Paper 158-2019

Automating ADSL Programming Using Pinnacle 21® Specifications

Tracy Sherman, Ephicacy Consulting Group Inc.;
Aakar Shah, Nektar Therapeutics

ABSTRACT

Standardization is the most basic pre-requisite to automation, and Pinnacle 21® ADaM Define specifications offer a useful and powerful standard for ADaM documentation. As adoption of Pinnacle 21 ADaM Define specifications increases, so does the opportunity for automation in the creation of ADaM datasets. Pinnacle 21 specifications provide the necessary metadata (variable data source, origin type, assignments, codelists and derivations) for generating SAS® code and code templates to create ADaM datasets.

This paper will demonstrate how Pinnacle 21 specifications for ADSL can be used to dynamically generate a program for producing an ADSL dataset. Specification guidelines based on origin type will be proposed to help streamline the specification writing process and increase the scope and sophistication possible in generated code.

INTRODUCTION

We strongly encourage that Pinnacle 21 specifications should be treated as the first step in dataset programming, not the last (Shah and Sherman, 2018). If we start programming with the 'define in mind', this eliminates or at least reduces re-work when it comes to define specification writing. It also provides for the ability to validate on an ongoing basis, for instance when new SDTM data is received or when statisticians want to review dataset specifications. If the define is available throughout the programming process, reviewers can simultaneously view the specifications and hyperlinked data to get a sense of the data quality through the use of the Pinnacle 21 validator. Too often, CDISC validation issues in ADaM are the result of skipping the SDTM CDISC validation step. If SDTM define specifications are used for programming purposes this eliminates the need for separate programming specifications. Validation issues can be caught at the beginning rather than at crunch time when ADaM and TLF programming have already been started and/or completed.

As with SDTM define specifications, which should be written prior to SDTM programming, ADaM define specifications should similarly be the first step in ADaM programming. These specifications can be used to create the required define.xml for early review and validation, whilst also providing the basis for automation in ADaM programming.

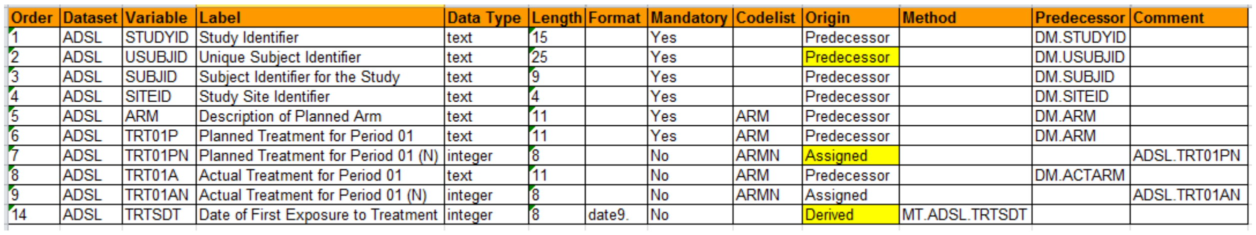
This paper will demonstrate how SAS® code for an ADSL dataset can be generated from the Datasets, Variables, Codelists, Methods and Comments worksheets in the ADaM Pinnacle 21 define specifications. The Origin column has three main entries (Predecessor, Assigned, and Derived) along with the less common entries such as eDT (electronic data transfer), Protocol and <blank>. These different origin types can be used to create generic SAS code using metadata stored in the specifications. Guidelines will be proposed for each origin type to streamline the specification writing process and for ensuring consistent SAS code is generated for each variable.

ADSL PINNACLE 21 SPECIFICATIONS

You can download the Pinnacle 21 Community software for no cost at https://www.pinnacle21.com/downloads. If you do not have a copy of the Define specification spreadsheet used by Pinnacle 21, you could easily retrieve it from the community version software by uploading any ADaM dataset. For more details on creating the specifications from source data please refer to https://www.pinnacle21.com/projects/using-opencdisc-community.

The ADaM Define specification workbook contains 13 worksheets: Study, Datasets, Variables, ValueLevel, WhereClauses, Codelists, Dictionaries, Methods, Comments, Documents, Analysis Displays, Analysis Results, and Analysis Criteria. We will focus on the Datasets, Variables, Codelists, Methods and Comments worksheets for this paper.

Starting with the Variables worksheet (see Figure 1), the Origin column has three main entries (Predecessor, Assigned, and Derived) for ADaM specifications along with less common ones such as eDT (electronic data transfer), Protocol and <blank>. Efficiencies for specification writing are gained if the Origin column on the Variables worksheet is completed first, followed by entering the content on the Methods worksheet if the variable is 'Derived' and the Comments worksheet if the variable is 'Assigned'. If Origin='Predecessor', the Predecessor column should be entered.



Order	Dataset	Variable	Label	Data Type	Length	Format	Mandatory	Codelist	Origin	Method	Predecessor	Comment
1	ADSL	STUDYID	Study Identifier	text	15		Yes		Predecessor		DM.STUDYID	
2	ADSL	USUBJID	Unique Subject Identifier	text	25		Yes		Predecessor		DM.USUBJID	
3	ADSL	SUBJID	Subject Identifier for the Study	text	9		Yes		Predecessor		DM.SUBJID	
4	ADSL	SITEID	Study Site Identifier	text	4		Yes		Predecessor		DM.SITEID	
5	ADSL	ARM	Description of Planned Arm	text	11		Yes	ARM	Predecessor		DM.ARM	
6	ADSL	TRT01P	Planned Treatment for Period 01	text	11		Yes	ARM	Predecessor		DM.ARM	
7	ADSL	TRT01PN	Planned Treatment for Period 01 (N)	integer	8		No	ARMN	Assigned			ADSL.TRT01PN
8	ADSL	TRT01A	Actual Treatment for Period 01	text	11		No	ARM	Predecessor		DM.ACTARM	
9	ADSL	TRT01AN	Actual Treatment for Period 01 (N)	integer	8		No	ARMN	Assigned			ADSL.TRT01AN
14	ADSL	TRTSDT	Date of First Exposure to Treatment	integer	8	date9.	No		Derived	MT.ADSL.TRTSDT		

Figure 1 ADSL Pinnacle 21 specifications highlighting the main origin types

GUIDELINES FOR SPECIFICATIONS BASED ON ORIGIN TYPE

In Table 1, each origin type is described, and specific guidelines are included to complete the specifications. This will help standardize your study specifications and allows dataset programming code to be generated. The more consistent your specifications are written, the easier it is to tailor SAS code.

Origin	Description	Specification Writing Guideline
Predecessor	Data that is copied from a variable in another dataset. For example, predecessor is used to link ADaM data back to SDTM variables or other ADaM variables to establish traceability. If selected, the Define will include text from the Predecessor column.	Predecessor column written in the form of *domain.variable.* Examples: • DM.STUDYID • SUPPDM.QVAL where QNAM="COHORT" • MI.MISTRESC where MITESTCD="ESTRCPT"
Assigned	Data from individual judgment or a third-party adjudicator and does not include subject or investigator data that is collected as part of the CRF, eDT or derived based on an algorithm. Coded terms that are supplied as part of a coding process (as in --DECOD). Most of the coded terms, such as PARAMCD, AVISITN, and DTYPE. Use with the Comment column and corresponding ID column on the Comments worksheet.	Description column on the Comments worksheet: If a numeric variable has a codelist then enter: Numeric code for <*source variable name*> If a character variable has a codelist then enter: Character code for <*source variable name*> If the variable is set to a text string then enter: Set to <*text string*>. If a variable uses a specific format such as ISO3361 then enter: "ISO3166 code for COUNTRY". If the Document column is populated enter: "Column <*column name*> from <*document name*>*".*

Origin	Description	Specification Writing Guideline
Derived	Data that is not directly collected on the CRF or received via eDT but is calculated by an algorithm or reproducible rule defined by the sponsor, which is dependent upon other data values. Use with the Method column on the Variables worksheet.	Add a new column 'SAS Code' to the Methods worksheet. Add SAS code such as: 'floor((rficdt-brthdt+1)/365.25);' or 'if .<aage<65 then agegr2="<65"; else if aage >=65 then agegr2=">=65";' or '%m_bl (in=qs1, out=ecog_bl)'; Add a high-level description to the Description column on the Methods worksheet.
eDT	Data that is received via electronic Data Transfer (eDT). Use with the Comment column and corresponding ID column on the Comments worksheet.	Description column on the Comments worksheet: If the Document column is populated enter: "Column <*column name*> from <*document name*>*"* such as 'Column AUC from pk_parameters'. If applicable, specify the name of the document (e.g. 'pk_parameters) in the Document column on the Comments worksheet.
Protocol	Data which may be specified only in the protocol and not appear on a CRF or transferred via eDT. Use with the Comment and corresponding ID column on Comments worksheet.	Enter details into the Description column on the Comments worksheet.
<blank>	Blank if value level data has more than one origin specified on the value level worksheet. An example would be AVAL/AVALC variables in the ADLB dataset that commonly have both 'Derived' and 'Predecessor' origin types.	ValueLevel worksheet must be completed and have at least two different origin types.

Table 1 Define Specification Guidelines Depending on Origin Type

Predecessor Variables

For variables that originate from a predecessor variable, the Predecessor column should be written in the form of *domain.variable or domain.variable* where *variable = 'xxxxx'*.

Assigned Variables

Numeric and character variables that have a codelist should be written on the Comments worksheet. It should be consistently written as 'Numeric code for <*source variable name*>' and 'Character code for <*source variable name*>', respectively. If a variable uses a specific format such as ISO3361 then text such as: "ISO3166 code for COUNTRY". And lastly, if the Document column is populated enter: "Column <*column name*> from <*document name*>*".*

Derived Variables

Add a new column named 'SAS Code' after the final column on the Methods worksheet. SAS code can be added for the following:

1) simple derivations ('floor((rficdt-brthdt+1)/365.25);') or
2) IF-THEN statements ('if .<aage<65 then agegr2="<65"; else if aage >=65 then agegr2=">=65";') or
3) study specific macro calls such as baseline derivations (%m_bl (in=qs1, out=ecog_bl);).

If the derivation is in a different form than above, the text in the Description column is placed in the dataset program as a comment to be used as a guide for the programmer. For example, in adsl.sas, the method to derive the variable PKFL would be displayed as a comment:

```
data VAR_20;
   * PKFL=Y if SAFFL=Y and subject has non-missing PC.PCORRES. Else N. *;
   keep STUDYID USUBJID PKFL;
run;
```

eDT Variables

If the data originated from an eDT (electronic data transfer) then complete the Description column on the Comments worksheet in the form of: "Column <*column name*> from <*document name*>*"* such as 'Column AUC from pk_parameters'. If the data is taken from a specific column, then enter details into the Description column.

If applicable, specify the name of the document (e.g. 'pk_parameters') in the Document column on the Comments worksheet. Ensure the document is also entered onto the Documents worksheet to prevent define validation findings.

Protocol Variables

Generally, origin='Protocol' is populated for variables in the SDTM define specifications although there may be cases where Protocol is required for ADaM. In the Description column on the Comments worksheet enter the text string that the variable was set to (e.g. Set to 'EVALUATOR').

<Blank> Variables

The Origin column can be set to <blank> if the value level data has more than one origin specified on the ValueLevel worksheet. An example would be AVAL/AVALC variables in the ADLB dataset that commonly have both 'Derived' and 'Predecessor' origin types.

PROCESS FLOW

Now that we have guidelines to complete the define specifications, the specifications are ready to be used to generate a dataset program. This section will describe the process involved.

As shown in Figure 2, three macros are used to generate the adsl.sas code.

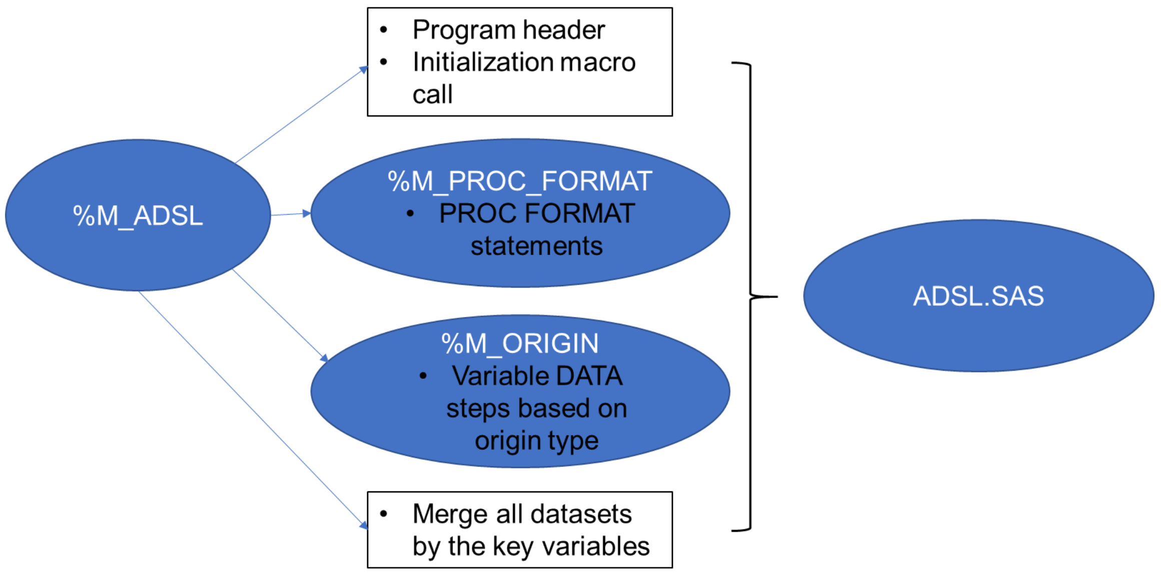

• Program header
• Initialization macro call
%M_ADSL
%M_PROC_FORMAT
• PROC FORMAT statements
%M_ORIGIN
• Variable DATA steps based on origin type
• Merge all datasets by the key variables
ADSL.SAS

Figure 2 Process flow to write SAS code to ADSL.sas

The macro, %M_ADSL, is used to call in the proc format macro, %M_PROC_FORMAT, the variable level macro, %M_ORIGIN, and output the SAS code to adsl.sas. Full code is available upon request.

The main steps are as follows:

1. Create the SAS program header and add the %include initlnx.sas statement. Initlnx.sas is the initialization program used to assign libnames and SAS options.
2. Call the utility, %M_PROC_FORMAT, that creates the PROC FORMATs based on the Pinnacle 21 specification Codelists worksheet. This will write the PROC FORMAT statements after the initialization call.
3. Merge all the SDTM supplemental domains with the parent domains for instant access to all the available SDTM data.
4. Call in the macro, %M_ORIGIN, to output DATA step code for each variable depending on origin type.
5. Merge all datasets together by the key variables taken from the Datasets worksheet. For ADSL, the key variables are STUDYID, USUBJID.
6. Add the macro call for assigning variable attributes such as variable labels and lengths.

GENERATE PROC FORMAT SAS CODE FROM PINNACLE 21 SPECIFICATIONS

The macro, %M_PROC_FORMAT, generates PROC FORMAT code from the Codelists worksheet. The high-level steps in the code are outlined in the steps below. Full code is available upon request.

1. Bring in Pinnacle 21 ADaM specifications for ADSL
2. Using the codelist metadata from the Codelists worksheet, create a text variable that concatenates the 'Decoded Value' and the 'Order' column for codelists that have a Data Type that is either 'integer' or 'float'. If the codelist has a Data Type = 'text' then concatenate the 'Decoded Value' with the 'Term' column. This variable will take the form of a VALUE or INVALUE PROC FORMAT statement.
3. Using the Variables worksheet, determine which ADSL variables have a codelist present in the 'Codelist' column.
4. Output PROC FORMAT code using a DATA _NULL_ with a FILE statement using the MOD option. SAS statements are written using PUT statements with the appropriate VALUE or INVALUE code depending on the data type.

GENERATE VARIABLE LEVEL SAS CODE FROM PINNACLE 21 SPECIFICATIONS

The macro, %M_ORIGIN, generates variable level SAS code depending on origin. The high-level steps in the code are outlined in the steps below: See APPENDIX 1 for snippets of code and descriptions.

1. Bring in Pinnacle 21 ADaM specifications for ADSL
2. Create macro variable, &KEYVAR, from key variables listed on the Datasets worksheet in the define specifications. For ADSL, the key variables are STUDYID and USUBJID.
3. If Origin='Predecessor' on the Variables worksheet and the source data is available then output specific DATA step SAS code to adsl.sas to bring in the source variable and name it according to the specifications. Merge all predecessor datasets by &KEYVAR into a dataset called 'predecessor'.
4. If Origin='Assigned' and source data is available then output specific DATA step SAS code to adsl.sas to bring in the source variable and assign values. Use the predecessor dataset created in Step 3 as the input to all assigned variables. Merge all variable level datasets by KEYVAR into a dataset name 'assigned'.

5. If Origin='Derived', 'eDT' or <blank> then add specific DATA step code from the Methods worksheet or add the high-level description as a SAS comment. Use the assigned dataset created in Step 4 as input for the derived variables. Merge all variable level datasets by KEYVAR into a dataset named 'derived'.
6. Merge 'predecessor', 'assigned' and 'derived' intermediate datasets by &KEYVAR.

ADSL.SAS

The utilities described above (%M_ORIGIN, %M_PROC_FORMAT and %M_ADSL) are used together to generate the program for adsl.sas. Output 1 shows an abbreviated sample of the generated ADSL code.

The program header and initialization macro call are written first, then the PROC FORMAT code, followed by merging the SDTM supplemental domains with the parent domains. The data step for each variable is written in the order of origin type and the Order column that is specified on the Variables worksheet. In Output 1, the dataset VAR_6 is created from a predecessor variable with order=6; VAR_11 from an assigned variable with order=11; and VAR_27 is a derived variable with order=27.

```
/*******************************************************************
*** Program: adsl.sas
*** Programmer: tsherman
*** Date: 05JUN19
*** Description: Analysis Dataset Subject Level
*******************************************************************/
%inc “&inpath/tools/initlnx.sas”;

proc format;
   invalue AGEGR1N
    '<50'=1
    '50 to <65'=2
    '>=65'=3;
run;
…etc.
data VAR_6;
   set DM;
   rename ARM=TRT01P;
   keep STUDYID USUBJID ARM;
run;
…etc.
data VAR_11;
   set predecessor;
   COHORTN = input(COHORT,COHORTN.);
   keep STUDYID USUBJID COHORTN;
run;
…etc.
data VAR_27;
   set assigned;
   AAGE = floor((rficdt-brthdt+1)/365.25);
   keep STUDYID USUBJID AAGE;
run;
…etc.
** Merge all predecessor, assigned and derived datasets **;
data final;
   merge predecessor assigned derived;
   by &keyvar;
run;
```

Output 1. Sample adsl.sas code generated from Pinnacle 21 ADaM specifications

CONCLUSION

Through the use of Pinnacle 21 ADaM define specifications, we have demonstrated that it is possible to produce a SAS program, adsl.sas to generate the ADSL dataset. The Origin column from the Variables worksheet can be used to design generic DATA step code for each variable. PROC FORMAT code can be written by extracting metadata from the Codelists worksheet, derivations can be taken from the Methods worksheet and variables that are Assigned use metadata from the Comments worksheet.

Similar logic can be applied to other SDTM and ADaM datasets as well. For SDTM, slight modifications would be required to the macro code to remove the merging of supplemental domains with the parent domains.

Specification writing guidelines for each origin type are presented to enable SAS code to be dynamically generated. Having these guidelines streamlines the define specification writing process and ensures a high-quality electronic submission to regulatory authorities.

We hope that this paper encourages users to program with the ‘define in mind’ and to transition the completion of the define specifications from an electronic submission to a programming kick-off activity.

REFERENCES

Shah, Aakar and Tracy Sherman, 2018. Doctor's ‘Prescription’ to Re-engineer Process of Pinnacle 21 Community Version Friendly ADaM Development. PharmaSUG 2019 Conference Proceedings. Paper DS-15.

ACKNOWLEDGMENTS

We would like to thank Ganesh Gopal from Ephicacy Consulting Group Inc. and Susan Zhao from Nektar Therapeutics for their continued support and encouragement in conference attendance, as well as all our family, friends and colleagues. A special thank-you to Brian Fairfield-Carter for his review and suggestions.

CONTACT INFORMATION

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

Name: Tracy Sherman
Enterprise: Ephicacy Consulting Group, Inc.
E-mail: shermantracy@gmail.com

Name: Aakar Shah
Enterprise: Nektar Therapeutics
E-mail: AShah@nektar.com

APPENDIX 1

STEP 1.

The Pinnacle 21 specifications for ADSL are read in with the LIBNAME statement using the XLSX option:

```
libname specs xlsx "pathname/studyname_adam.xlsx";
```

STEP 2.

The list of key variables for ADSL is pulled from the Datasets worksheet in the Pinnacle 21 ADaM defines specifications. The idea here is that the code can be used for other ADaM dataset programming in addition to ADSL. The key variables for ADSL are STUDYID and USUBJID.

```
data keyvar;
   set specs.datasets (where=(lowcase(dataset)="&ds"));
   keyvar=tranwrd(key_variables,',','');
run;

proc sql noprint;
  select keyvar
  into :keyvar separated by ''
  from keyvar;
quit;
%put &keyvar;
```

STEP 3.

If Origin=’Predecessor’ and source data is available then output specific DATA step SAS code to adsl.sas. This section of the macro, brings in the source variable and determines if a where clause was used in the predecessor column from the specifications.

```
data ds1;
   length var whr $200;
   set specs.variables
     (where=(lowcase(dataset)="&ds" and origin="Predecessor")); ❶
   ds=scan(predecessor,1,'.');
   out='VAR'||'_'||strip(put(order,8.)); ❷

   if index(predecessor,'where')=0 or index(ds,'SUPP') then do;
      if index(ds,'SUPP')=0 then var=strip(scan(predecessor,2,'.')); ❸
      else var=strip(scan(predecessor,2,'"')); ❹
      ds=strip(compress(tranwrd(ds,'SUPP',''))); ❺
   whr='';
end;
else do; ❻
   var=scan(predecessor,1,'where');
   var=strip(scan(var,2,'.'));
   type='wheres';
   whr=strip(scan(predecessor,2,'where'));
end;
run;
```

❶ Select ADSL specifications that have Origin=’Predecessor’ on the Variables worksheet from the Pinnacle 21 specifications.

❷ Create output dataset names named VARx where x equals the order variable from the Variables worksheet.

❸ For variables originating from a supplemental domain or those that do not contain a ‘where’ statement, VAR will equal the predecessor variable name after the ‘.’.

❹ Variables from a SUPP domain, use the name of variable (need domains created above with SUPP already merged in).

❺ Create a dataset name

❻ Predecessor variables that include a where clause other than from a SUPP domain (e.g. DS.DSDECOD where DSSPID="CRF: END OF TREATMENT"). VAR equals the variable name after the domain name. A new variable, WHR, is declared as the text after the word 'where'.

In the following DATA _NULL_ step, three macro variables are created with , &VARx, &OUTx and &NVARx from the variables created in the preceding DATA step. Variable name is placed in the macro variable &VARx, output dataset name is placed in &OUTx and finally the number of variables is placed in &NVARx.

```
data _null_;
   set ds1 end=eof;
   call symput("VAR"||strip(put(_n_, 8.)),strip(variable));
   call symput("OUT"||strip(put(_n_, 8.)),strip(out));
   if eof then call symput("NVAR", strip(put(_n_,8.)));
run;
%put &nvar;
```

DATA _NULL_ is used with a %DO loop to output SAS code to adsl.sas for each predecessor variable. This section of code makes use of the MOD option in the file statement which allows additional rows to be added to an existing file such as adsl.sas.

```
%do z = 1 %to &nvar;
   data _null_;
      set ds1 (where=(variable="&&var&z")) end=EOF;
      file "&inpath/prod/programs/adam.adsl.sas" mod; ①
      dlm = byte(9); ②
      if _n_=1 then do;
         put " ";
         put "data &&out&z..;"; ③
         put dlm+(-1) "set " ds';' ; ④
      end;
      if type='wheres' then do;
         put dlm+(-1) "where " whr';'; ⑤
      end;
      if var ne variable then do;
         txt=strip(var)||'='||strip(variable);
         put dlm+(-1) "rename " txt';'; ⑥

      end;
      put dlm+(-1) "keep &keyvar " var";"; ⑦
      if EOF then do;
         put 'run;'; ⑧
      end;
   run;
%end;
```

① Using the FILE statement with the MOD option, records are written to adsl.sas.

② The BYTE function with the number 9 returns the appropriate space added for indentation of SAS code

③ PUT statement used to write the data statement with the macro variable &&OUT&Z that resolves to the output dataset name

④ Add the SET statement with the applicable dataset name contained in the DS variable

⑤ If a WHERE statement exists in the specifications, concatenate ‘where’ with the variable WHR that contains the where text string

⑥ Add a RENAME statement if the predecessor variable requires to be renamed

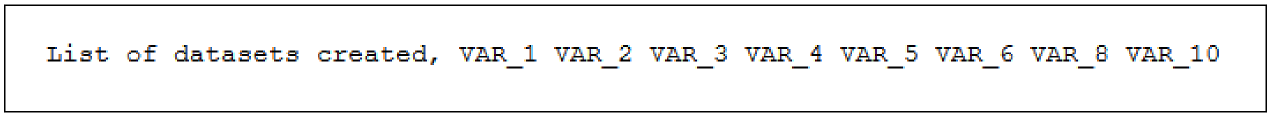
⑦ PUT statement used to write a KEEP statement using the KEYVAR macro variable in STEP 2.

⑧ If end of file (EOF), then write a ‘run;’ statement

The macro variable OUTNAM contains a concatenated list of all datasets created above.

```
%Local outnam;
data _null_;
   length outnam $200;
   retain outnam ' ' ;
   set ds1 end=eof;
   outnam = trim(left(outnam))||''||left(out);
   if eof;
   call symput('outnam',strip(outnam));
   keep ds outnam type whr var variable order out;
run;
%put List of datasets created, &outnam;
```

Output 1 The log might show:



```
List of datasets created, VAR_1 VAR_2 VAR_3 VAR_4 VAR_5 VAR_6 VAR_8 VAR_10
```

Output 2. Output from %put &outnam

```
data _null_;
   file "&inpath/prod/programs/adam/&ds..sas" mod;
   dlm = byte(9);
   put " ";
   put "*** Merge all predecessor data sets by key variables ***;";
   put "data predecessor;";
   put dlm+(-1) "merge DM &outnam;" ;
   put dlm+(-1) "by &keyvar;" ;
   put "run;" ;
run;
```

STEP 4.

If Origin=’Assigned’. Similar code to Step 3 with different DATA _NULL_ put statements:

```
data _null_;
   set assign1 (where=(variable="&&var&z")) end=EOF;
   file "&inpath/prod/programs/adam/&ds..sas" mod;
   dlm = byte(9);
   if _n_=1 then do;
      put " ";
      put "data &&out&z..;";
      put dlm+(-1) 'set predecessor;';
   end;
```

```
      if type='format' or type='set to' then do;
         put dlm+(-1) variable '= ' statement';';
         put dlm+(-1) "keep &keyvar " variable";";
      end;
      if type='ISO' then do;
         put dlm+(-1) variable '= ' statement';';
         put dlm+(-1) "keep &keyvar " variable";";
      end;

** If no codelist then output the description in the specifications **;
      if type not in ('format','set to','ISO') then do;
         put dlm+(-1) '*** ' variable '= ' description' ***;';
         put dlm+(-1) "keep &keyvar " variable";";
      end;

      if EOF then do;
         put 'run;';
      end;
   run;
```

STEP 5.

If Origin='Derived', 'eDT' or <blank>. Similar code to Step 3 with different DATA _NULL_ PUT statements based on the derivations from the Methods worksheet.

```
   data _null_;
      set derive_1 (where=(variable="&&var&z")) end=EOF;
      file "&inpath/prod/programs/adam/&ds..sas" mod;
      dlm = byte(9);
      put " ";
      if type='OTHER' then do;
         put "data &&out&z..;";
         put dlm+(-1) "set assigned;" ;
         put dlm+(-1) variable '= ' sas_code;
      end;
      else if type='IF THEN' then do;
         put "data &&out&z..;";
         put dlm+(-1) "set assigned;" ;
         put dlm+(-1) sas_code;
      end;
      else if type='STUDY MACRO' then do;
         put "*** Study specific macro for step below ***;" ;
         put sas_code;
         put " ";
         put "data &&out&z..;";
         put dlm+(-1) "*** set dataset from macro call above ***;" ;
      end;
      else if type='USE DESCRIPTION' then do;
         put "data &&out&z..;";
         put dlm+(-1) '*** ' variable '= ' description' ***;';
      end;
      put dlm+(-1) "keep &keyvar " variable";";
      if EOF then do;
         put 'run;';
      end;
   run;
```

STEP 6.

Merge datasets together by Key Variables obtained from Pinnacle 21 specifications Datasets worksheet. For ADSL, the key variables are STUDYID and USUBJID.

```
** Create macro var outnam for all datasets to be included below ***;
   data _null_;
      length outnam outvar1 outvar2 $2000;
      retain outnam ' ' ;
      retain outvar1 ' ' ;
      retain outvar2 ' ' ;
      set derive_1 end=eof;
      outnam = trim(left(outnam))||''||left(out);
      outvar1 = trim(left(outvar1))||''||left(variable);
      outvar2 = trim(left(outvar2))||''||left(variable);
      if eof;
      call symput('outnam',strip(outnam));
      call symput('outvar1',strip(outvar1));
      call symput('outvar2',strip(outvar2));
      keep ds outnam outvar1 outvar2 type whr var variable order out;
   run;

   data _null_;
      file "&inpath/prod/programs/adam/&ds..sas" mod;
      dlm = byte(9);
      put " ";
      put "*** Merge all derived variable data sets by key variables ***;";
      put "data derived;";
      put dlm+(-1) "merge &outnam;" ;
      put dlm+(-1) "by &keyvar;" ;
      put "run;" ;
   run;
```