PhUSE 2017

Paper IS02

Completing the Data set Jigsaw: Hints and Tips for Assembling Data sets Together

Cerys Rand, PRA Health Sciences, Swansea, United Kingdom

ABSTRACT

Most, if not all, programmers have been in the situation where they have completed the edge of the jigsaw, separated all the pieces into colour piles and are left with one job. The job of joining the pieces together in order to solve the puzzle that is how to combine multiple data sets together. So how do you go about piecing together all of your data sets to complete the jigsaw? What code do you use, and which study documents are vital for assisting you with making your decisions? This paper will offer insight into methods of assembling data sets together, highlighting the differences and similarities between certain SAS® statements by using easy to follow examples and steps, as well as describing how study design and source data set design impacts on the choices you make.

INTRODUCTION

The aim of this paper is to explore the various approaches that are available to accumulate data sets together and observe how using these various SAS® statements can result in the same as well as different output data sets. The objective of looking at each SAS® statement is to fully understand the similarities and differences between each SAS® statement in order to choose the most optimal SAS® statement for a given task. This will be achieved in several stages; firstly by considering study design, then by viewing source data sets and lastly by looking at SAS® statements. Each sponsor has their own unique study set up and often even studies belonging to the same sponsors will have a different study set up. This results in very similar but still different study documents and can result in the programmer choosing to combine data in a different way to how the data would have been combined on other similar studies. The more exposure to different studies a programmer has the more efficient a programmer can become in selecting how to bring data together whether it be for SDTM data sets, ADaM data sets or TFL outputs. Once a programmer is presented with a multitude of data it can often be a trial and error exercise in order to choose the correct function. Masses of study documents can be as much of a help as a hindrance and without knowing which documents to consult and/or where to look in each document then too much time can be spent trying to understand what the data is telling you and how to display this information in a compliant form.

Once the programmer has processed and understood the data the next challenge is knowing which SAS® functions to use to achieve this. Having a full understanding of the available functions and being able to use them easily will help a programmer develop data sets much more effortlessly and will also aim to keep a programmer within any time constraints there maybe.

STUDY DOCUMENTS

Once the protocol is available development of the blank Case Report Form (CRF) begins, it is at this point that choices for unsuspecting programmers are already being made without any programmer input. The review of study documents is essential for the lead programmer, it may be assumed that other departments who are responsible for the creation of study documents are aware of programming needs and requirements but this unfortunately is not always true and can cause extra work and/or re-work later on in the programming process when combining data. The way a study is designed and the database is built plays a crucial part in which SAS® statements are essential to constructing data sets. When reviewing study documents such as the protocol, blank CRF, data transfer specifications (DTS) and edit check specifications it is essential for the lead programmer to cross reference as many of these documents as possible in order to identify discrepancies. The design of the blank CRF and evidently the study database dictates to programmers how the data starts its life cycle within data sets. The data has to be developed into SDTM data sets, ADaM data sets and eventually TFLs, all of which will be required to be manipulated and mapped using industry rules and standards. The review of the blank CRF and DTS is vital for programmers to ensure data coming from the database and vendors is compatible with each other. Receiving unsuited data makes the programmer's job of completing the data set jigsaw all that bit harder. A few important points to look for and keep in mind while reviewing study documents is the format that the data will be received in, lengths of variables,

controlled terminology and whether a variable is free text or not. Try and predict how the information and data from the CRF and DTS will look and how it could go wrong, for example if a variable is to be free text opposed to a drop down box is there the potential for spelling mistakes, which ultimately can cause an issue when joining data sets and have a direct impact on the SAS® statement chosen to combine the data.

Once the protocol, annotated Case Report Form (aCRF) and DTS are signed off and stable, the work of the programmer can truly begin. The programmer takes an active role in the development of the aCRF and the data mapping specifications for both SDTM and ADaM data sets. Although programmers are not involved with the actual writing of the statistical analysis plan (SAP) and TFL shells, the programmer's review of the SAP and shells is compulsory to guarantee successful ADaM data set domains.

SOURCE DATA SETS

Input data sets need to be given careful consideration before they can all be summarised as one data set. When combining raw and vendor data to form an SDTM data set, a different route may be required to when multiple SDTM data sets are joined to form an ADaM data set, the same can be said for when TFL output data sets are created from ADaM data sets. The structure that source data arrives in can vary a great deal and this in itself can affect how a programmer will choose to combine data.

The first source of data is the CRF database, this data needs to be combined and reconciled with the data received from external vendors, and often there is an overlap of variables such as visit dates and time points. When joining these two different sources of data, a programmer needs to ensure no data is overwritten, sometimes caused by a SAS® MERGE statement or duplicate records sometimes caused by a SAS® SET statement. Another reason other than the choice of SAS® statement that can cause these issues is dirty data, working with dirty raw data can sometimes add to the confusion of which statement to choose when building SDTM data sets. Even if no external vendor data is required for the SDTM data set commonly the data will originate from multiple CRF data sets which need to be combined into one SDTM data set. For example, SDTM.LB (Lab data) can come from chemistry, hematology and/or urinalysis CRF pages depending on the requirements of a study and as much caution needs to be taken when choosing how to combine these as when looking at CRF and external vendor data. Data from external vendors and the CRF database is often referred to as raw data and it is this data that forms the edge of the jigsaw that is needed to produce TFL outputs. The combination of CRF and vendor data along with industry standards are usually the standard blocks a programmer will use to formulate SDTM data sets. Once a programmer has chosen which functions to turn raw data into SDTM data the next step is to produce ADaM data sets.

ADaM data sets, similar to SDTM data sets have to be built using industry standards but rather than using raw data ADaM data set are built with SDTM data sets. The function chosen to combine the SDTM data sets needs to ensure that there is traceability from the ADaM data sets back to the SDTM data sets and inevitably back to the raw data sets. ADaM data sets need to contain the variables required for TFLs outputs to be produced and in order to guarantee this, derivation, often complex, is required within the ADaM data sets. A programmer must ensure that the SDTM data sets are brought together using functions that insure the necessary derivation can be implemented.

SAS® FUNCTIONS

SAS® statements: One of the main challenges faced by programmers on a daily basis is knowing which route to take in order to complete the data set jigsaw using the quickest and efficient method. There are several options available to a programmer allowing them to combine data sets to form an overall data set. The ones this paper will be focusing on are; SET, MERGE, MODIFY and UPDATE statements. By using dummy data sets as examples to demonstrate how each SAS® statements operates and groups together data.

SET: One method of combining data is through use of the SET statement. The SET statement can be used in a number of different ways for example, on its own to concatenate data sets or with a BY statement to merge data sets. The way in which SAS® processes the data using the SET statement alone is to concatenate the data presented in the data sets together.

Example 1,

Input data sets:

Data set A:

	subject	var_a
1	001	1
2	002	2
3	003	3

Data set B:

	subject	var_b
1	001	1
2	002	2
3	003	3

Example code:

```
data ab;
  set a b;
run;
```

Data set AB:

	subject	var_a	var_b
1	001	1	.
2	002	2	.
3	003	3	.
4	001	.	1
5	002	.	2
6	003	.	3

All of the records from Data set A are read into Data set AB followed by all of the records from Data set B. Unless told using a DROP or KEEP statement the new data set, in this case Data set AB, will contain every variable from Data set A and Data set B alongside each other. Data set A and Data set B have a common variable called SUBJECT, when set together this variable is retained in Data set AB but will only appear once with the records from Data set A coming out on top of the records from Data set B. Any variable with the same variable name, such as SUBJECT, will be required to have the same variable type, either numeric or character, SUBJECT in Data set A cannot be character and then be numeric in in Data set B if the two data sets are to be set together. It is also possible to use the SET statement with the same input data set more than once.

The SET statement can also be used to process the merging of data this can be done to combine more than one data set or the same data set more than once but sub-setting for different variables.

Example 2,

Input data sets:

Data set A2:

	subject	var_a	var_b
1	001	1	1
2	003	.	3

Data set B2:

	subject	var_a
1	001	1
2	002	2
3	005	3

Example code:

```
data ab2;
  set a2;
  set b2;
run;
```

Data set AB2:

	subject	var_a	var_b
1	001	1	1
2	002	2	3

By using the SET statement on each input data set SAS® creates a one-to-one merge of the data sets based around the common variable SUBJECT. Data set AB2 only contains the same number of records as the smallest input data

set, so in the case data set AB2 contains two records as this is the number of records in data set B2. One thing to watch out for when using this method of merging data is the potential to overwrite variables, as there is no BY statement any common variables in input data sets will be overwritten by the last input data set, hence why in Data set AB2 SUBJECT = 003 now has VAR_A= 2 where in the input data sets, A2, SUBJECT 003 had VAR_A as missing.
The SET statement can also be used in conjunction with a BY statement in order to concatenate data. The BY statement relies on a common variable in the input data sets, in example 3 below this is the variable, SUBJECT.

Example 3,

Input data sets:

Data set A3:

	subject	var_a
1	001	1
2	002	2
3	002	3
4	005	3

Data set B3:

	subject	var_b
1	001	1
2	002	2
3	003	1
4	003	3

Example code:

```
data ab3;
   set a3 b3;
   by SUBJECT;
run;
```

Data set AB3:

	subject	var_a	var_b
1	001	1	.
2	001	.	1
3	002	2	.
4	002	3	.
5	002	.	2
6	003	.	1
7	003	.	3
8	005	3	.

Before using a BY statement with a SET statement SAS® requires both all input data sets to be sorted by the variables in the BY statement. SAS® processes the input data sets by reading in each observation and variable from each input data set. The output data set will contain all of the variables from the inputs data sets and the total number of observations from the input data sets, ordered by the variable in the BY statement.
The SET statement primarily combines data based on the order of the observations within the input data sets and for the examples above this method works fine but often data is needed to be combined to form one data set from two or more input data sets based on common values or variables within the input data sets. This is where the MERGE statement can be a better choice than the SET and BY statement combination.

MERGE: The MERGE statement can be used to combine multiple data sets together to form one data set in a variety of ways. When using the MERGE statement it is important to ensure that the input data sets are sorted in the order of the variables stated in the BY statement before merging them else SAS® will produce an ERROR.

Example 4,

The variables stated in the BY statement are used by SAS® to process which variables should be used to match observations together from the input data sets to construct a new output data set.

Input data sets:

Data set MA4:

	subject	var_a
1	001	1
2	002	2
3	003	3
4	004	4

Data set MB4:

	subject	var_b
1	001	1
2	002	2
3	003	3
4	004	1
5	005	3

Example code:

```
data mab4;
  merge ma4 mb4;
  by SUBJECT;
run;
```

Data set MAB4:

	subject	var_a	var_b
1	001	1	1
2	002	2	2
3	003	3	3
4	004	4	1
5	005	.	3

The output data set, MAB4, contains records from all input data sets maintaining all variables. Where a variable or record is in one but not the other input data set SAS® sets this to be a blank value, as is done above for SUBJECT 005, variable VAR_A.

In example 1, the input data sets only contain mutual variables stated in the BY statement, when using the MERGE statement careful consideration needs to be taken in order not to overwrite values, this can occur where input data sets both contain variables not specified in the BY statement.

Example 5,

Input data sets:

Data set MA5:

	subject	var_a
1	001	1
2	002	2
3	003	3
4	004	4

Data set MB5:

	subject	var_a	var_b
1	001	1	1
2	002	1	2
3	003	1	3
4	004	1	1
5	005	1	3

Example code:

```
data mab5;
  merge ma5 mb5;
  by SUBJECT;
run;
```

Data set MAB5:

	subject	var_a	var_b
1	001	1	1
2	002	1	2
3	003	1	3
4	004	1	1
5	005	1	3

The input data sets MA5 and MB5 both contain the variables SUBJECT and VAR_A but only SUBJECT is in the BY statement when merging them together, therefore VAR_A from MA5 is being overwritten by VAR_A from MB5 as MB5 is the second input data set SAS® looks at when merging the two together, hence VAR_A in MAB5 has the values from MB5. A way to avoid this is to include all mutual variables in the BY statement or if the mutual variables are not required in the BY statement it is also possible to rename the variables when merging the data sets.

Example 6,

Input data sets:

Data set MA6:

	subject	var_a
1	001	1
2	002	2
3	003	3
4	004	4

Data set MB6:

	subject	var_a	var_b
1	001	1	1
2	002	1	2
3	003	1	3
4	004	1	1
5	005	1	3

Example code:

```
data mab6;
    merge ma6 mb6 (rename = (VAR_A=VAR_B1));
    by SUBJECT;
run;
```

Data set MAB6:

	subject	var_a	var_b1	var_b
1	001	1	1	1
2	002	2	1	2
3	003	3	1	3
4	004	4	1	1
5	005	.	1	3

By renaming VAR_A from the input data set, MB6, when SAS® processes the merging of the two input data sets VAR_A is not overwritten in MA6 but instead a new variable is created. Therefore MAB6 now have 4 variable displaying the values of both VAR_A from both input data sets.
The SET and MERGE statements combine data in different ways.

Example 7,

Input data sets:

Data set MA7:

	subject	var_a
1	001	1
2	002	2
3	002	3
4	003	3

Data set MB7:

	subject	var_a
1	001	1
2	002	2
3	003	1
4	003	3

Example code:

```
data mab71;
    merge ma7 mb7
    by SUBJECT;
run;
```

Data set MAB71:

	subject	var_a
1	001	1
2	002	2
3	002	3
4	003	1
5	003	3

The MERGE statement, provided the data being merged is one-to-one, affixes data in a horizontal fashion as can be seen in data set MAB71 but when the same two input data sets are integrated together with a SET and BY statement, they are joined in a vertical manner as shown below in MAB72.

Example code:

```
data mab72;
    set ma7 mb7
    by SUBJECT;
run;
```

Data set MAB72:

	subject	var_a
1	001	1
2	001	1
3	002	2
4	002	3
5	002	2
6	003	3
7	003	1
8	003	3

Both the SET and MERGE statement are popular statements used to combine multiple input data sets together, both having their own strengths and weaknesses as can be seen from the examples above. Another method that can be used in a similar way to the SET statement is the UPDATE statement.

UPDATE: The UPDATE statement can only be used with a maximum of two input data sets. The UPDATE statement is useful when it is required to replace values from one data set, often known as the master input data set, with values from another secondary input data set, usually known as the transaction data set. When using the UPDATE statement SAS® requires there to be a BY statement used directly after the UPDATE statement and it is vital that

both input data sets are sorted by the variables in the BY statement. Within the master input data set there must be a unique value of the variables in the BY statement, else only the first value will be updated.

Example 8,

Input data sets:

Data set UA8 – Master data set:

	name	city	country	subject	siteid
1	Ericson, Jane	Seattle	USA	0001	1
2	Dix, Martin	London	UK	0002	1
3	Gabrielli, Theresa	Paris	FRANCE	0003	2
4	Clayton, Aria	San Francisco	USA	0004	2

Data set UB8 – transaction data set:

	name	city	country	subject	siteid
1	Ericson, Jane	Seattle	USA	0001	1
2	Dix, Martin			0002	1
3	Gabrielli, Theresa	Paris	FRANCE	0003	2
4	Clayton, Aria	San Francisco	USA	0004	2
5	Archuleta, Ruby	London	UK	0005	3

Example code:

```
data uab8;
 update ua8 ub8;
 by SUBJECT name;
run;
```

Data set UAB8:

	name	city	country	subject	siteid
1	Ericson, Jane	Seattle	USA	0001	1
2	Dix, Martin	London	UK	0002	1
3	Gabrielli, Theresa	Paris	FRANCE	0003	2
4	Clayton, Aria	San Francisco	USA	0004	2
5	Archuleta, Ruby	London	UK	0005	3

It is possible to create new observations within the master data set using the UPDATE statement. The master data set, UA8 has only four observations but when the UPDATE statement is used with UB8 as the transaction data set the master data set is updated to contain six observations due to SUBJECT 0002 not having unique BY variables in both data sets and therefore creating two observations.

The MERGE and UPDATE statement handle updating observations differently, the UPDATE statement changes the values of the unique observations in the master data set by applying the values from the transaction data set whereas the MERGE statement will replace existing values in the initial input data set with values from the second, third or any other input data set. With the MERGE statement should there be any missing values in the second input data set these values in the primary input data set will be replaced with these. The UPDATE statement will not replace missing values from the transaction date into the master data set unless the default options of the UPDATE statement are changed. If the same input data sets, UA8 and UB8, and the same BY statements used in example 8 were used with a MERGE statement there would be a different output data set.

Example 8 - UPDATE

Obs	name	city	country	subject	siteid
1	Ericson, Jane	Seattle	USA	0001	1
2	Dix, Martin	London	UK	0002	1
3	Gabrielli, Theresa	Paris	FRANCE	0003	2
4	Clayton, Aria	San Francisco	USA	0004	2
5	Archuleta, Ruby	London	UK	0005	3

Page Break

Example 8 - MERGE

Obs	name	city	country	subject	siteid
1	Ericson, Jane	Seattle	USA	0001	1
2	Dix, Martin			0002	1
3	Gabrielli, Theresa	Paris	FRANCE	0003	2
4	Clayton, Aria	San Francisco	USA	0004	2
5	Archuleta, Ruby	London	UK	0005	3

SUBJECT 0002, has missing values in the transaction data set, therefore, when they are MERGED on they are missing in the output data set but with the UPDATE data set the values from the master data set are maintained. The SET, MERGE and UPDATE statements are all used to create a new copy of the main input data set where the main input data set can be changed by adding or overwriting variables or creating new variables from the other input data sets. An output data set is not always required and sometimes all that is needed is for one of the input data sets to simply be modified in some way using another input data set.

MODIFY: The MODIFY statement updates observations of the existing data set but the consequence of this is that no updates can be carried out to the structure of that data set. The main purpose of the MODIFY statement is to replace all or part of an observation from one data set with an observation from another.

The MODIFY statement, unlike the MERGE and UPDATE does not require the input data sets to be sorted by the BY statement variables beforehand. The MODIFY statement also allows for there to be duplicate values of the BY variables in both the master and transaction data set unlike the UPDATE statement.

Example 9,

Input data sets:

Data set MOA9:

	subject	var_a	var_b
1	001	1	1
2	002	2	2
3	003	3	3
4	004	4	4

Data set MOB9:

	subject	var_a	var_b
1	002	1	5
2	003	1	6
3	004	1	7

Example code:

```
data moa9;
  modify moa9 mob9;
  by SUBJECT;
run;
```

Data set MOA9:

	subject	var_a	var_b
1	001	1	1
2	002	1	5
3	003	1	6
4	004	1	7

As the MODIFY statement does not create a new output data set but rather update the master input data set, MOA9, the MODIFY statement is restricted and the information in the master data set cannot be changed, variables in the master data set are not able to be added, delete or have attributes changed. It is not possible to have observations in MOB9 that are not already in MOA9.

The following table from the SAS® website (Comparing Modifying, Merging, and Updating Data Sets, n.d.) show a brief comparison between the MERGE, UPDATE and MODIFY statements.

Comparison	MERGE	UPDATE	MODIFY
Is sorting of data sets required?	Merge with By statement: Required One on one merge: Not required	Required	Not required
Requirement of Unique By values	Not required	Master data set: Required Transaction data set: Not required	Not required
Can variables be created or deleted	Can be created	Can be created	Cannot be created
Number of data sets that can be combined	Unlimited	Two	Two
Missing value handling	Non-missing values are overwritten	If there are missing values in transaction data set the values in the maser are not updated	If UPDATEMODE option is MISSINGCHECK then does not overwrite non-missing else overwrite. Default: MISSINGCHECK

SUMMARY

As can be seen in the examples above often careful consideration will need to be given in order to not overwrite data and therefore data become lost in the combination process. Although the UPDATE and MODIFY statements have their advantages, they can only be used to combine two data sets which when combining numerous raw data sets, therefore are not necessarily the best statement of choice where neither the MERGE nor the SET statement are not restricted by the number of data sets they can combine. Each programmer will develop their own preferred way in which to complete the jigsaw that is to assemble data and in several circumstances there is no right or wrong method.

ACKNOWLEDGMENTS

Thank you to Ethan Jenkins and Hedd-Wyn Jones for interesting discussions, reviewing the abstract and paper and proof-reading the paper and sharing ideas.

References

Comparing Modifying, Merging, and Updating Data Sets. (n.d.). Retrieved from http://support.SAS®.com/documentation/cdl/en/basess/58133/HTML/default/viewer.htm#a002645429.htm

SAS® Certification Prep Guide: Base Programming for SAS® 9, Third Edition. (n.d.).

CONTACT INFORMATION

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

Cerys Rand
PRA Health Sciences
Llys Tawe
Kings Road
Swansea
SA1 8PG
Work Phone: +44 (0)1792 525780
Email: randcerys@prahs.com