_raw ;
   studyid=_studyid;
   usubjid = trim(studyid)||"-"||trim(subjid);
   domain ="VS";

   IF pulse ne . THEN DO;
     vstestcd = "PULSE";
     vstest   = "Pulse Rate";
     vsorresu = "beats/min";
     vsorres  = strip(put(pulse, best.));
       OUTPUT;
   END;
   IF resp ne . THEN DO;
     vstestcd = "RESP";
     vstest   = "Respiratory Rate";
     vsorresu = "breaths/min";
     vsorres  = strip(put(resp, best.));
       OUTPUT;
   END;
RUN;
```

Considering the simple and complete state of the raw data dataset, choosing one of the two approaches in this scenario comes down to personal preferences. When the original dataset is more complex, the correct verticalisation with both approaches becomes more complicated. Table 4 shows a part of the resulting dataset based on both approaches.

Table 4: SDTM.VS domain for subject 101_001

	STUDYID	DOMAIN	USUBJID	VSTESTCD	VSTEST	VSORRES	VSORRESU	VISITNUM	VISIT
1	PHUSE2018	VS	PHUSE2018-101_001	PULSE	Pulse Rate	92	beats/min	0	Screening
2	PHUSE2018	VS	PHUSE2018-101_001	RESP	Respiratory Rate	12	breaths/min	0	Screening
3	PHUSE2018	VS	PHUSE2018-101_001	PULSE	Pulse Rate	79	beats/min	99	Final Visit
4	PHUSE2018	VS	PHUSE2018-101_001	RESP	Respiratory Rate	16	breaths/min	99	Final Visit

PITFALLS AND WORK-AROUNDS FOR INCOMPLETE DATA

Unfortunately handling a dataset such as the simple one presented in our example is rarely the case in our day to day work as the structure of the datasets we are working with tends to be more complex.

Table 5 presents a vital signs dataset with missing values. Let's look at possible verticalisation approaches with PROC TRANSPOSE and the OUTPUT statement when dealing with incomplete data.

Table 5: Original incomplete vital signs dataset

	STUDYID	SITEID	SUBJID	VISIT	VISITNUM	VSDT	VSND	PULSE	PULSEND	RESP	RESPND
1	PHUSE2018	101	101_001	Screening	0	11DEC2017	0	92	0	12	0
2	PHUSE2018	101	101_001	Final Visit	99	16APR2018	0	.	1	16	0
3	PHUSE2018	101	101_002	Screening	0	11DEC2017	0	86	0	14	0
4	PHUSE2018	101	101_002	Final Visit	99	23FEB2018	1	.	0	.	0
5	PHUSE2018	102	102_001	Screening	0	12DEC2017	0	68	0	.	1

This dataset contains three additional categorical variables with values of 0 and 1 that display whether one, or all vital signs assessments have or have not been performed for a specific visit. The default value of those variables is set to 0. If none of the vital signs assessments have been performed at a specific visit, variable VSND is set to 1, while the individual "NOT DONE"-variables PULSEND and RESPND remain the default value of 0. If only individual assessments have not been performed, the individual "NOT DONE"-variables are set to 1 while VSND remains 0.

It is preferable not to overcrowd SDTM domains with redundant information. Therefore, we want to avoid creating all individual observations for a visit if none of them have been performed. So, if variable VSND is set to 1, which indicates that none of the assessments have been performed, just one observation for the respective visit should be created and the SDTM variable VSTESTCD should be set to the value "VSALL" while VSTEST should be set to the value "Vital Signs". Instead of filling the SDTM variable VSORRES with a value – as no assessment result is available – SDTM variable VSSTAT will be set to "NOT DONE" according to the SDTM rules. As the CRF does not ask for a reason why the vital signs were not assessed, the variable VSREASND (reason not done) will be set to "REASON NOT COLLECTED".

If only individual assessments have not been performed at a visit (PULSEND or RESPND is then set to 0), the individual observations with VSTESTCD set to "PULSE" or "RESP" and VSTEST set to "Pulse Rate" or "Respiratory Rate" will be created but variable VSORRES will again be set to missing while variables VSSTAT and VSREASND will be filled with values "NOT DONE" and "REASON NOT COLLECTED".

Let's have a look at how to achieve our goal best with the PROC TRANSPOSE and afterwards with the OUTPUT statement. As the information of the "NOT DONE" variables is needed, they could be included in the VAR statement in the transposing procedure.

```
PROC TRANSPOSE DATA = vs_raw OUT = vs_raw_t (rename = (COL1 = vsres _NAME_ = testcd));
   BY studyid siteid subjid visitnum visit vsdt;
   VAR vsnd pulse pulsend resp respnd;
RUN;
```

The resulting dataset for subject 101_001 at the Final Visit is shown in Table 6.

Table 6: Transposed incomplete vital signs dataset for subect 101_001

STUDYID	SITEID	SUBJID	VISITNUM	VISIT	VSDT	TESTCD	_LABEL_	VSRES
PHUSE2018	101	101_001	99	Final Visit	16APR2018	VSND	Vitals Not Done	0
PHUSE2018	101	101_001	99	Final Visit	16APR2018	PULSE	Pulse Rate	.
PHUSE2018	101	101_001	99	Final Visit	16APR2018	PULSEND	Pulse Not Done	1
PHUSE2018	101	101_001	99	Final Visit	16APR2018	RESP	Respiratory Rate	16
PHUSE2018	101	101_001	99	Final Visit	16APR2018	RESPND	Resp Rate Not Done	0

The previously horizontal arranged variables are now arranged in one variable, one name and result after the other. If the transposed dataset is now added to the template dataset as in the first example, one would face some problems while trying to create the SDTM.VS domain.

8

```
DATA vs_transpose;
  SET sdtmtmpl.vs
      vs_raw_t;

  usubjid = trim(studyid)||"-"||trim(subjid);
  domain = "VS";
  SELECT (testcd);
      WHEN ("VSND") DO;
         vstestcd = "VSALL";
         vstest = "Vital Signs";
         vsorresu = "";
      END;
      WHEN ("PULSE", "PULSEND" ) DO;
        vstestcd = "PULSE";
        vstest = "Pulse Rate";
        vsorresu = "beats/min";
      END;
      WHEN ("RESP","RESPND") DO;
        vstestcd = "RESP";
        vstest = "Respiratory Rate";
        vsorresu = "breaths/min";
      END;
      OTHERWISE
      call missing (vstestcd,vstest,vsorresu);
  END;
   vsorres = strip(put(vsres, best.));
```

9

```
  SELECT ;
      WHEN (testcd = "VSND" and vsres = 1) DO;
         vsstat = "NOT DONE";
         vsreasnd = "REASON NOT COLLECTED";
         CALL missing (of vsorres:);
      END;
      WHEN (testcd = "PULSEND" and vsres = 1) DO;
         vsstat = "NOT DONE";
         vsreasnd = "REASON NOT COLLECTED";
         CALL missing (of vsorres:);
      END;
      WHEN (testcd = "RESPND" and vsres = 1) DO;
         vsstat = "NOT DONE";
         vsreasnd = "REASON NOT COLLECTED";
         CALL missing (of vsorres:);
      END;
      OTHERWISE call missing (vsstat,vsreasnd);
  END;
RUN;
```

Although the SELECT statement can be used to create observations with information about assessment results or assessments that were not performed, due to the inclusion of the variables that indicate an assessment has not been performed in the list of variables that should be transposed, the resulting dataset includes double entries instead of the desired single observations.

Table 7 shows the resulting dataset from our data step.
The first observation where VSTESTCD stores the value VSALL and the last observation where the variable stores the value RESP are actually not needed for this subject as they store the value of 0 for the variable VSORRES, therefore assessments have been performed.

On the other hand, we do need the transposed variable PULSEND for this subject as this specific assessment has not been performed for subject 101_001 at the Final Visit.

As you can see, re-arranging incomplete data after transposing can get quite tricky. You need to make sure to only include those variables in the VAR statement that actually need to be transposed.

Table 7: SDTM.VS-like dataset of incomplete data for subject 101_001[1011]

STUDYID	DOMAIN	USUBJID	VSTESTCD	VSTEST	VSORRES	VSORRESU	VSSTAT	VSREASND	VISITNUM	VISIT
PHUSE2018	VS	PHUSE2018-101_001	VSALL	Vital Signs	0				99	Final Visit
PHUSE2018	VS	PHUSE2018-101_001	PULSE	Pulse Rate		beats/min			99	Final Visit
PHUSE2018	VS	PHUSE2018-101_001	PULSE	Pulse Rate			NOT DONE	REASON NOT COLLECTED	99	Final Visit
PHUSE2018	VS	PHUSE2018-101_001	RESP	Respiratory Rate	16	breaths/min			99	Final Visit
PHUSE2018	VS	PHUSE2018-101_001	RESP	Respiratory Rate	0	breaths/min			99	Final Visit

Fortunately there is a better way to use the TRANSPOSE procedure while also including the information if assessments have not been performed: merging the transposed dataset with the original pre-transposed dataset.
First, the original dataset is transposed, but this time variables PULSEND and RESPND, which indicate if a single assessment has not been performed, are not included into the list of variables that are supposed to be transposed.

```
PROC TRANSPOSE DATA = vs_raw OUT = vs_raw_t (rename = (COL1 = vsres _NAME_ = testcd));
   BY studyid siteid subjid visitnum visit vsdt;
   VAR vsnd pulse resp;
RUN;
```

We receive the same transposed dataset as in Table 6, only this time the observations with the values “PULSEND” and “RESPND” for variable TESTCD are missing. Second, the transposed dataset is merged with the original, horizontal dataset.

```
DATA vs_merged;
   MERGE vs_raw_t
         vs_raw;
   BY studyid siteid subjid visitnum visit vsdt;
RUN;
```

Let's look at a compressed[12] output of the result dataset for subjects 101_001 and 101_002 at the Final Visit (Table 8).

Table 8: Merged, transposed and original dataset for patients 101_001 and 101_002

STUDYID	SITEID	SUBJID	VISITNUM	VISIT	VSDT	TESTCD	_LABEL_	VSRES	VSND	PULSE	PULSEND	RESP	RESPND
PHUSE2018	101	101_001	99	Final Visit	16APR2018	VSND	Vitals Not Done	0	0	.	1	16	0
PHUSE2018	101	101_001	99	Final Visit	16APR2018	PULSE	Pulse Rate	.	0	.	1	16	0
PHUSE2018	101	101_001	99	Final Visit	16APR2018	RESP	Respiratory Rate	16	0	.	1	16	0
PHUSE2018	101	101_002	99	Final Visit	23FEB2018	VSND	Vitals Not Done	1	1	.	0	.	0
PHUSE2018	101	101_002	99	Final Visit	23FEB2018	PULSE	Pulse Rate	.	1	.	0	.	0
PHUSE2018	101	101_002	99	Final Visit	23FEB2018	RESP	Respiratory Rate	.	1	.	0	.	0

As we remember from the original dataset of our incomplete data (Table 5), pulse rate assessment has not been performed for patient 101_001 at the Final Visit, while for patient 101_002 no vital signs assessments at all have been performed. This is also depicted in the new, merged dataset (Table 8) but now the dataset stores three observations per subject and visit as every observation from the transposed dataset (three variables have been transposed, therefore three individual observations per subject and visit have been created) is merged to every observation per subject and visit in the original dataset. In a data step similar to the one before the desired SDTM.VS domain can now be created.

In contrast to the previous data step, the values PULSEND and RESPND are not included in the select statement to create the observations for the vital signs pulse rate and respiratory rate. Therefore, unnecessary double entries are avoided.

```
DATA vs_transpose;
  SET sdtmtmpl.vs
      vs_merged;

  usubjid = trim(studyid)||"-"||trim(subjid);
  domain = "VS";

  SELECT (testcd);
    WHEN ("VSND") DO;
      vstestcd = "VSALL";
      vstest = "Vital Signs";
      vsorresu = "";
    END;
    WHEN ("PULSE") DO;
      vstestcd = "PULSE";
      vstest = "Pulse Rate";
      vsorresu = "beats/min";
    END;
    WHEN ("RESP") DO;
      vstestcd = "RESP";
      vstest = "Respiratory Rate";
      vsorresu = "breaths/min";
    END;
    OTHERWISE
      call missing
      (vstestcd,vstest,vsorresu);
  END;
  vsorres = strip(put(vsres, best.));

  SELECT;
    WHEN (testcd = "VSND" and vsnd = 1) DO;
      vsstat = "NOT DONE";
      vsreasnd = "REASON NOT COLLECTED";
      call missing (of vsorres:);
    END;
    WHEN (testcd = "PULSE" and pulsend = 1) DO;
      vsstat = "NOT DONE";
      vsreasnd = "REASON NOT COLLECTED";
      CALL missing (of vsorres:);
    END;
    WHEN (testcd = "RESP" and respnd = 1) DO;
      vsstat = "NOT DONE";
      vsreasnd = "REASON NOT COLLECTED";
    CALL missing (of vsorres:);
    END;
    OTHERWISE call missing (vsstat,vsreasnd);
  END;

  IF vsnd eq 1 and testcd ne "VSND" THEN DELETE;
  IF vsnd ne 1 and testcd eq "VSND" THEN DELETE;
RUN;
```

Furthermore, due to the merging of the transposed dataset with the original, un-transposed one, the values of variable TESTCD, which was created through transposing, and the values of the original, un-transposed variables VSND, PULSEND and RESPND can be used to determine for which entries it is needed to create observations that indicate that single tests or all vital signs tests have not been done for individual visits.

As the variable VSND has been transposed to be able to create a single observation per subject and visit if no vital signs assessment at all has been performed, there are still a few unnecessary entries in the dataset.
Therefore, observations of individual assessments – that is, observations with the values "PULSE" and "RESP" for the variable TESTCD – that store the value of 1 for variable VSND are omitted as this means that no assessments have been performed for this visit and only one observation with the value "VSND" for the variable TESTCD is to remain in this case.

Furthermore, observations are omitted, that have the value “VSND” for variable TESTCD but the value 0 for variable VSND as this indicates that either all assessments have been performed or only individual, but not all, assessments are missing. In these cases the individual observations will be created and either the variable VSORRES or variables VSSTAT and VSREASND will be filled – depending on if the individual assessment has been performed or not.

Table 9 shows a compressed[13] output of the result dataset for subjects 101_001 and 101_002 at the Final Visit.
As you can see, the pulse rate has not been assessed for subject 101_001 at the Final Visit but the respiratory rate has. Therefore the individual assessment observations are created by filling variables VSSTAT and VSREASND for one and variables VSORRES and VSORRESU for the other observation.

Table 9: SDTM.VS-like dataset of incomplete data for subjects 101_001 and 101_002 at Final Visit

STUDYID	DOMAIN	USUBJID	VSTESTCD	VSTEST	VSORRES	VSORRESU	VSSTAT	VSREASND	VISITNUM	VISIT
PHUSE2018	VS	PHUSE2018-101_001	PULSE	Pulse Rate			NOT DONE	REASON NOT COLLECTED	99	Final Visit
PHUSE2018	VS	PHUSE2018-101_001	RESP	Respiratory Rate	16	breaths/min			99	Final Visit
PHUSE2018	VS	PHUSE2018-101_002	VSALL	Vital Signs			NOT DONE	REASON NOT COLLECTED	99	Final Visit

No vital signs assessments have been performed at the Final Visit for subject 101_002, therefore only one observation has been created with a general umbrella term for the vital signs assessment name and variables VSSTAT and VSREASND have been filled while variables VSORRES and VSORRESU remain empty.

As shown, when dealing with incomplete data, verticalisation of data through PROC TRANSPOSE can be achieved by merging verticalised data with horizontal data to have a comfortable starting position when it comes to creating an SDTM domain.

If you are not comfortable using PROC TRANSPOSE, you can achieve your desired result with the OUTPUT statement in a data step as well but you will need to keep some things in mind.

WATCH OUT WHAT YOU PUT OUT – OBSTACLES OF THE OUTPUT STATEMENT

We have looked at a way to create SDTM domains by complementing an SDTM domain template with all needed variables including predefined labels, variable types and variable lengths with the original data. After working with complete data in our previous example, let’s look at an example on how to use the OUTPUT statement with incomplete data.

When creating a new dataset by combining two datasets with the SET or the MERGE statement as in our case, keep in mind the working mode of the Program Data Vector (PDV) and the possibility of automatic retain of values from one observation to the next[14]. To avoid automatic retain one needs to state the desired value for each variable that could be affected by it. This includes assigning a missing value to variables when necessary.

This means, if either none of the vital signs assessments have been performed or individual assessments have not been performed for a visit, the values for VSORRES and VSORRESU need to be set to missing. On the other hand, if assessments have been performed, variables VSSTAT and VSREASND need to be set to missing.
If this is not being done, values of the previous observation that has been created in the output dataset would be taken over to the next variable[15].

```
DATA vs_output;
   SET sdtmtmpl.vs
       vs_raw (RENAME=(studyid=_studyid));

   studyid = _studyid;
   usubjid = trim(studyid) || "-" ||
   trim(subjid);
   domain = "VS";
   IF vsnd eq 1 THEN DO;
      vstestcd = "VSALL";
      vstest   = "Vital Signs";
      vsorresu = "";
      vsstat   = "NOT DONE";
      vsreasnd = "REASON NOT COLLECTED";
      vsorres = "";
      OUTPUT;
   END;
ELSE DO;
   IF pulse ne . THEN DO;
      vstestcd = "PULSE";
      vstest   = "Pulse Rate";
      vsorresu = "beats/min";
      vsstat   = "";
      vsreasnd = "";
      vsorres  = strip(put(pulse, best.));
      OUTPUT;
   END;
   ELSE IF pulsend eq 1 THEN DO;
      vstestcd = "PULSE";
      vstest   = "Pulse Rate";
      vsstat   = "NOT DONE";
      vsreasnd = "REASON NOT COLLECTED";
      vsorres  = "";
      OUTPUT;
   END;
      IF resp ne . THEN DO;
         vstestcd = "RESP";
         vstest   = "Respiratory Rate";
         vsorresu = "breaths/min";
         vsstat   = "";
         vsreasnd = "";
         vsorres  = strip(put(resp, best.));
         OUTPUT;
      END;
      ELSE IF respnd eq 1 THEN DO;
         vstestcd = "RESP";
         vstest   = "Respiratory Rate";
         vsstat   = "NOT DONE";
         vsreasnd = "REASON NOT COLLECTED";
         vsorres  = "";
         OUTPUT;
      END;
   END;
RUN;
```

One possible way to avoid the automatic RETAIN option of the PDV is splitting the program code into two data steps. In a first data step the input dataset would simply be added to the template dataset through a SET statement and afterwards the data would be verticalised in a second data step.

Although the problem of the automatic retaining of values from one observation to the next is circumvented by splitting the programming into two data steps, when it comes to the OUTPUT statement, you need to keep in mind that several output observations are created out of one input observation (as in our case, creating one observation for every assessment per subject and visit) where varying variables in the output dataset need to be filled or remain empty for each output observation.

This means that although the automatic retaining of values from one observation to the next is of no concern, similar mechanisms are in place in these cases. Therefore, you still need to make sure to assign the correct value – may it be a present or missing one – for each variable that is of concern in every OUTPUT statement.

Finally, one needs to consider if the OUTPUT statement approach is a feasible option depending on the state of the original dataset. Not only is it necessary to include all variables that could be affected by the OUTPUT statement to store incorrect values, depending on the number of variables in the original dataset based on which the vertical observations are created (as in our case the individual vital signs assessments), using the OUTPUT statement can lead to an inflated data step that is difficult to oversee and prone to programming errors.

If you are familiar with macro coding, using a macro code could be a way to use the OUTPUT statement approach while limiting the possibilities of program errors. This approach could especially be the appropriate choice if the individual OUTPUT statements are similar or almost the same as in our example. All three of the OUTPUT statement approaches lead to the same output dataset (Table 10).

```
*) 1.;
%MACRO cre8vs(invar=, testcd=, test=, unit=, status=, reason=);

   *) For each new parameter, set all values to missing;
   call missing (vstestcd, vstest, vsorres, vsorresu, vsstat, vsreasnd);

   vstestcd = upcase("&testcd");
   vstest   = strip("&test");
   vsorresu = strip("&unit.");

   %IF &invar. ne %THEN %DO;
      vsorres  = strip(put(&invar., best.));
   %END;

   vsstat = strip("&status.");
   vsreasnd = strip("&reason.");
%MEND;
```

```
*) 2.;
DATA vs_output0;
  SET sdtmtmpl.vs
      vs_raw;
RUN;
```

```
*) 3.;
DATA vs_output;
  SET vs_output0;

  usubjid = trim(studyid) || "-" || trim(subjid);
  domain = "VS";

  IF vsnd eq 1 THEN DO;
    %cre8vs(testcd=VSALL, test=Vital Signs, status=NOT DONE, reason=REASON NOT COLLECTED);
    OUTPUT;
  END;
  ELSE DO;
    IF pulse ne . THEN DO;
      %cre8vs(invar=pulse, testcd=PULSE, test=Pulse Rate, unit=beats/min, status=,reason=);
      OUTPUT;
    END;
    ELSE IF pulsend eq 1 THEN DO;
      %cre8vs(testcd=PULSE, test=Pulse Rate, status=NOT DONE, reason=REASON NOT COLLECTED);
      OUTPUT;
    END;
    IF resp ne . THEN DO;
      %cre8vs(invar=resp, testcd=RESP, test=Respiratory Rate, unit=breaths/min, status=,reason=);
      OUTPUT;
    END;
    ELSE IF respnd eq 1 THEN DO;
      %cre8vs(testcd=RESP, test=Respiratory Rate, status=NOT DONE, reason=REASON NOT COLLECTED);
      OUTPUT;
    END;
  END;
RUN;
```

Table 10: SDTM.VS table with selected variables

STUDYID	DOMAIN	USUBJID	VSTESTCD	VSTEST	VSORRES	VSORRESU	VSSTAT	VSREASND	VISITNUM	VISIT
PHUSE2018	VS	PHUSE2018-101_001	PULSE	Pulse Rate	92	beats/min			0	Screening
PHUSE2018	VS	PHUSE2018-101_001	RESP	Respiratory Rate	12	breaths/min			0	Screening
PHUSE2018	VS	PHUSE2018-101_001	PULSE	Pulse Rate			NOT DONE	REASON NOT COLLECTED	99	Final Visit
PHUSE2018	VS	PHUSE2018-101_001	RESP	Respiratory Rate	16	breaths/min			99	Final Visit
PHUSE2018	VS	PHUSE2018-101_002	PULSE	Pulse Rate	86	beats/min			0	Screening
PHUSE2018	VS	PHUSE2018-101_002	RESP	Respiratory Rate	14	breaths/min			0	Screening
PHUSE2018	VS	PHUSE2018-101_002	VSALL	Vital Signs			NOT DONE	REASON NOT COLLECTED	99	Final Visit
PHUSE2018	VS	PHUSE2018-102_001	PULSE	Pulse Rate	68	beats/min			0	Screening
PHUSE2018	VS	PHUSE2018-102_001	RESP	Respiratory Rate			NOT DONE	REASON NOT COLLECTED	0	Screening

CONCLUSION

Using PROC TRANSPOSE as well as the OUTPUT statement in a data step are both possible ways to transform horizontal arranged datasets into vertical arranged ones. However, depending on the state of the original data, using additional work-arounds may be necessary.

ACKNOWLEDGMENTS

I would like to thank my colleagues at Clinipace, especially Christina Gelhorn, Anita Eisberg, and Gaye Lucas, for their support during the creation of this paper.

CONTACT INFORMATION

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

Kristina Zweier
Clinipace
Helfmann-Park 10
Eschborn / D-65760
Email: kzweier@clinipace.com
Web: https://www.clinipace.com/

1 Clinical Data Interchange Standards Consortium
2 As use of a pre-defined template is a common approach in creating SDTM domains and due to the limited scope of this paper, the topic of creating such templates will not be discussed.
3 Due to the limited space of this paper, detail about the exact functioning of the TRANSPOSE procedure and the OUTPUT statement will not be presented. For further details, visit the SAS support page at https://support.sas.com/en/support-home.html.
4 Study Data Tabulation Model
5 Case Report Form
6 When using the BY-statement with PROC TRANSPOSE, the dataset should be sorted according to the BY-variables prior to transposing.
7 In our example, the vital signs names and labels are SDTM-compliant, meaning, they correspond to SDTM.VS test codes and test names; therefore we could also assign the values of the variables TESTCD and _LABEL_ to VSTESTCD and VSTEST. As this is not always the case, a safer approach is to assign the names directly as above.
8 If a single assessment was not performed, the variable test code and test name still have to represent the respective assessment, therefore the variables that indicate that an assessment has not been performed, are included in the SELECT statement.
9 In this SELECT statement, SDTM variable VSSTAT will be set to “NOT DONE” if the value of the result variable VSRES for the observations with those values for the variable TESTCD that indicate that either all assessments for a visit (VSND) or individual assessments (PULSEND, RESPND) have not been performed, is 1.
10 Double entries for pulse rate and respiratory rate because of inclusion of “Not Done” variables PULSEND and RESPND in the SELECT statement.
11 Due to limited space this table does not include all variables of the output dataset but is still comprehensible.
12 Due to limited space this table does not include all variables of the output dataset but is still comprehensible.
13 Due to limited space this table does not include all variables of the output dataset but is still comprehensible.
14 To learn more about the PDV, consult Arthur Xuejun Li’s paper “Essentials of the Program Data Vector (PDV): Directing the Aim to Understanding the DATA Step!” http://support.sas.com/resources/papers/proceedings13/125-2013.pdf
15 Due to limited space, a thorough explanation of the automatic retain mechanisms of the PDV has not been presented. This topic is discussed in the paper “Check your code! Control your data!” https://www.phusewiki.org/docs/Conference%202017%20IS%20Papers/IS07.pdf