PharmaSUG 2026 - Paper AP-220

Do One Task, Get Another Done for Free:
Use DEFX Tags in Comments to Fill Out Your define.xml

Brendan Bartley, Harvard T.H. Chan School of Public Health
Megan Hinger, Rho Inc.

ABSTRACT

Have you ever wanted to take care of two tasks while only doing one? If so, this is the commenting system for you. We all know we need to comment our programs, but sometimes are lax about this. I suspect this may happen less often if the comment would save time and effort filling out the define.xml at the end of a project. At our workplace, code review is also a big part of our validation process, so our comments must ensure that the intent of the code matches what is in the code. The DEFX tag system will save time and effort finding programs and writing comments for the define.xml while encouraging the good coding practice of commenting one's code.

INTRODUCTION

So, how does this work? It starts by giving a syntax to our DEFX tags. Then we have a program, aptly named DEFX, that collects these tags in all of our data creation programs. We work on a Linux system, so we use the power of the GREP command within a FILENAME statement that uses the PIPE option to complete this task. Once we have the list of DEFX tags, then the DEFX program creates a dataset with the elements needed to fill out the following tabs in the define.xml (Excel spreadsheet using a .xlsx extension) created by Pinnacle 21: Datasets, Methods, Variables, and Comments. The key is a simple syntax and identification of the DEFX tags, so let's begin there.

DEFX TAGS

What is a DEFX tag? It's simply a comment that starts with "DEFX". We named the tag "DEFX" to reference and abbreviate the define.xml file and not be confused with an actual word. Only after a google search for this paper did I find out it's a cryptocurrency and apparently a DJ, but it's likely a safe tag name in the PHARMA world. Let's dive into more about these DEFX tags with examples for different purposes.

Let's look at the syntax and examples of these tags. Using a pipe to divide sections is by design to help with lining up sections of comments to make it look like data and easier review.

1. For derived Computation variables, the syntax is:

```
/* DEFX | < Dataset >.< Variable > | < Description > */
```

2. Examples:

```
/* DEFX | ADAE.AERELFL | Y if AE.AEREL="RELATED"     */
/* DEFX | ADAE.AERELFL | N if AE.AEREL="NOT RELATED" */
/* DEFX | ADAE.AERELFL | otherwise set to missing    */
```

3. If a description needs more lines to keep code and comments narrow for easy reading, then we can accommodate that as well:

```
/* DEFX | ADAE.AESAECAT | Group by reason for serious adverse event;       */
/* DEFX | ADAE.AESAECAT | if participant has more than 1 reason,          */
/* DEFX | ADAE.AESAECAT | then prioritize by death, life threatening,     */
/* DEFX | ADAE.AESAECAT | hospitalization, other:                          */
/* DEFX | ADAE.AESAECAT | if AE.AESDTH="Y"  then                           */
/* DEFX | ADAE.AESAECAT | AESAECAT='Results in Death';                     */
/* DEFX | ADAE.AESAECAT | else if AE.AESLIFE='Y' then                     */
/* DEFX | ADAE.AESAECAT | AESAECAT='Is Life Threatening';                  */
```

```
/* DEFX | ADAE.AESAECAT | else if AE.AESHOSP='Y' then                        */
/* DEFX | ADAE.AESAECAT | AESAECAT='Requires or Prolongs Hospitalization'; */
/* DEFX | ADAE.AESAECAT | else if AE.AESMIE='Y' then AESAECAT=              */
/* DEFX | ADAE.AESAECAT | 'Other Medically Important Serious Event';       */
```

4. Table 1 shows other types of tags and their syntax.

Type of Tag	Syntax and/or Example
Imputation Variable	/* DEFX | < Dataset >.< Variable > | Imputation | < Description > */
Predecessor Variable	/* DEFX | < Dataset >.< Variable > | Predecessor | < Predecessor Dataset.Variable > */
Assigned Variable	/* DEFX | < Dataset >.< Variable > | Assigned */
Comments for a Variable	/* DEFX | < Dataset >.< Variable > | Comments | < Description > */
Comments for a Dataset	/* DEFX | < Dataset > | Comments | < Description > */
Method that Applies to A Variable in All Datasets	/* DEFX | < Variable > | < Description > */ Example: /* DEFX | ACALDAY | ACALDAY=ADADHERE.ADT-ADSL_PRE.RFXSDT; */ /* DEFX | ACALDAY | if missing treatment start date (ADSL_PRE.RFXSDT) */ /* DEFX | ACALDAY | then ACALDAY=ADADHERE.ADT-ADSL_PRE.RFSDT */
AVAL by PARAMCD	/* DEFX | AVAL.< Dataset >.PARAMCD.EQ.< PARAMCD VALUE > | < Description > */ NOTE: This tag only updates the Methods tab and not the Variables tab.

Table 1. Types of Tags and Syntax

Related to these DEFX tags are checks that the DEFX macro performs:

1. A < Dataset >.< Variable > must be created in only one program
2. A < Dataset >.< Variable > or < Variable > must exist in the “Variables” tab in the define.xml

DEFX MACRO PARAMETERS

What are the parameters of the DEFX macro?

The first required parameters of the macro are METALIB and DEFXLSX. The METALIB parameter contains the LIBNAME that will store the metadata dataset created by the macro to fill in the define.xlsx file identified in DEFXLSX. The program uses a .xlsx file as input and output because it is easier to edit than a .xml file and easy to convert to XML. The metadata dataset created allows traceability from the DEFX tag to the define.xlsx file, including variables created within the macro. We recommend that METALIB location and location of the DEFXLSX file be the same. In our process, we have a program named p21zip that creates a standard directory location and name, /p21, based on an analysis location and contains a zip file ready for upload to the Pinnacle21 website. This is typically the location of the define.xlsx file created by the Pinnacle21 website and used in the DEFX macro.

For the other parameters, OUTXLSX lets the user change the name and location of the define.xlsx file output, but the default is the value in DEFXLSX. The TAGLOC parameter contains the location of the directory, or directories, that have the SAS programs to search for DEFX tags. The default is the current location of the program running the DEFX macro.

We recently added another parameter, KEEPTAGS. This allows a user to keep the contents of the define.xlsx file when it differs from what is found in the DEFX tags. The default behavior of DEFX is to overwrite any content in the define.xlsx file with what is found in the DEFX tags.

Table 2 is a summary of these parameters.

Parameter Name **R = Required** **O = Optional**	**Description**
(R) METALIB	Library name where the Metadata Dataset output by the macro should be stored. The /p21 subdirectory within an analysis or monitoring directory is the recommended input for the METALIB parameter. Both the defxit (script version of DEFX macro – see below) and p21zip scripts output their data products to the /p21 directory by default.
(R) DEFXLSX	Full file path and name of the define .xlsx file (with extension) that should be updated by the macro.
(O) OUTXLSX	Full file path and name of a .xlsx file (with extension) to save updated define.xml metadata. Default is &DEFXLSX
(O) TAGLOC	A file path (or multiple file paths separated by a space) containing SAS files with the tags for updating the define .xlsx file from Pinnacle21. The default is the current folder containing the program calling %DEFX.
(O) KEEPTAGS	Option to keep rows in the Methods and Comments tabs that do not match with current tags in &TAGLOC programs. Default is N N - Only include information from current tags in Methods and Comments tabs Y - Retain existing Methods and Comments that do not match with current tags If manual edits have been made to a define.xlsx document and there are no corresponding tags present in &TAGLOC, this option allows the user to retain the manually entered comments in &OUTXLSX.

Table 2. DEFX Macro Parameters and Descriptions

DEFX MACRO OUTPUT

The standard edits to the DEFXLSX file involve:

- Datasets tab:
    - Comments on a dataset will be added to the “Comment” column
- Variables tab:

    - "Origin" and "Method" columns are filled in based on "Computation" and "Imputation" tags
        - See Figure 2 and Figure 3 in DEFX EXAMPLES.
    - All "Predecessor" variables will have the "Origin" and "Predecessor" columns updated
        - See Figure 1 in DEFX EXAMPLES.
    - All "Assigned" variables will have the "Origin" column updated
    - Comments on a variable will be added to the "Comment" column
- Methods tab:
    - All "Computation" and "Imputation" methods will be filled in and adding rows for missing values
    - Any "Computation" or "Imputation" method with more than one '.', e.g., AVAL and PARAMCD notation, will be added only to the Methods tab. It will change nothing in the Variables tab. See Figure 4 in DEFX EXAMPLES.
- Comments tab:
    - All comments will be added to "Description" column and adding rows for missing values

The metadata dataset contains the following variables and is the intermediate step between the DEFX tag and filling out the define.xlsx:

Variable Name	Description
Program	Name of program containing the DEFX tag
Name	The second part of the tag in one of these formats: < Dataset > < Variable > < Dataset >.< Variable > AVAL.< Dataset >.PARAMCD.EQ.<PARAMCD VALUE>
Dataset	The < Dataset > part of Name
Variable	The < Variable > part of Name
Type	Contains one of 3 values from the third part of some DEFX tags: Imputation Computation – created within DEFX if there is no designation of "Assigned", "Imputation", "Predecessor", and "Comments" from the DEFX Tag Comments
Origin	The value is derived based on the original value in Type read from the DEFX Tag: Assigned Derived – created within DEFX if Type is "Imputation" or "Computation" Predecessor
Predecessor	Lists the Predecessor when the Origin value is "Predecessor"
Description	The description from the last part of the DEFX tags and if the DEFX tag was on multiple lines, then this will contain all lines concatenated together in order.

Table 3. Types of Tags and their Syntax

DEFX MACRO STEPS

STEP 1: MAKE DATASETS OF THE TABS IN DEFINE.XLSX

We use the PROC IMPORT code for the following tabs from the define.xlsx file entered in DEFXLSX:

- "Methods"

- “Variables”
- “Comments”
- “Datasets”

In Program 1, we show the code for the “Methods” tab. For the other tabs, we just change the OUT= option name and SHEET statement name. We have the output dataset name start with “OLD_” to identify what came from the define.xlsx file in the DATAFILE= option. We use the REPLACE option to ensure we have what is expected in case there’s is another dataset with the same name. The SHEET= statement identifies the tab name to make into a dataset. The GETNAMES= statement gets the variable names from the first row of the tab. To allow spaces in the name of variables, the program uses the VALIDVARNAME = ANY option.

Once we have created a dataset, we collect the order of variables in a list so we can output our edits in the same order to the new metadata dataset and define.xlsx file we output. We use an underscore as the first character in an effort to avoid interfering with a macro variable with the same name.

```
/* NEED validvarname to keep spaces in variable names from the Define.xlsx file */
options validvarname = ANY;

/* READ IN METHODS TAB INTO A DATASET NAMED OLD_METHODS */

proc import OUT      = OLD_METHODS
            DATAFILE = "&DEFXLSX"
            DBMS     = xlsx
            REPLACE
            ;
   SHEET    = "Methods";
   GETNAMES = YES;
run;

/* Store column order of spreadsheet for use in a RETAIN statement later in code */
/* Because some columns have spaces, we need to separate the column names     */
/* with "n so that they resolve to:                                             */
/* "Data Type"n                                                                 */
/* which will make SAS see that as one name and not 2                           */

proc sql noprint;
   select name into : _METHORD separated by '"n "'
   from dictionary.columns
   where libname = "WORK" and
         memname = "OLD_METHODS"
   ;
quit;
```

Program 1. Read in each tab into a dataset

STEP 2: COLLECT DEFX TAGS INTO A DATASET

In Program 2, we use the FILENAME statement with the PIPE option to call a Linux command and create a file easily read into a dataset.

```
/*************************************************************************/
/* IMPORT TAGS                                                           */
/*************************************************************************/
/* This command stores output into a filename so it can be imported as a dataset */
/* It is grepping all the SAS files in the same directory as the program running */
/* this macro                                                            */
/* It grabs all rows containing both the tag 'DEFX' and the '|' separator,       */
/* so headers and macro calls do not get grabbed                         */
/* It then drops the tag and adds a pipe after the program name so the metadata  */
/* dataset contains the program where the tag originated                 */
```

```
filename tags pipe
"defxfiles=`echo &GREPFILES | sed 's|@|/*.sas|g'`; grep -i 'DEFX' $defxfiles |
grep  '|' | sed 's/sas:/sas \|/g' | sed 's|/\*||g' | sed 's|\*/||g' | cut -d'|' -f1,3-" ;
```

Program 2. Collect DEFX Tags Into a File

The key Linux command is GREP. This command locates every line with a search string within a set of files. The output from this command lists the name of the file first, which is how we can populate the PROGRAM variable in the output dataset. For the DEFX macro, it greps the SAS files in the TAGLOC directories or directory from which the DEFX macro runs. It grabs the rows that contain both the "DEFX" tag and the "|" separator. This avoids grabbing lines from our header or calls to the DEFX macro. The rest of the Linux command cleans up the line so there are only "|" separators and eliminates the /* */ comment characters. So, the FILENAME has lines that look like this:

makeadqol.sas | ADQOL.AVAL.TOTSC | If less than 50% of ADQS.AVAL where ADQS.PARCAT1 = "Quality of Life" are missing, then
makeadqol.sas | ADQOL.AVAL.TOTSC | 100 - 25 * (average of ADQS.AVAL where ADQS.PARCAT1 = "Quality of Life").
makeadqol.sas | ADQOL.PARAMCD | Assigned
makeadqol.sas | ADQOL.PARAM | Assigned
makeadqs.sas | ADQS.QSTESTCD | Predecessor | QS.QSTESTCD
makeadqs.sas | ADQS.QSTEST | Predecessor | QS.QSTEST

This file is then read into a dataset via an INFILE statement and uses INPUT lines to create four variables in this order: Program, Name, Type, and Description. If Description does not have a value, then the value in TYPE is moved to the DESCRIPTION variable. In a later data step, if Type is missing, then the value changes to "Computation". Multiple lines of a description become one row in the dataset.

FINAL STEP: MERGE DATASETS TO UPDATE DEFINE.XLSX FILE

After the program does minor checks already listed above, the program then merges the dataset with tag information with the appropriate "OLD_" dataset. The value of KEEPTAGS, will determine the order of the MERGE:

- The default for KEEPTAGS is N, so that new tag information is preferred in the output.
- If KEEPTAGS is Y, then the program favors the information already in the define.xlsx.

Once we have the dataset that we want to use to update the define.xlsx ,then we use PROC EXPORT with similar code to the PROC IMPORT, but use an OUTFILE= option instead of a DATAFILE= option for the define.xlsx file name to write to. We reference this file as &OUTXLSX and use PUTNAMES = YES instead of GETNAME= to populate the spreadsheet with the names from the dataset.

```
/* Update the Methods Tab in the DEFINE XML*/
proc export data    = __METHODS
            outfile = "&OUTXLSX"
            dbms    = xlsx
            REPLACE
            ;
    sheet = "Methods";
    putnames = yes;
run;
```

Program 3. PROC EXPORT Code for Methods Tab

DEFX EXAMPLES

Let's see what this looks like. In the directory where we make our datasets, we have a program with the following DEFX tags within it along with several others interspersed with code creating the variables in our final dataset:

```
/* DEFX | ADILB.LBDTC     | Predecessor | LB.LBDTC */
/* DEFX | ADILB.LBDY      | Predecessor | LB.LBDY */
/* DEFX | ADILB.SRCFORM   | If SRCDOM = "LB", then SUPPLB.QVAL where   */
/* DEFX | ADILB.SRCFORM   | SUPPLB.QNAM = "SRCFORM". If SRCDOM = "MB", */
/* DEFX | ADILB.SRCFORM   | then SUPPMB.QVAL where SUPPMB.QNAM = "SRCFORM". */


/* DEFX | ADILB.AVAL.ALT     | LB.LBSTRESN where LB.LBTESTCD = "ALT" and LBSTAT is not "NOT DONE"    */
/* DEFX | ADILB.AVAL.BILI    | LB.LBSTRESN where LB.LBTESTCD = "BILI" and LBSTAT is not "NOT DONE"   */
/* DEFX | ADILB.AVAL.BUREAN  | LB.LBSTRESN where LB.LBTESTCD = "BUREAN" and LBSTAT is not "NOT DONE" */
```

After we validate and run our programs making our datasets and submit those datasets to Pinnacle21 to create a define.xml in our /p21 directory, we run our DEFX program. In the directory of our data creation programs, there is a SAS program that only calls DEFX:

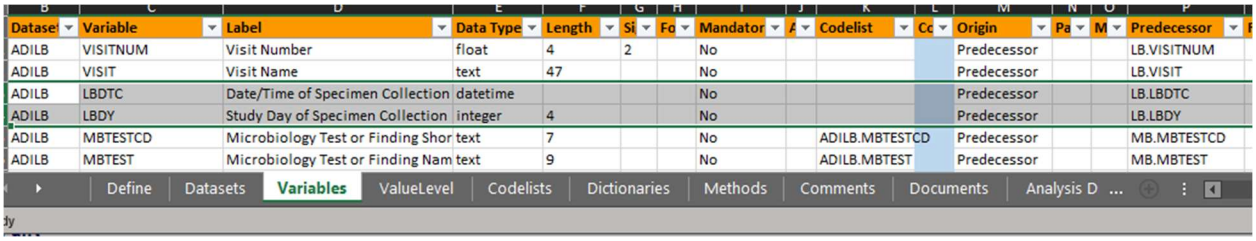
```
/* Typically, our LIBNAME statements live in an autoexec.sas file, */
/* but for this example, we are showing it within our DEFX program   */

libname metadata "/my/study/analysis/makedata/programs";

* Create dataset within the metadata library and populate the shell located as
specified below;
%DEFX( METALIB = metadata,
       DEFXLSX = ../p21/define.xlsx
     );
```

Our users prefer a Linux script to run the DEFX macro because they do not want a program in their data creation directory that does not make data. The programmers created defxit which lets users run the command at the Linux prompt with the default location being the same as where we store other Pinnacle21 related files.

Once we run the program with our DEFX macro, then we see our define.xlsx file populated with the information from our DEFX tags. Figure 1 shows the Variables tab update of Predecessor information (Origin and Predecessor columns) for ADILB.LBDTC and ADILB.LBDY from the tags above.

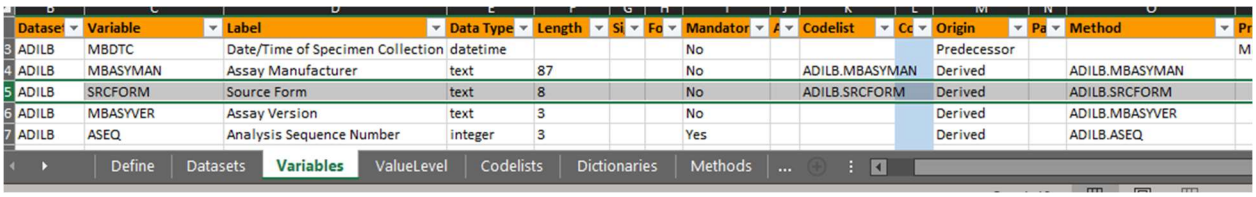


Dataset	Variable	Label	Data Type	Length	Si	Fo	Mandator	A	Codelist	Co	Origin	Pa	M	Predecessor
ADILB	VISITNUM	Visit Number	float	4	2		No				Predecessor			LB.VISITNUM
ADILB	VISIT	Visit Name	text	47			No				Predecessor			LB.VISIT
ADILB	LBDTC	Date/Time of Specimen Collection	datetime				No				Predecessor			LB.LBDTC
ADILB	LBDY	Study Day of Specimen Collection	integer	4			No				Predecessor			LB.LBDY
ADILB	MBTESTCD	Microbiology Test or Finding Shor	text	7			No		ADILB.MBTESTCD		Predecessor			MB.MBTESTCD
ADILB	MBTEST	Microbiology Test or Finding Nam	text	9			No		ADILB.MBTEST		Predecessor			MB.MBTEST

Define | Datasets | Variables | ValueLevel | Codelists | Dictionaries | Methods | Comments | Documents | Analysis D ...

Figure 1. ADILB.LBDTC and ADILB.LBDY Predecessor Update in Variables Tab

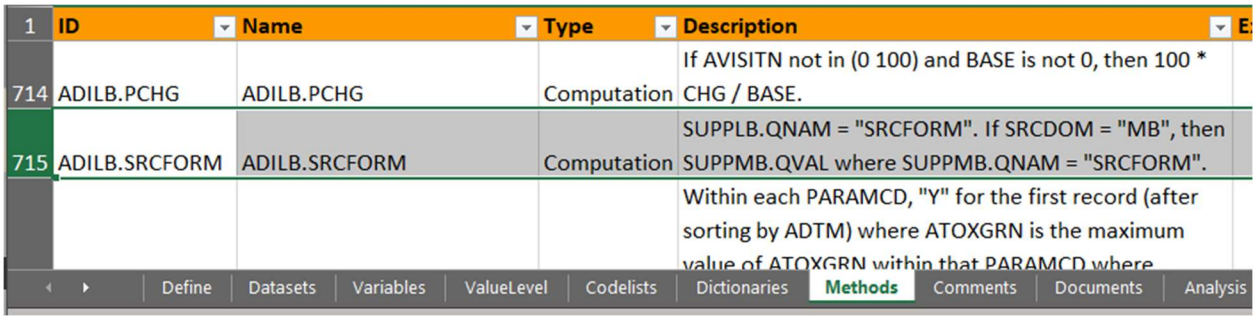
For the ADILB.SRCFORM variable we define, Figure 2 shows that the Origin and Method columns update in the Variables tab and Figure 3 shows the updates in the Description in the Methods tab, which is on multiple lines of comments above.



Dataset	Variable	Label	Data Type	Length	Mandator	Codelist	Origin	Method
ADILB	MBDTC	Date/Time of Specimen Collection	datetime		No		Predecessor	
ADILB	MBASYMAN	Assay Manufacturer	text	87	No	ADILB.MBASYMAN	Derived	ADILB.MBASYMAN
ADILB	SRCFORM	Source Form	text	8	No	ADILB.SRCFORM	Derived	ADILB.SRCFORM
ADILB	MBASYVER	Assay Version	text	3	No		Derived	ADILB.MBASYVER
ADILB	ASEQ	Analysis Sequence Number	integer	3	Yes		Derived	ADILB.ASEQ

Define | Datasets | Variables | ValueLevel | Codelists | Dictionaries | Methods | ...

Figure 2. ADILB.SRCFORM Imputation Update in Variables Tab

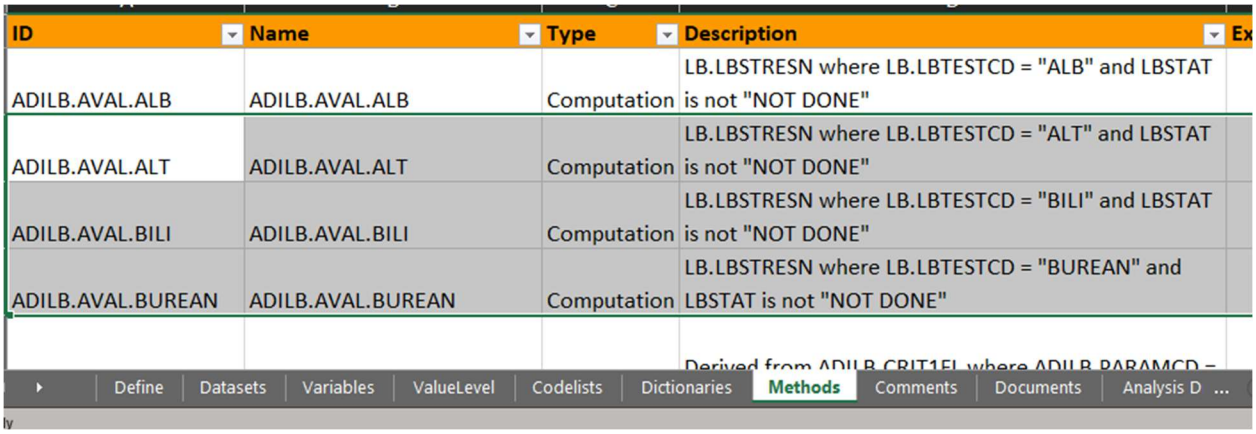


	ID	Name	Type	Description
714	ADILB.PCHG	ADILB.PCHG	Computation	If AVISITN not in (0 100) and BASE is not 0, then 100 * CHG / BASE.
715	ADILB.SRCFORM	ADILB.SRCFORM	Computation	SUPPLB.QNAM = "SRCFORM". If SRCDOM = "MB", then SUPPMB.QVAL where SUPPMB.QNAM = "SRCFORM".
				Within each PARAMCD, "Y" for the first record (after sorting by ADTM) where ATOXGRN is the maximum value of ATOXGRN within that PARAMCD where

Define | Datasets | Variables | ValueLevel | Codelists | Dictionaries | Methods | Comments | Documents | Analysis

Figure 3. ADILB.SRCFORM Description Update in Methods Tab

Figure 4 shows the Methods tab update of Description for ADILB.AVAL.<VALUE> values.



ID	Name	Type	Description
ADILB.AVAL.ALB	ADILB.AVAL.ALB	Computation	LB.LBSTRESN where LB.LBTESTCD = "ALB" and LBSTAT is not "NOT DONE"
ADILB.AVAL.ALT	ADILB.AVAL.ALT	Computation	LB.LBSTRESN where LB.LBTESTCD = "ALT" and LBSTAT is not "NOT DONE"
ADILB.AVAL.BILI	ADILB.AVAL.BILI	Computation	LB.LBSTRESN where LB.LBTESTCD = "BILI" and LBSTAT is not "NOT DONE"
ADILB.AVAL.BUREAN	ADILB.AVAL.BUREAN	Computation	LB.LBSTRESN where LB.LBTESTCD = "BUREAN" and LBSTAT is not "NOT DONE"
			Derived from ADILB.CRIT1FL where ADILB.PARAMCD =

Define | Datasets | Variables | ValueLevel | Codelists | Dictionaries | Methods | Comments | Documents | Analysis D ...

Figure 4. ADILB.AVAL.<VALUE> Descriptions Update in Methods Tab

As this example shows, the DEFX tag system does not get someone out of updating the define.xml, but lets them do it as they are programming as opposed to doing it when all programming is done and searching for the program and information to place in the define.xml. At CBAR, these comments are reviewed and validated by someone before the final run, so they should accurately reflect what is in the code and thus allow a user to comment their code and fill in their define.xml.

CONCLUSION

The DEFX code is below in the Appendices. It does contain some homecooked macros for simple tasks, but these are described so a reader can create their own version, if they don't already have one. The DEFX program is a simple tool to help with other processes. It encourages better and more consistent commenting because it rewards users by avoiding time and effort filling out a define.xml for easy edits at the end of a project. I hope you feel like it's one of those buy-one-and-get-one-free sales you can't pass up and will use it to help with your current work processes.

REFERENCES

To better understand the Linux code, copy and paste it into either explainshell.com, or shellcheck.net, or potentially look at Bash Reference Manual for a deep dive.

ACKNOWLEDGMENTS

This work was supported by the Statistical and Data Management Center (SDMC), of the Advancing Clinical Therapeutics Globally for HIV/AIDS and Other Infections (ACTG), under the National Institute of Allergy and Infectious Diseases (NIAID) grant No. UM1 AI068634.

This work was supported by the Statistical and Data Management Center (SDMC) of the International Maternal Pediatric Adolescent AIDS Clinical Trials Network (IMPAACT) under the National Institute of Allergy and Infectious Diseases (NIAID) grant No. UM1 AI068616.

Thank you to Megan Hinger for creating the DEFX macro and Shawn Ward for heavily testing the program. Thank you to Scott Anderson for creating the defxit script. Thank you to Heather Ribaudo for the time to work on this paper and presentation.

CONTACT INFORMATION

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

Brendan Bartley
bbartley@sdac.harvard.edu

Megan Hinger
megan_hinger@rhoworld.com

APPENDICES

APPENDIX A: DEFX CODE

Please excuse the small font, but I believe in code aesthetics and larger font made it difficult to read the code. Please utilize document magnification for viewing.

```
/*************************************************************************/
/* PURPOSE:         Grab tags from SAS files and then use the info to update the
/*                  Define XML 'Methods' and 'Variables' tabs
/*
/* INPUT:           An excel File
/* OUTPUT:          Updated Input Excel File and metadata dataset: &METALIB.meta_defx
/* MACROS USED:     %maxlength, %nobs, %trimlen
/* NOTES:           - TO SHORTEN CODE TO THE IMPORTANT PROCESSING,
/*                    REMOVED CHECKS OF VALUES ENTERED IN PARAMETERS
/*                  - REMOVED %local STATEMENT FOR MANY MACRO VARIABLES
/*************************************************************************/
%macro defx( METALIB   =   /* LIBNAME TO STORE METADATA DATASET                        */
           , DEFXLSX   =   /* FULL PATH AND NAME OF define.xlsx FILE TO EDIT           */
           , OUTXLSX   =   /* OUTPUT FULL PATH AND NAME OF define.xlsx EDITED          */
           , TAGLOC    =   /* FILEPATH LOCATIONS CONTAINING SAS PROGRAMS WITH TAGS     */
           , KEEPTAGS  = N /* N = OVERWRITE INFORMATION IN Methods and Comments TABS   */
                           /* Y = RETAIN INFORMATION ALREADY IN Methods AND Comments TABS */
           );
%end;

%if %length(&KEEPTAGS) = 0 %then %let KEEPTAGS = N;
%else                            %let KEEPTAGS = %substr(%upcase(&KEEPTAGS),1,1);


/*************************************************************************/
```

```
/* IMPORT EXCEL SHEETS                                                        */
/*****************************************************************************/
/* Need validvarname to keep spaces in variable names from the Define Excel Spreadsheet*/
options validvarname = ANY;

proc import out      = OLD_METHODS
            DATAFILE = "&DEFXLSX"
            DBMS     = xlsx
            REPLACE
            ;
    SHEET    = "Methods";
    GETNAMES = YES;
run;

proc import out      = OLD_VARS
            DATAFILE = "&DEFXLSX"
            DBMS     = xlsx
            REPLACE
            ;
    SHEET    = "Variables";
    GETNAMES = YES;
run;

/* Must also import 'Comments' and 'Datasets' tab for new Comments tag */
proc import out      = OLD_COMMS
            DATAFILE = "&DEFXLSX"
            DBMS     = xlsx
            REPLACE
            ;
    SHEET    = "Comments";
    GETNAMES = YES;
run;

proc import out      = OLD_DSETS
            DATAFILE = "&DEFXLSX"
            DBMS     = xlsx
            REPLACE
            ;
    SHEET    = "Datasets";
    GETNAMES = YES;
run;

/*****************************************************************************
*** GENERATE LISTS FROM EXCEL SHEETS                                       ***
*****************************************************************************/
proc sql noprint;

    /* Need to get list of datasets so tags that have only one name          */
    /* not 'DATASET.VARIABLE' can be determined if it's a dataset or variable */
    select dataset into : DNMS separated by '","'
    from OLD_DSETS
    ;

    /* Store column orders from the spreadsheet for use in a retain  */
    /* Because some columns contain spaces, we need to separate the  */
    /* column names with "n so they resolve like:                    */
    /* "Data Type"n                                                  */
    /* Which will make SAS see that as one name and not 2: Data Type */

    select name into :_METHORD separated by '"n "'
    from dictionary.columns
    where libname = "WORK" and
          memname = "OLD_METHODS"
    ;

    select name into :_VARORD separated by '"n "'
    from dictionary.columns
    where libname = "WORK" and
          memname = "OLD_VARS"
```

```
    ;

    select name into :_COMORD separated by '"n "'
    from dictionary.columns
    where libname = "WORK" and
          memname = "OLD_COMMS"
    ;

    select name into :_DSETORD separated by '"n "'
    from dictionary.columns
    where libname = "WORK" and
          memname = "OLD_DSETS"
    ;
quit;

/*****************************************************************************/
/* PROCESS GREP LOCATION(S)                                                  */
/*****************************************************************************/
%if %length(&TAGLOC) > 0 %then %do;

    %let GREPFILES = ;

    %do i = 1 %to %sysfunc(countw(&TAGLOC, ' '));
        %let GREPFILES = &GREPFILES %scan(&TAGLOC,&i,' ')@;
    %end;
%end;
%else %do;

    %let GREPFILES = *.sas;
%end;

/*****************************************************************************/
/* IMPORT TAGS                                                               */
/*****************************************************************************/
/* This command stores output into a filename so it can be imported as a dataset */
/* It is grepping all the SAS files in the same directory as the program running this macro */
/* It grabs all rows containing both the tag 'DEFX' and the '|' separator,       */
/* so headers and macro calls don't get grabbed                                  */
/* It then drops the tag and adds a pipe after the program name                  */
/* so the meta dataset contains the program where the tag originated             */

filename tags pipe "defxfiles=`echo &GREPFILES | sed 's|@|/*.sas|g'`; grep -i 'DEFX' $defxfiles | grep
'|' | sed 's/sas:/sas \|/g' | sed 's|/\*||g' | sed 's|\*/||g' | cut -d'|' -f1,3-" ;

%let GREPERR = ;

/*Import the grep tags as a dataset. Use the piped results as the input. */
data __tags;

    length Program $100.
           Name $40.
           Type $1000.
           Description $1000.
           err $100
           ;

    infile tags DLM="|" missover end = eof;
    input Program $ Name $ Type $ Description $;

    Name = upcase(Name);

    retain err "";

    /* Depending on if the tag did NOT include the Type, then Description got put */
    /* in Type and needs to be moved to Description and Type should be blank      */
    if missing(Description) then do;
        Description = Type;
        Type        = "";
    end;
```

```
    /* Check for grep error and send it to macrovariable                    */
    /* Have to add a space between / and * in grep command or it won't print */
    if find(Program,"grep:") then do;
        err = catx(' ',err,substr(tranwrd(Program,'/*','/ *'),6));
        err = substr(err,1,index(err,'*')-1);
    end;

    if eof then do;
        call symput('GREPERR',err);
    end;

    drop err;
run ;

/* If grep has an error then exit */
%if %length(&GREPERR) > 0 %then %do;
    %put %upcase(error: (cbar) &__COREPROG -) The following Directory(ies) does not exist or does not
contain SAS files:;
    %put %upcase(error: (cbar) &__COREPROG -) &GREPERR;
    %return;
%end;

/* %NOBS is a CBAR macro that obtains the number of observations in a dataset */
%if %nobs(__tags) = 0 %then %do;
    %put %upcase(error: (cbar) &__COREPROG -) No tags were found. Please check the README for the correct
tag format.;
    %return;
%end;

/*****************************************************************************
*** Edit Master Dataset                                                     ***
*****************************************************************************/
/* Reorder Columns into program name, columns needed for Methods tab, */
/* columns needed for Variables tab                                   */
data __tags;
    set __tags
    ;

    if missing(Type) then Type = "Computation";
    else                  Type = propcase(Type);

    /* Switched substr to scan to accommodate new naming convention */
    /* scan will work if Name is ".Variable"                        */
    if countc(Name,'.') = 1 then do;

        Dataset  = scan(Name,1,'.');
        Variable = scan(Name,2,'.');
    end;
    else if countc(Name,'.') = 0 then do;

        if Name in ("&DNMS") then Dataset  = Name;
        else                      Variable = Name;
    end;
run;

/* Data needs to be sorted in order to use by statement to merge multi-line descriptions*/
/* Have to add type to handle multi-line comments */
proc sort data = __tags;
    by Program Name Type;
run;


/* Use retain to combine multiple row descriptions */
data __tags;
    set __tags
        ;
    by Program Name Type;
```

```
    length Desc $1000 ;
    retain Desc "";

    if first.Name or first.Program or first.Type then Desc ="";

    if last.Name or last.Program or last.Type then do;
        Description = catx('',Desc,Description);
        output;
    end;
    else Desc = catx('',Desc,Description);
    drop Desc;
run;

/* Removed blanking of Name so it can be used for tag checking */
data __tags;
    set __tags;
    length Predecessor $20 Origin $25;

    if strip(Type)  = "Predecessor" then do;
        Origin      = Type;
        Predecessor = Description;
        Description = "";
        Type        = "";
    end;
    else if strip(propcase(Description)) = "Assigned" then do;
        Origin      = "Assigned";
        Description = "";
        Type        = "";
    end;
    else if strip(Type) in ("Imputation" "Computation") then Origin = "Derived";
run;

/* Prepare for merge by sorting */
proc sort data = __tags
          out  = addcoms (where= (Type = "Comments")) nodupkey;
    by Name;
run;

proc sort data = __tags
          out  = sorttags;
    by Name;
run;

/* Need to add comments column by merging */
data __tags;

    retain Program Name Dataset Variable Type Origin Predecessor ;
    merge addcoms  (in = in1 keep = Name)
          sorttags (in = in2)
          ;
    by Name;

    if in1 then CFL = "Y";
run;

data __tags;
    set __tags;

    if   CFL = "Y" then Comment = Name;
    drop CFL;
run;

/* %trimlen is a CBAR macro that trims the length of all variables               */
/* It is needed here because there was a large allowance for Description lengths */
%trimlen(INDATA = __tags);

/* Save a permanent dataset with all the information that is going to be          */
/* added to the Define XML.                                                       */
/* Trimlen resets variable order, so retain needs to be set after running trimlen. */
```

```
data __tags;
    retain Program Name Dataset Variable Type Origin Predecessor Comment;
    set __tags;
run;

/*************************************************************
*** DO TAG CHECKS                                          ***
*************************************************************/

proc sort data = __tags;
    by Type Name;
run;

/* Flag any Names made in more than one program. */
data badtgs;
    set __tags;
    by Type Name ;
    if not (first.Name and last.Name) then DUPFL = "Y";
run;

/* Create master list of variables in define */
data chknms;
    length Predecessor $20;
    set OLD_VARS;

    Name        = catx('.',strip(Dataset),strip(Variable));
    Predecessor = Name;

    keep Dataset Variable Name Predecessor;
run;

proc sort data = chknms nodupkey;
    by Name;
run;

proc sort data = badtgs;
    by Name;
run;

/* Check Dataset and Variable combos */
data badnms;
    merge chknms (in = inc keep = Name)
          badtgs (in = inm)
          ;
    by Name;

    if inm;
    if inm and not inc and countc(Name,'.') = 1 then BADNMFL = "Y";
run;

proc sort data = chknms
          out  = chkvrs nodupkey;
    by Variable;
run;

proc sort data = badnms;
    by Variable;
run;

/* %maxlength is a CBAR macro that finds max length for same named variables in DSETS */
/* then creates a macro variable named MAXLENGTH that is a LENGTH statement for these */
/* same named variables - see data step below                                          */
%maxlength(dsets = chkvrs badnms)

/* Check variable only tags */
data badvrs;
    retain DUPFL BADNMFL BADVRFL Program Name Dataset Variable
           Type Origin Predecessor Comment;
```

```
    &MAXLENGTH;

    merge chkvrs (in = inc keep = Variable)
          badnms (in = inm)
          ;
    by Variable;

    if inm;
    if inm and not inc and countc(Name,".") = 0 and missing(Dataset) then BADVRFL = "Y";
run;

data &METALIB..meta_defx;
    retain DUPFL BADNMFL BADVRFL Program Name Dataset Variable
           Type Origin Predecessor Comment;
    set badvrs;
run;

/* Added check and exit when commas appear in Name column */
%if %nobs(badvrs, where = countc(Name,",")) > 0 %then %do;
    %put %upcase(error: (cbar) &__COREPROG -) A comma appears to be entered in the <dataset>.<variable>;
    %put %upcase(error: (cbar) &__COREPROG -) column of a &__COREPROG tag. Please remove the comma and;
    %put %upcase(error: (cbar) &__COREPROG -) rerun the program. See .lst file for list of Programs and
tags.;
    %put %upcase(error: (cbar) &__COREPROG -) &__COREPROG will now exit.;

    title1 'List of tags containing commas in your TAGLOC directory.';
    proc print data = badvrs;
        var Program Name;
        where countc(Name,",") > 0;
    run;
    %return;
%end;

%if %nobs(badvrs,where = DUPFL = 'Y') > 0 %then %do;
    %put %upcase(note: (cbar) &__COREPROG -) There are tag Names that have been defined in multiple
programs;
    %put %upcase(note: (cbar) &__COREPROG -) These tags will NOT be added to your DEFINE XLSX.;

    title1 "At least one Tag has been created in more than one program. See below for details.";
    proc print data = badvrs (where = (DUPFL = "Y"));
    run;
%end;

%if %nobs(badvrs,where = BADNMFL = 'Y') > 0 %then %do;
    %put %upcase(note: (cbar) &__COREPROG -) There are tags for Dataset.Variable combos that do NOT exist
in the Variables tab of your DEFINE;
    %put %upcase(note: (cbar) &__COREPROG -) These tags will NOT be added to your DEFINE XLSX.;

    title1 "At least one Dataset.Variable combo does NOT exist in the Variables tab of your DEFINE.”;
    title2 ”See below for details.";
    proc print data = badvrs(where = (BADNMFL = "Y"));
    run;
%end;

%if %nobs(badvrs,where = BADVRFL = 'Y') > 0 %then %do;
    %put %upcase(note: (cbar) &__COREPROG -) There are single name tags that do NOT match a dataset or
variable.;
    %put %upcase(note: (cbar) &__COREPROG -) These tags will NOT be added to your DEFINE XLSX.;

    title1 "At least one single name tag does NOT match a dataset or variable name";
    title2 "in the Variables tab of your DEFINE. See below for details.";
    proc print data = badvrs(where = (BADVRFL = "Y"));
    run;
%end;

title;

/* Create a dynamic list of all flags that can be used in all where statements */
proc sql noprint;
```

```
    select Name, Name into :BADFL separated by " ne 'Y' and ", :DROPFL separated by " "
    from dictionary.columns
    where libname = "WORK"   and
          memname = "BADVRS" and
          substr(name,length(name)-1) = "FL"
          ;
quit;


/* If using OUTXLSX option copy DEFXLSX to OUTXLSX. Else, define */
/* OUTXLSX as DEFXLSX.                                           */

%if %length(&OUTXLSX) > 0 %then %do;

      x "cp &DEFXLSX &OUTXLSX";
%end;
%else %let OUTXLSX = &DEFXLSX;

/*****************************************************************************/
/* UPDATE Methods                                                            */
/*****************************************************************************/
/* Need to sort and Merge by so any columns not created using the tags and already */
/* filled in the define XML aren't deleted                                   */
/* Do not add flagged variables to DEFINE or Predecessor or Assigned         */
/* Adjusted to use where over if and use any flagged variables               */

data NEW_METHODS;
    set &METALIB..meta_defx (where = (strip(Type) not in ("" "Comments" "Predecessor" "Assigned") and
                                      &BADFL ne 'Y'));
    drop Program Dataset Variable Origin Predecessor &DROPFL Comment;
    ID = Name;
run;

%if %nobs(NEW_METHODS) > 0 %then %do;

    proc sort data = NEW_METHODS;
        by Name;
    run;

    /* Drop Description and Type if not retaining old Methods */
    proc sort data = OLD_METHODS

        %if &KEEPTAGS = N %then %do;
            (drop = Description Type)
        %end;
        ;
        by Name;
    run;

    /* Reminder %maxlength is a CBAR macro finding and setting max length of same named variables */
    %maxlength(dsets = OLD_METHODS NEW_METHODS);

    /* Combine old and new, and drop any rows that are no longer in the tags */
    /* Do not drop any rows from old Methods sheet if user wants to          */
    /* retain previous unmatched methods (KEEPTAGS ne N)                     */
    data __METHODS;
        &MAXLENGTH;

        %if &KEEPTAGS = N %then %do;

            merge NEW_METHODS (in = in1)
                  OLD_METHODS
                  ;
            by Name;
            if in1;
        %end;
        %else %do;

            merge OLD_METHODS
```

```
                NEW_METHODS
                ;
            by Name;
        %end;
    run;

    /* Order variables using spreadsheet order  */
    data __METHODS;
        retain "&_METHORD"n;
        set __METHODS;
    run;

    /* Update the Methods Tab in the DEFINE XML*/
    proc export data    = __METHODS
                outfile = "&OUTXLSX"
                dbms    = xlsx
                REPLACE
                ;
        sheet    = "Methods";
        putnames = yes;
    run;
%end;

/*****************************************************************************/
/* UPDATE Variables                                                          */
/*****************************************************************************/
/* Added sort to be able to use first.Name */
proc sort data = &METALIB..meta_defx
          out  = sortvar;
    by Name;
run;

/* Grab any variables that apply to multiple datasets */
/* Drop any bad tags and the flags                    */
data Rpt_Vars;
    set sortvar (where = (missing(Dataset) and not missing(Variable) and &BADFL ne 'Y'));

    by Name;

    if not (first.Name and Last.Name) and Type = "Comments" then delete;

    /* Blank out Method for non-Derived Variable    */
    If Origin ne "Derived" then Name = "";

    rename Name = Method;
    drop Dataset &DROPFL Description Program Type;
run;

/* If there are variables that should apply accross multiple datasets they need */
/* To be merged differently                                                     */
/* Reminder - %nobs is CBAR macro that retrieves number of obs in a dataset     */
%if %nobs(RPT_Vars) > 0 %then %do;

    /*First sort for the merge*/
    proc sort data = RPT_Vars;
        by Variable;
    run;

    proc sort data = OLD_Vars;
        by Variable;
    run;

    /* Then make sure variable lengths are the same */
    %maxlength(dsets = RPT_Vars OLD_Vars);

    /* Then grab only rows from the Excel spreadsheet that has the relevant Variables */
    data multivars;
        &MAXLENGTH;
```

```
        merge OLD_Vars
              RPT_Vars (in = in1)
              ;
        by Variable;
        if in1;
    run;


    /*The merge only applies to the first variable match so a separate dataset */
    /* with a retain statement has to be used to carry the values down to all  */
    /* the other datasets                                                      */
    /* Added Predecessor to list of variables to carry the values              */
    data multivars;
        set multivars;

        by Variable;

        length Var  $40.
               Orig $20.
               Comm $20.
               Pred $20.
               ;

        retain Var "" Orig "" Comm "" Pred "";

        if first.Variable then do;
            Var  = Method;
            Orig = Origin;
            Comm = Comment;
            Pred = Predecessor;
        end;
        else do ;
            Method      = Var;
            Origin      = Orig;
            Comment     = Comm;
            Predecessor = Pred;
        end;

        drop Var Orig Comm Pred;
    run;

    /* Sort again to add these variables back to the full dataset to merge with */
    /* the rest of the Methods                                                  */
    proc sort data = OLD_Vars;
        by Dataset Variable;
    run;

    proc sort data = multivars;
        by Dataset Variable;
    run;

    %maxlength(dsets = multivars OLD_Vars);

    data OLD_Vars;
        &MAXLENGTH;
        merge OLD_Vars multivars;
        by Dataset Variable;
    run;
%end;


/* Variables like PARAMCD have multiple methods for the same variable                                 */
/* The 'Name' would look like AVAL.<DATASET>.PARAMCD.EQ.<Value>                                       */
/* They do not get populated in the Variables tab and need to be dropped from the dataset             */
/* Methods that need to be applied across multiple datasets have no Dataset value and need to be dropped*/
/* Adjusted to use where over if and use any flagged variables                                        */
data NEW_Vars;
    set sortvar (where = (not missing(Dataset)  and
                          not missing(Variable) and &BADFL ne 'Y'));
```

```
    by Name;
    if not (first.Name and Last.Name) and Type = "Comments" then delete;
    drop Description Program Type &DROPFL;

    /* Blank out Method for non-Derived variables    */
    if Origin ne "Derived" then Name = "";

    rename Name = Method;
run;

/* Added conditional so tabs are only updated as needed */
%if %nobs(NEW_Vars) > 0 %then %do;

    /* Need to sort for merging */
    proc sort data = NEW_Vars;
        by Dataset Variable;
    run;

    proc sort data = OLD_Vars;
        by Dataset Variable;
    run;

    %MAXLENGTH(DSETS = NEW_Vars OLD_Vars);

    /* Need to merge the New Methods into the Variables tab */
    /* Added all variables to retain statement              */
    /* add numeric order for sorting                        */
    data __Vars;
        &MAXLENGTH;
        merge OLD_Vars
              NEW_Vars
              ;
        by Dataset Variable;
        sortord = putn(order, 5.);
    run;

    /* Sort data by dataset and order to match original order in spreadsheet */
    proc sort data = __Vars;
        by Dataset sortord;
    run;

    /* Order variables using spreadsheet order  */
    data __Vars;
        retain "&_VARORD"n;
        set __Vars(drop = sortord);
    run;

    /* Update the Variables Tab in the DEFINE XML*/
    proc export data    = __Vars
                outfile = "&OUTXLSX"
                dbms    = xlsx
                REPLACE
                ;
        sheet    = "Variables";
        putnames = yes;
    run;
%end;

/*****************************************************************
*** Update Comments Tab                                       ***
*****************************************************************/
/* Extract any Comments from tab */
data NEW_COMMS;
    set &METALIB..meta_defx (where = (Type = "Comments" and &BADFL ne 'Y'));
    ID = Name;
    keep ID Description;
run;

/* Only update tab as needed */
```

```
%if %nobs(NEW_COMMS) > 0 %then %do;

    proc sort data = NEW_COMMS;
        by ID;
    run;

    /* Drop Description if not retaining old comments */
    proc sort data = OLD_COMMS

        %if &KEEPTAGS = N %then %do;
            (drop = Description)
        %end;
        ;
        by ID;
    run;

    %maxlength(dsets = OLD_COMMS NEW_COMMS);

    /* Combine old and new, and drop any rows that are no longer in the Comments */
    /* Do not drop any rows from old Comments sheet if the user wants            */
    /* to retain previous unmatched comments (KEEPTAGS ne N)                     */
    data __COMMS;
        &MAXLENGTH;

        %if &KEEPTAGS = N %then %do;

            merge NEW_COMMS (in = in1)
                  OLD_COMMS
                  ;
            by ID;
            if in1;
        %end;
        %else %do;

            merge OLD_COMMS
                  NEW_COMMS
                  ;
            by ID;
        %end;
    run;

    /* Order variables using spreadsheet order  */
    data __COMMS;
        retain "&_COMORD"n;
        set __COMMS;
    run;

    /* Update the Methods Tab in the DEFINE XML*/
    proc export data    = __COMMS
                outfile = "&OUTXLSX"
                dbms    = xlsx
                REPLACE
                ;
        sheet    = "Comments";
        putnames = yes;
    run;
%end;

/*****************************************************************
*** Update Datasets                                           ***
*****************************************************************/
/* Extract any Dataset Comments  */
data NEW_DSETS;
    set &METALIB..meta_defx (where = (Type = "Comments" and missing(Variable) and &BADFL ne 'Y'));
    Dataset = Name;
    keep Dataset Comment;
run;
```

```
/* Only updates tab as needed */
%if %nobs(NEW_DSETS) > 0 %then %do;

    proc sort data = NEW_DSETS;
        by Dataset;
    run;

    proc sort data = OLD_DSETS (drop = Comment);
        by Dataset;
    run;

    %maxlength(dsets = OLD_DSETS NEW_DSETS);

    data __DSETS;
        &MAXLENGTH;

        merge NEW_DSETS
              OLD_DSETS
              ;
        by Dataset;
    run;

    /* Order variables using spreadsheet order  */
    data __DSETS;
        retain "&_DSETORD"n;
        set __DSETS;
    run;

    /* Update the Datasets Tab in the DEFINE XML*/
    proc export data    = __DSETS
                outfile = "&OUTXLSX"
                dbms    = xlsx
                REPLACE
                ;
        sheet    = "Datasets";
        putnames = yes;
    run;
%end;

/************************/
/* RESET Options        */
/************************/
options validvarname = V7;

%mend defx;
```