PhUSE 2014

Paper AD09

Automated generation of program documentation by means of tagged SAS comments and metadata for integrated analysis

Thomas Wollseifen, inVentiv Health Germany GmbH, Eltville am Rhein, Germany

ABSTRACT

It is sometimes difficult to maintain documentation and keep it in sync with the programs. In integrated analysis, pooling of study data is dependent on the study design and the metadata of the pooled database. The pooled data is then used for integrated summary reports of safety and efficacy. When preparing documentation files for a submission to an authority it is required to document all derivations of the different variables for each analysis domain (ADSL, ADAE, etc.). Preparing the documentation for the transfer files is often a tedious manual process.

I will present a method of automating this process using predefined tags within SAS programs. The system interprets these tags and combines the metadata with the comments that describe the programmatic derivations to create the documentation. Finally, documentation for the individual domains and studies is consolidated into a single deliverable.

INTRODUCTION TO INTEGRATED DATABASES

Integrated databases of clinical studies contain information collected from more than one clinical study of the same drug within a specific therapeutic area. The integrated databases are used to perform the so-called Integrated Safety Summary (ISS) and Integrated Summary of Efficacy (ISE). The summaries are required for drug approval.

In the process of combining different data sources, individual study data elements often need to be transformed or harmonized so that they represent a consistent or standard context. This transformation must be done without loss of data integrity or interpretation. Combining data from multiple sources yields increased statistical power to identify important relationships between data elements. Without the increased sample size of an integrated database, the relationship between infrequently occurring rare events involving some dependent factor, such as treatment regimen, may be difficult to identify.

To build the integrated database the meta-data structure of the different domains need to be specified to collect the data of different studies (see Fig. 1). The ADaM (Analysis Data Model) data structure is used for the different analysis data sets of the different domains (ADSL, ADAE, ADLB, ADEX, etc.). These analysis data sets are “one step away” from the analysis. When preparing analysis data sets derived variables are prepared for specific analysis of safety and efficacy. New records might be added to the data sets with additional information needed for analysis.

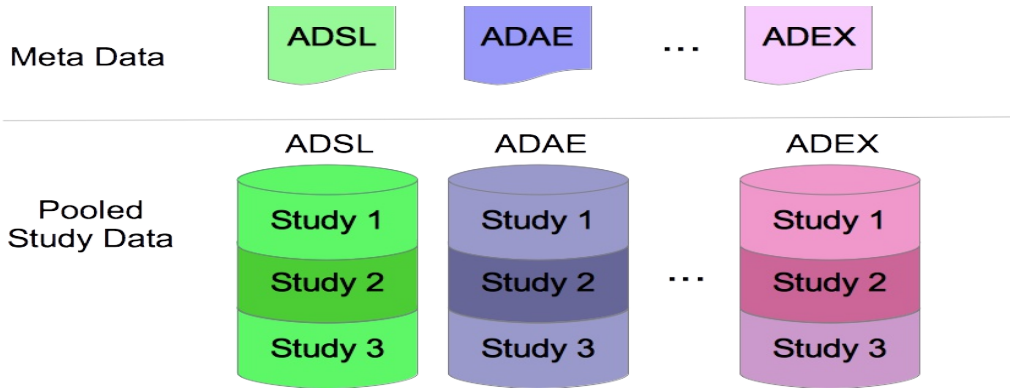

Fig. 1: Integrated database of clinical studies with different domains and meta data

To integrate different clinical studies into the pooled database, transfer or derived programs need to be written. For every domain, a different transfer program is written and will later create one data domain. It will read the source data of the clinical study and will create a harmonized data domain given in the specified meta data structure of the pooled database.

According to ADaM, specific variables are created with a certain naming convention.

After the pooled database is created and all domains for the different studies to be included are prepared by the different transfer programs, the Integrated Summary of Safety (ISS) and Efficacy (ISE) are prepared and the report is submitted to the authorities. The authorities also need the database of the integrated reports and therefore all values from the Case Report Form (CRF) to the pooled database and then the report need to be traced back.
For the submission, descriptions of the pooled database of the different domains are created as so-called Define Documents. They include information about the meta data of the pooled database for each domain and about each variable including the name, label, type, format and length of a variable and how the variable is derived.

The information about the derivations of each variable is usually pre-specified for the integrated database. For the inclusion of the data of the different studies, different derivations also need to be made in order to integrate the data into the pool and to meet the specifications of the meta data and the protocol for the ISS and ISE.
For different clinical studies transfer programs are usually different and the derivations are also different.
The documentation of these derivations and how the data is transferred from the study database (usually in a different structure with own meta data) which does not completely fit with the integrated database needs to be prepared in the Define Document.

The following picture (Fig. 2) shows an excerpt of the documentation for the ADSL domain of a specific study with comments derived automatically from the transfer program by the macro %parse_comments. We will describe the process of parsing comments (%parse_comments) later in this essay.

ADSL Subject-Level Analysis Dataset							
Variable name	**LABEL**	**type AND length**	**format**	**Format decodes**	**Origin**	**Role**	**Comment**
STUDYID	Study Identifier	Char 10	$10.				1000: ADSL.STUDYID
ADSNAME	Dataset Name	Char 8	$ADSNM.	ADSL=ADSL			1000: ADSL.ADSNAME
USUBJID	Unique Subject Identifier	Char 20	$20.				1000: ADSL.USUBJID
AGE	Age	Num 8	3.				1000: ADSL.AGE
AGEU	Age Units	Char 6	$AGEU.	YEARS=YEARS			1000: ADSL.AGEU
BRTH_D	Birth Day	Num 8	2.				1000: BRTH_D is derived from BIRTHDTH
BRTH_M	Birth Month	Num 8	2.				1000: BRTH_M is derived from BIRTHDTH
BRTH_Y	Birth Year	Num 8	2.				1000: ADSL.BRTH_Y is derived from BIRTHDTH
<variable>	<label>	<type>	<format>	<decodes>			<studyid:> <domain>.<variable> [comment/derivation texts]

Fig. 2: Excerpt of documentation of the ADSL domain for specific study using %parse_comments

The last line of the excerpt shows the general structure of the automatically generated comment:

<studyid:> <domain>.<variable> [comment/derivation texts]

The text in the *Comment* column is automatically read by the parser from the transfer program and combined with meta information of a study.
In Fig. 3 an excerpt of a transfer program is displayed. It shows a comment including the name of the derived variable and a derivation text. Additionally, it shows the SAS code of the derivation.

```
/*** BRTH_D is derived from BIRTHDT ***/
 if not(missing(birthdt)) then brth_d=day(birthdt);
 else brth_d=.;


 /*** BRTH_M is derived from BIRTHDT ***/
 if not(missing(birthdt)) then brth_m=month(birthdt);
 else brth_m=.;
```

Fig. 3: Excerpt of a transfer program including comments of specific derivations for the ADSL domain

The preparation of the documentation can be a tedious, manual process. In some cases the data directly comes from the study database and is easy to include into the integrated database. But in other cases, the variables need to be derived. These derivations need to be documented for each and every variable which does not fit to the meta data.
To automate the process of preparing the documentation of the transfer programs for the integrated database we will use standardized SAS comments in the transfer programs. Whenever a variable is derived the programmer should write a comment. Later, every transfer program of each integrated study will be parsed by a SAS macro %parse_comments where the comments will be extracted and processed to prepare the documentation, including the meta data information.

In the following paragraph, the macro %parse_comments is presented and we show how it is used to create the documentation of the integrated database needed for the submission to the authorities. First we will introduce a section about comments in SAS programs and how they will be interpreted and used to comment derivations in the transfer programs.

PARSING COMMENTS IN SAS PROGRAMS

SAS syntax allows three different ways to make a comment in a SAS program. Examples are shown below.

```
/* Comment version 1 */
*  Comment version 2 ;
%* Comment version 3 ;
```

Fig. 4: Comments in SAS programs

Comments could be multi-line comments. See the following example. Later, the macro %parse_comments could be expanded to “understand” multi-line comments and to assign matched variable names and descriptions correctly.

```
/*
 Common multi-line comment style.
 Line 2
 Line 3
 */
```

Fig. 5: Multi-line comments in SAS

The macro %parse_comments will parse the program file. It keeps only the SAS comments and searches for matched variables fitting the meta data. SAS commands are not interpreted by the macro – only SAS comments.

The programmer needs to document all necessary derivations in the comments. An ideal transfer program has comments or descriptions for every variable to be derived. Later, this process simplifies the documentation of the transferred data to the integrated database. After the automated generation of the documentation files, post-processing is possible. The automated generation reduces the amount of time needed to submit the ISE and ISS and additional technical documents to the authorities. It also makes the transfer programs more readable for reviewers and second programmers/reviewers.

Each comment should start with the variable name to be derived followed by a colon and the description of the derivation. A comment could be single-line or multi-line. If a multi-line comment is used, the comment for one variable should end with the <end> tag. Then the description of the next variable could also be included in a multi-line comment.
The following SAS comments show the syntax how a variable of the transfer program should be commented.

```
/* VARIABLE1:<derivation text> */

/* VARIABLE2:derivation text line1
             derivation text line2
             ...
   <end>
   VARIABLE3:derivation text line1
             derivation text line2
             ...
   <end>
*/
```

Fig. 6: Syntax of comments suitable for macro %parse_comments

The macro %parse_comments then goes through all variables of the specific domain and adds the comments from the processed program.

If the variable is already available in the study data, the macro writes just the information of the study ID, the domain and the variable name in the following structure to the define document:

<studyid>: <domain>.<variable name>

The following example shows a variable already available in the study data. In this case it is not necessary to add a comment to the program. The programmer is of course allowed to add also a comment that specific variables are available in the study data and fit to the pool database.

STUDYID	Study Identifier	Char 10	$10.				1000: ADSL.STUDYID

In the example above the variable STUDYID comes from the study domain data set and is not derived differently. The variable STUDYID could be directly transferred to the pool domain.
In the following example the variable BIRTH_Y is not available in the study data, thus it is derived from the birth date variable BIRTHDTH. After macro %parse_comments run, the following text appears in the documentation file including the parsed comment in the last column.

BRTH_Y	Birth Year	Num 8	2.				1000: ADSL.BRTH_Y is derived from BIRTHDTH using SAS year function

The description *"BRTH_Y is derived from BIRTHDTH using SAS year function"* is parsed from the SAS transfer program comment. See the following extract of a transfer file for the ADSL domain.

```
/* BRTH_Y: is derived from BIRTHDT using SAS year function*/
if not(missing(birthdt)) then brth_y=year(birthdt);
else brth_y=.;
```

To include multiple study documentation, the macro adds the study ID: <study ID: domain.variable name> (see for example: *1000: ADSL.STUDYID).* In this case the study ID is 1000. In the current version the macro could only process one study. If you want to combine multiple studies it is possible to combine the data sets later to a single data set and include the resulting data set into the define document. This is an additional step and could be performed after running the %parse_comments macro for a specific study.

In the following paragraph we will introduce regular expressions to search for comments in SAS program code. We will also introduce the macro %parser which is part of the %parse_comments macro.

HOW CAN REGULAR EXPRESSIONS READ COMMENTS

In [1] Ostermiller explains how comments /* */ in typical C-programs are parsed with regular expressions. He started with a simple approach to just read a single comment line.

```
/\*.*\*/
```

/* finds the start of the comment. .* finds any number of any character, and */ finds the end of the expression.
The problem is that this approach does not find comments spanning multiple lines.

In our case, the SAS program code which is read by the %macro parser is stored line by line in the records of a data set. In our approach, the macro %parser cannot parse multiple comment lines directly. We tried to find the start and the end symbols of a comment. When the macro %parser parses a multi-line comment, it looks for a specific variable (of the meta data of the integrated database) and looks for an end tag <end> in order to distinguish between different matched variables within one combined comment. This problem is also displayed in Fig. 6.

We used the following regular expression to find the start and end of a comment.

Start comment:
```
_comment1= prxparse("/\/\*|\*/");
_comment1n=prxmatch(_comment1, trim(line));
```

End comment:
```
_comment2= prxparse("/\*\/|((\*)*;)/");
_comment2n=prxmatch(_comment2, trim(line));
```

After processing comments the macro %parse looks for specific variable names in the comments. This could be performed by the following simple regular expression:

```
pattern=prxparse("/&&_vardomain&i/");
foundmatch=prxmatch(pattern,comment);
```

Later, the retrieved description following the matched variable is added according to the syntax defined previously.

In the following paragraph, we will present the structure of the macro %parse_comments which performs the whole process of reading transfer files, parsing the SAS code and preparing documentation files for the different domains.

IMPLEMENTATION OF %PARSE_COMMENTS MACRO

The %parse_comments macro could parse one or more transfer programs of a single clinical study. The macro consists of the following steps:

1. %readprog(): read transfer files and store code lines in a temporary data set
2. %parser(): parses the code lines and extracts relevant comments/description for each found variable of a domain
3. %getmetadatasstudy: reads the meta data of a study
4. %joinstudymeta(): joins the study meta data with matched variables of the found comments of the transfer file
5. %create_documentation: creates a word document for a domain including meta data of a domain and adds the matched comments of the parser of each variable found in the comment of a transfer file.

In the following Fig. 7 the overview of the parsing process is displayed.

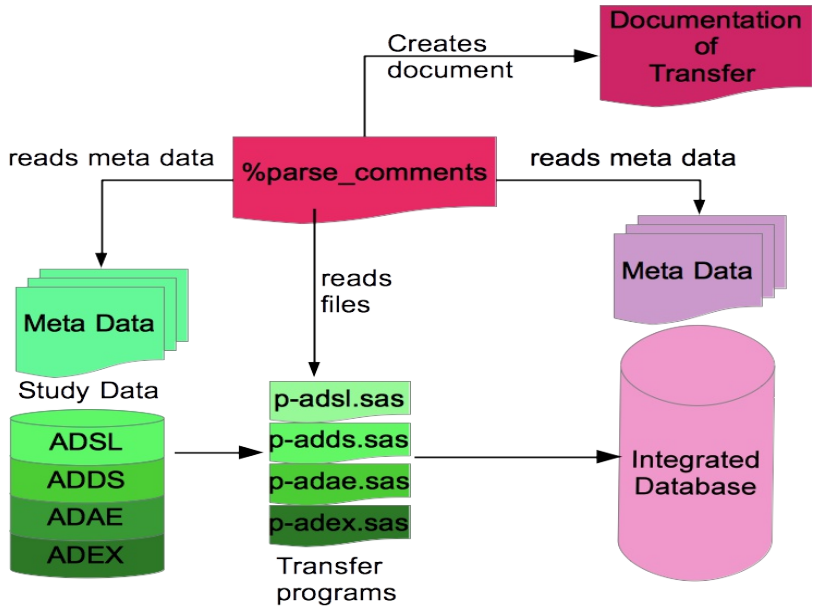

Fig. 7: %parse_comments overview

The following SAS code (Fig. 8) shows the general construction of the %parse_comments macro and its sub-steps. The parser uses a %do-loop to go through all domains in the given directory. It is possible also to run the macro with just one domain.

```
%macro parse_comments(studyid= /*Study ID of the clinical study   */
                      ,domain =/*Domain (transfer file) e.g. <domain|domain1#domain2#...|all> */
                      ,path=   /*path to the transfer files */
                      ,meta=   /*Library to the meta data of the pool */
                      );

   …
```

```
   %do d=1 %to &_dircountdomain;

      %let _domain=&&_dirdomain&d.;

      %readprog(study=&studyid., domain=&_domain, path=&path,outdat=_&studyid._prog&_domain);
      %parser(study=&studyid, metalib=&meta,domain=&_domain);
      %getmetadatastudy(study=&studyid, libstudy=in, domain=&_domain);
      %joinstudymeta(study=&studyid, metalib=&meta, domain=&_domain );
      %create_documentation(study=&studyid, metalib=&meta, domain=&_domain);
   %end;
   …

%mend;
```

Fig. 8: Macro %parse_comments and general construction of the macro

The macro first reads the SAS code of the transfer programs and saves the program code into a temporary data set. This is done by the macro %readprog. Every code line is stored as a record in the temporary data set.

Then the parser starts its work with the help of the macro %parser. The parser goes through the code lines and extracts all the comments.
The meta data of a specific domain of the integrated data base is then read. All variables with name, variable id, label, type, length, format and additional descriptions are read from the meta data set and for each variable a macro variable is created. Later, the %parser macro goes through all macro variables (variables of the meta data) in a %do-loop (Fig. 9) and tries to find a match in the comments of the read transfer files.

```
%do i=1 %to &_varcountdomain.;
  pattern=prxparse("/&&_vardomain&i/");  /*Pattern of a variables*/
  lengthcomment=length(comment);
  if prxmatch(pattern,comment) >0 then do;
    if missing(foundmatch1) then do;
            foundmatch1=prxmatch(pattern,comment);
      foundmatch1c="&&_vardomain&i: "|| compbl(left(substr(comment,foundmatch1,lengthcomment)));
      sasname1="&&_vardomain&i";
   end;
   else do;
      foundmatch2=prxmatch(pattern,comment);
      foundmatch2c="&&_vardomain&i: "|| compbl(left(substr(comment,foundmatch2,lengthcomment)));
      sasname2="&&_vardomain&i";
   end;
  end;
%end;
```

Fig. 9: do-loop of the macro %parse to find match of a variable within a comment

For every match, a record is saved in a temporary data set. The following figure Fig. 10 shows an example of a match.
The left column *LINE* shows the SAS code read from the transfer file. The right column *FOUNDMATCH* shows a match of a found variable within a comment. In this case all code lines are in the data to show also common SAS code between the comments. The parse actually only goes through the extracted comments of a SAS transfer file.

	LINE	FOUNDMATCH1C
34	*** BRTH_Y is derived from BIRTHDT ***;	BIRTHDT: BIRTHDT ***;
35	if not(missing(birthdt)) then brth_y=year(birthdt);	
36	else brth_y=.;	
37	*** BRTH_D is derived from BIRTHDT ***;	BIRTHDT: BIRTHDT ***;
38	if not(missing(birthdt)) then brth_d=day(birthdt);	
39	else brth_d=.;	
40	/*** BRTH_M is derived from BIRTHDT ***/	BIRTHDT: BIRTHDT **
41	if not(missing(birthdt)) then brth_m=month(birthdt);	
42	else brth_m=.;	
43	/***BASEBMI is rounded to 1 decimal place ***/	BASEBMI: BASEBMI is rounded to 1 decimal pl…
44	if not(missing(basebmi)) then basebmi=round(basebmi,0.1);	
45	/*** BMIGR/BMIGRN - derive BMI group from BASEMI***/	BMIGR: BMIGR/BMIGRN - derive BMI group fr…
46	if .<basebmi<= 30 then bmigrn=1;	
47	else if 30<basebmi<=35 then bmigrn=2;	
48	else if basebmi>35 then bmigrn=3;	

Fig. 10: Example of a match of a variable within a comment displayed in a temporary dataset

In the next step the macro %getmetadatastudy reads the meta data of the processed domain of the study. With the help of the macro %joinstudymeta the study meta data is joined with the parsed and matched transfer file by the

variable names. Finally, a report is prepared as Define Document including the parsed comments. If a variable of the study data already fits with the integrated database, this information is also added to the Define Document.

The following figure (Fig. 11) shows details of the %parse_comments macro with its sub-macros and how they access different data sources to generate a documentation file.

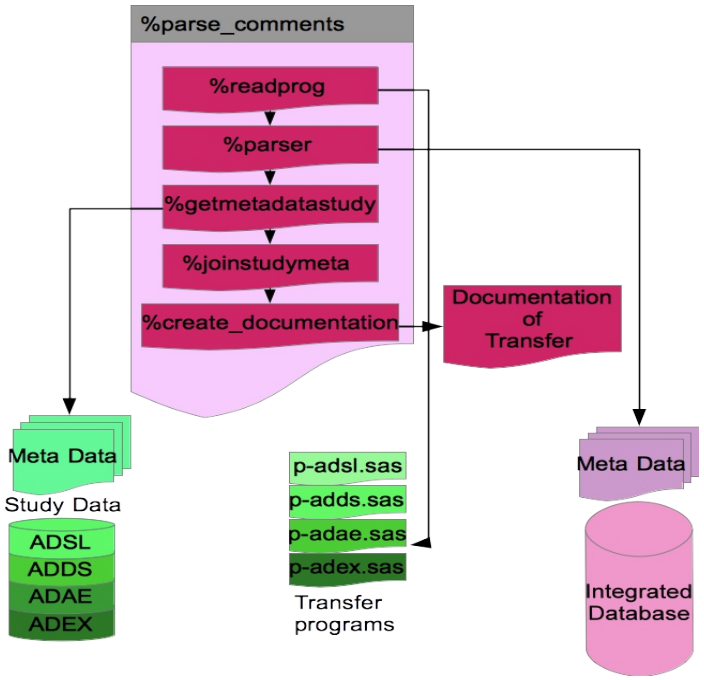

Fig. 11: Sub-macros of macro %parse_comments

RESULTS

We now show some examples how the macro %parse_comments interprets programming comments.

In the first example single-line comments are parsed. The variable BASEBMI is found and added with the description of the comment to the documentation file. The variables BMIGR and BMIGRN appear in one comment. The parser found both variables and adds the same text to the documentation document.

```
/***BASEBMI is rounded to 1 decimal place ***/
 if not(missing(basebmi)) then basebmi=round(basebmi,0.1);

 /*** BMIGR/BMIGRN - derive BMI group from BASEMI***/
      if .<basebmi<= 30 then bmigrn=1;
 else if 30<basebmi<=35 then bmigrn=2;
 else if basebmi>35     then bmigrn=3;
 if not(missing(basebmi)) then bmigr=strip(put(bmigrn,z_bmi.));
```

BASEBMI	Baseline Body Mass Index (kg/m2)	Num 8	5.1	1000: BASEBMI: BASEBMI is rounded to 1 decimal place
BMIGR	Grouped Baseline BMI	Char 40	$40.	1000: BMIGR: BMIGR/BMIGRN - derive BMI group from BASEMI
BMIGRN	Grouped Baseline BMI, N	Num 8	BMI.	1000: BMIGRN: BMIGRN - derive BMI group from BASEMI

In the following example a multi-line comment is parsed and added to the documentation file.

```
/*AEBODSCD (Body System or Organ Class Code) is derived
  from AEMLLT according to SOC_CD codelist
*/
aebodscd=put(aemllt,soc_cd.);
```

AEBODSCD	Body System or Organ Class Code	Char 8	$MEDDRA.	1000: AEBODSCD: AEBODSCD (Body System or Organ Class Code) is derived from AEMLLT according to SOC_CD codelist

CONCLUSION

The documentation of programs is often helpful for reviewers and other programmers to understand the code of the program author. For documentation purposes and with regard to submissions of clinical study reports, including all necessary technical documents is required to show the documentation of the derived data for each variable. The macro %parse_comments is a helpful tool to automate the process of documentation. It reads the SAS source code of transfer files, extracts only comments from the program relating to the variables in the meta data of the integrated database. Finally, it combines the meta data information of each domain in a Word document with additional information about the derivations and descriptions of their sources. This derivation is directly parsed from the commented source code of the transfer file.

A future version of the macro %parse_comments will read multiple studies and combine the descriptions for the different domains into one single document. Currently, the results of the macro %parse_comments could be merged and would show the documentation for multiple studies – see the example below. This is already possible but only one study canbe processed with one macro call.

Variable name	LABEL	type AND length	format	Format decodes	Origin	Role	Comment
<variable>	<label>	<type>	<format>	<decodes>			<studyid 1:> <domain>.<variable> [comment/derivation texts] <studyid 2:> <domain>.<variable> [comment/derivation texts] <studyid 3:> <domain>.<variable> [comment/derivation texts] ...

All SAS programs mentioned are available from the author on request.

REFERENCES

[1] http://ostermiller.org/findcomment.html: Finding Comments in Source Code Using Regular Expressions

ACKNOWLEDGMENTS

I would like to thank Colin de Klerk (Principal Statistical Programmer, inVentiv Health Clinical) for his great support and review of this manuscript.

CONTACT INFORMATION

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

Thomas Wollseifen
InVentiv Health Germany GmbH
Grosse Hub 10d
Eltville am Rhein / 65344
Work Phone: +49 (0) 6123 70437-10
Email: thomas.wollseifen@inventivhealth.com
Web: www.inventivhealth.com