PharmaSUG 2015 - Paper QT09

Using Meta-data to Identify Unused Variables on Input Data Sets

Keith Hibbetts, Inventiv Health Clinical, Indianapolis, IN

ABSTRACT

Meta-data is a very powerful tool in the creation of clinical trial data. It's commonly used to define data standards, and can also be used to increase the level of automation and efficiency in data transformation. This paper will explore an additional benefit to using meta-data in data transformation: it can be used to programmatically identify any variables in the input datasets that weren't utilized. Identifying these unused variables provides a valuable quality check to help ensure that no important data is being lost in the data transformation.

INTRODUCTION

Improving quality and efficiency in clinical trial data transformation is a process that never ends. One of the most powerful tools in this ongoing effort is meta-data. Using meta-data to define requirements is a very effective way to improve the efficiency in data transformation, as it greatly reduces the amount of time needed to program and validate. Once such a meta-data based system is in place, one additional benefit that it can provide is a macro that will programmatically identify any datasets or variables in the input libraries that were not utilized in the data transformation. This check can be used to verify that no applicable input data was inadvertently left off of the output data. The output of the macro will be an Excel spreadsheet listing all datasets and variables not used in the data transformation.

As with most creations, the inspiration for this macro came out of necessity. Near the completion of a highly complex project, it was discovered that a particular variable was not being populated for one study in a data integration project, but the information was collected for that study. Further research found that the information in question was in the input data, but in a non-standard location. Thus the idea for this macro was born, as it provides a method for systematically identifying this issue earlier in the process.

The code for such a macro will be highly dependent on the structure of the meta-data used to define the data transformations. As a result, this paper will focus on the concepts of the macro instead of the specific code used. For more specific information about the meta-data structure used in this example, see Clinical Trial Data Integration: The Strategy, Benefits, and Logistics of Integrating Across a Compound. The important aspects of the meta-data used in this example are as follows:

1. The example is from a system used for data integration (combining data from multiple studies into one integrated database).
2. The requirements document is an Excel file, with one tab per output data set.
3. Within each dataset's tab, there are columns for each of the input studies. These columns are each in the same structure, and define the transformations necessary for each input study.
4. Within the meta-data for an input study, there are columns which define the input library, input data set, and input variable(s).

SETUP TO CALL THE MACRO

There are many options when building a macro to check for unused input variables/data sets. In this example, the macro is built to be called once output datasets have been generated, and the macro will check the inputs for all datasets in one execution. For the rest of the paper, the macro will be referred to as UNUSED_INPUT. The program calling UNUSED_INPUT will need the following:

1. A libname statement pointing to the location of the output datasets
2. Libname statements for each of the input data libraries. Note, the libnames used should match exactly what was used in the data transformation programs.
3. A libname statement to define where the output file will be stored.

MACRO LOGIC

The macro is built with 4 parameters. These parameters are:

1. The name of the library containing the output datasets. This parameter will be referred to as LIB.
2. The name of the requirements meta-data Excel file. This parameter will be referred to as REQ_XLS
3. The name of the output Excel file. This parameter will be referred to as OUTFILE.
4. A yes/no flag to indicate whether unused datasets from the input library should be listed. This parameter will be referred to as LISTDATASETS. The purpose of this parameter is to allow the macro to produce meaningful output before all output datasets are created. If the UNUSED_INPUT were to be run on just a subset of output datasets, it is likely that the input libraries would contain several data sets that would be flagged as unused (because they will be used in datasets not yet created). In such a situation, setting LISTDATASETS to no allows the output to avoid “false positives”.

FIRST STEP: IDENTIFY THE DATA SETS TO PROCESS

The first step of the macro will be to identify which data sets exist in the output library, and populate macro variables based on that information. This is accomplished by using the VTABLE data set in the SASHELP library. The macro looks for records in VTABLE where LIBNAME = upcase(&LIB).

Once the data sets are identified, a macro variable (NUM) is created which stores the number of such datasets. Later in the macro processing, a series of three nested loops will execute. NUM will be used to define how many times the outermost loop will be executed. Macro variables will also be created to store the names of each data set. These macro variables will be named in the format DM!!_n_. For example, macro variable DM1 will store the name of the first data set, DM2 will store the name of the second dataset, and so on.

For each identified dataset, the corresponding tab from the requirements document (REQ_XLS) will be imported into SAS.

THE NESTED LOOPS

A series of three nested loops process next, and will utilize the requirements meta-data imported into SAS.

Outermost Loop

The outermost loop will process once for each output dataset (using the macro variable NUM to identify the number of iterations).

The first step in the loop will be to call a macro from the suite of tools used in the data transformation (GET_REQS). This macro processes the requirements meta-data imported from SAS, and produces an individual data set of requirements meta data for each input study. GET_REQS stores this metadata in the work library and uses a specific naming convention. That naming convention is as follows: &study._&dataset._req. &study refers to the name of the study, &dataset refers to the name of the output dataset, and the last node of the name is the text req.

The next step in the loop will be to identify each of the requirements meta data datasets created by GET_REQS. This will be accomplished by looking in SASHELP.VTABLE for records where MEMNAME values fit the nomenclature of the requirements data sets for the current dataset being processed (i.e. if looping for the first time, the macro looks for MEMNAME values for the dataset name stored in macro variable DM1). The names of each of these requirements meta-data data sets will be stored in macro variables named MDx (i.e. MD1 will store the first name, MD2 the second name, etc.). The macro variable NUMMD will store the number of such datasets.

After this step, the middle loop processes.

Middle Loop

The middle loop will process once for each requirements meta-data data set identified by the outer loop (using the macro variable NUMMD to identify the number of iterations).

The first step of the middle loop is to look in the requirements meta-data data set and identify the input data sets that it refers to. Within the requirements meta-data, there is a variable which identifies the input library and a variable that identifies the input data set. Each unique combination of values in these two variables represents an input data set. These unique values are stored in a dataset named USED_DATASETS, to record that these input data sets are utilized in the requirements. For each unique value, macro variable SSx is created to store the name of the input library (i.e. SS1 stores the library of the first input data set). Macro variable SDx is created to store the name of the input data set (i.e. SD1 stores the name of the first input data set). The macro variable NUMSDS will store the number of unique input data sets.

After this step, the innermost loop processes.

Innermost Loop

The innermost loop will process once for each input data set identified by the middle loop (using the macro variable NUMSDS to identify the number of iterations).

The first step of the innermost loop is to determine what variables exist on the input data set. This is done by utilizing the VCOLUMN data set in the SASHELP library. Any record from VCOLUMN where libname matches the input library and MEMNAME matches the input data set is used.

The next step is to determine what variables are referred to in the requirements meta-data. As you'll recall from above, the requirements meta-data contains a column for input library, a column for input data set, and a column for input variable(s). So any record matching the input library/data set being processed will be kept. Some records may list more than one input variable. We want a data set that lists each input variable on a separate record, so this loop contains a step which outputs a separate record for each variable. Because it is possible that a variable on the input data set could be used in processing for multiple output variables, this data set is sorted to eliminate duplicate records.

The next step is to merge the two resulting data sets (one containing all variables in the input data set, one containing all variables referred to in the requirements). Any variable contained in the input data set not referred to in the requirements is kept and stored in a working data set. The information kept is the name of the output dataset, input library, input data set name, and variable name.

POST-LOOP PROCESSING

After the three nested loops have executed, we have a dataset that contains all unused input variables. As it's possible that the same variable could have been unused from multiple input data sets, one record is saved for each unique unused variable name (with input libraries/data set values concatenated into one variable). This helps make the output easier to read. To this dataset, we add a null variable named COMMENTS. This will provide a place on the output to enter comments on why a particular variable was not used.

Identifying Unused Input Datasets

If the parameter LISTDATASETS is equal to yes, UNUSED_INPUT will also identify if there are any data sets in the input libraries that were not referred to at all in the requirements. The middle of the three nested loops creates a dataset containing all input data sets that were used in the requirements. This dataset is first sorted to remove all duplicates. Next, a macro variable is created storing all of the input libraries used.

The next step is to identify all data sets contained in these input libraries. This is done by utilizing the VTABLE dataset again, this time keeping any record where LIBNAME matches one of the values of the input libraries. The two data sets (one containing the names of the data sets used in requirements, the other containing the names of all data sets in the input libraries) are merged. Only records containing the names of data sets in the input libraries but not in the requirements are kept.

OUTPUT

Utilizing ODS, the data set containing all unused variables from input data sets is output as an Excel spreadsheet. If LISTDATASETS is set to yes, then the data set containing all unused data sets from the input libraries is also included in the Excel file. An example output file is below. This example is based on test data, and is shortened for display purposes. In this test scenario, two of the input studies contained medDRA-related variables in AE that were not used in the output data because the values were read in from the medDRA dictionary instead. One of the input studies erroneously contained the variable CO.COVAL1 which had null values in every record. One of the input libraries contained a data set named TEST which was not used in any output. The comments entered into the table were entered by the user on the output Excel file.

Input variables not used in requirements		
domain=AE		
input_variable	sources	comments
AEBDSYCD	ABCD.AE, EFGH.AE	MedDRA terms were looked up in the medDRA dictionary instead of taken from study data
AEBODSYS	ABCD.AE, EFGH.AE	MedDRA terms were looked up in the medDRA dictionary instead of taken from study data
AEDECOD	ABCD.AE, EFGH.AE	MedDRA terms were looked up in the medDRA dictionary instead of taken from study data
domain=CO		
input_variable	sources	comments
COVAL1	ABCD.CO	COVAL1 was null for all records in input data set, all comments in integration were <= 200 characters so COVAL1 was not included.
Datasets Not Used in Mappings		
library	dataset	
ABCD	TEST	

Table 1. Output Example

CONCLUSION

Meta-data is an extremely powerful tool in data transformation. This paper has outlined one of the many benefits, which is a programmatic method to identify variables and data sets from the input libraries that were not utilized. This provides an extra layer of quality assurance in your data transformation needs.

REFERENCES

Reynolds, Natalie and Hibbetts, Keith. 2014. “Clinical Trial Data Integration: The Strategy, Benefits, and Logistics of Integrating Across a Compound.” Proceedings of the PharmaSUG 2014 Conference. Available at http://www.pharmasug.org/proceedings/2014/AD/PharmaSUG-2014-AD21.pdf