PharmaSUG 2024 - Paper DS- 188

Automated Harmonization: Unifying ADaM Generation and Define.xml through ADaM Specifications

Wei Shao and Xiaohan Zou, Bristol Myers Squibb

ABSTRACT

In electronic submission packages, ADaM datasets and Define.xml stand as pivotal components. Ensuring consistency between these elements is critical. However, despite the importance of this, current methods still heavily depend on manual checks. To address this challenge, we introduce an innovative automated approach driven by ADaM specifications. Our solution involves a suite of SAS® macros engineered to streamline the translation from ADaM specification to both ADaM datasets and Define.xml. These macros orchestrate a seamless automation process, facilitating the generation of ADaM datasets while concurrently fortifying consistency between ADaM datasets and Define.xml. The automated processes include format creation, core variables addition, variable attributes generation, dynamic length adjustment based on actual values, and automatic ADaM specification updates from actual data. These macros act as dynamic tools, constructing datasets with precision, adjusting variable attributes, and most importantly, syncing Define.xml with actual data. Our automated tool system not only expedites ADaM datasets creation, but also ensures an inherent consistency with Define.xml. This amalgamation of automation and specification-based integrity significantly reduces manual errors, enhances data quality, and fortifies the efficiency of the submission process.

INTRODUCTION

ADaM specifications serve as comprehensive guidelines, delineating critical details essential for ADaM generation, and encompassing variable names, labels, lengths, data types, codelists, origins, derivations, and more. At our company, collaboration with PINNACLE[21](P21E) has yielded the ADaM Define Adaptor, a bespoke Define.xml generator tailored to extract Define.xml directly from ADaM specifications. This adaptation underscores the profound role of ADaM specifications not only as the bedrock for creating ADaM datasets, but also as the singular source for Define.xml generation. This pivotal role forms the foundation of our automation initiatives, rooted within the ADaM specification framework. In this paper, we introduce an innovative approach centered on automating ADaM datasets and Define.xml generation, aimed at ensuring seamless consistency between the two. Notably, our work features the development of automated ADaM tools adept at retrieving metadata such as core variables, variable attributes, and codelists directly from the ADaM specification. These tools seamlessly integrate this metadata into ADaM programming, facilitating swift and accurate ADaM dataset generation. Moreover, our utility empowers the dynamic adjustment of variable lengths in the specification based on actual ADaM data. Embracing this pioneering approach, we guarantee metadata consistency between Define.xml and ADaM datasets, amplifying accuracy and efficiency during regulatory submissions.

PROCESS FLOW FOR GENERATING ADAM DATASETS AND DEFINE

This section describes the process flow for creating ADaM datasets and Define.xml starting from ADaM specifications (Figure 1)

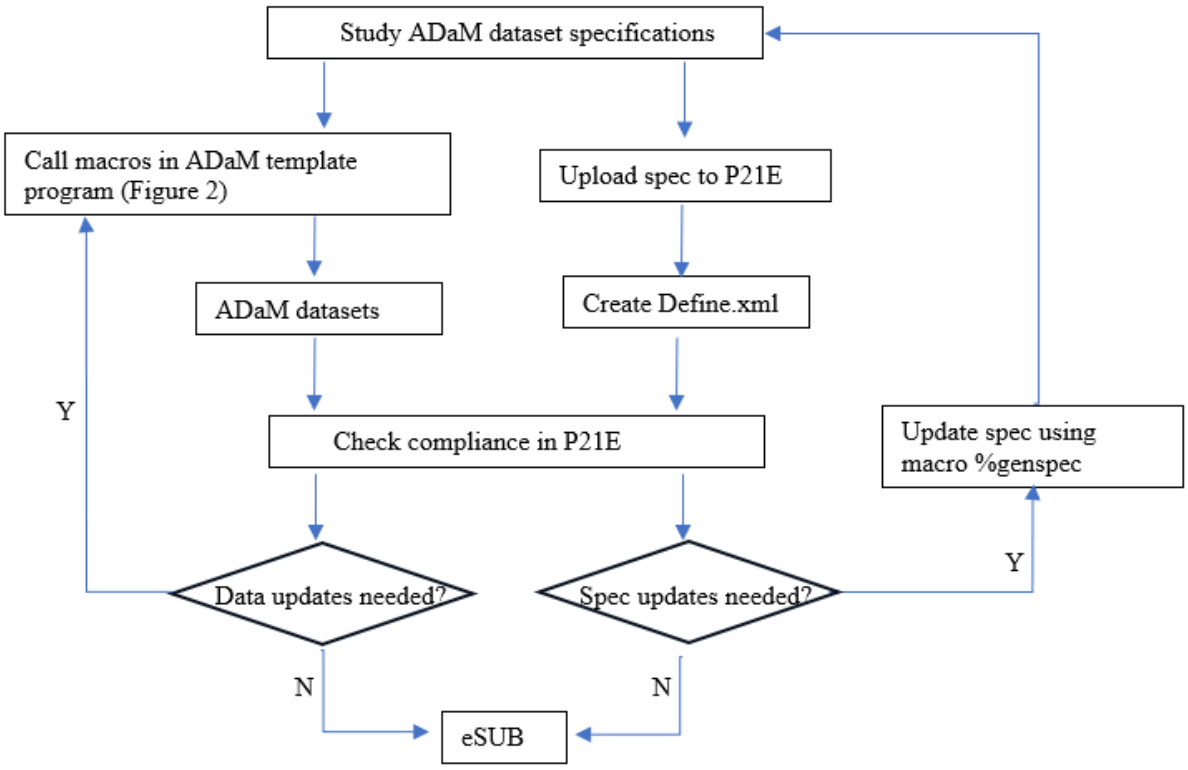

Figure 1 Overview of Process Flow for Creating ADaM Datasets and Define

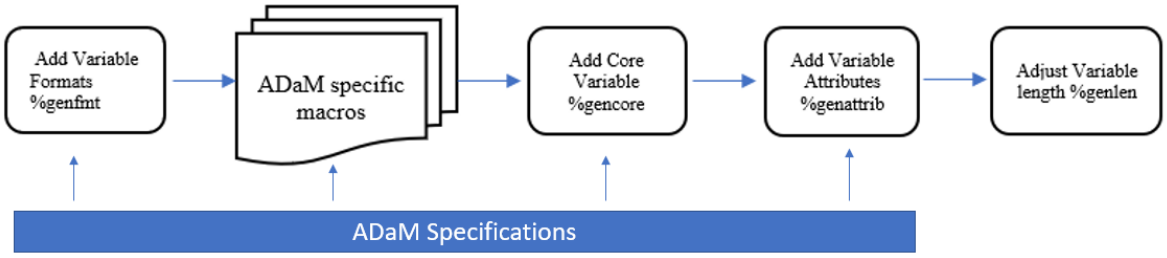

Figure 2 Code Pattern for ADaM Datasets

The ADaM generation workflow commences with ADaM specifications, initiating automation through an ADaM template program, which is ADaM programming in a structured and modular approach. Our ADaM template program comprises general macros applicable across multiple ADaM domains and domain-specific macros tailored to individual domains, as depicted in Figure 2. Notably, the framework for ADaM dataset generation hinges on 5 pivotal general macros, summarized in Display 1. Following ADaM dataset creation, a conformance check is performed within the P21E system. Any detected issues within the datasets or specifications prompt necessary actions to update the datasets or refine specifications to rectify these issues.

Similarly, the workflow for Define.xml generation originates from ADaM specifications. We employ our company's ADaM specifications template to develop study-level ADaM specifications, and these specifications are processed in P21E to generate Define.xml. Noteworthy in our toolkit is the incorporation

of a data-driven macro designed to dynamically adjust variable lengths based on actual data, enabling revisions in specifications for Define.xml creation. The iterative process involves multiple checks of the ADaM data package, including Define.xml, until all identified issues are resolved.

Since the automation of consistency checking is conducted from the beginning of ADaM programming to the end for regulatory submission, the high quality of submissions can be achieved in a cost-effective and efficient way.

Automation	Macro Name	Purpose	Macro call instance
1	%genfmt	Create formats and informats based on codelists (terms and decodes) from ADaM specifications	%genfmt (inspec = &spec., adsname = adsl);
2	%gencore	Retrieve the list of core variables from ADaM specifications and populate them into ADaM domains	%gencore (inspec = &spec., adslib = adam, indset = adae, outdset = adae);
3	%genattrib	Extract variable attributes (names, labels, lengths, formats) from ADaM specifications and assign them to the variables in ADaM domains	%genattrib (inspec = &spec., adslib = adam, adsname = adsl, adslabel=%str(Subject-Level Analysis Dataset));
4	%genlen	Trim the length of all character variables to the maximum length of the text string for each variable in ADaM domains	%genlen (adslib = adam, adsname = adsl);
5	%genspec	Update the Spec by adjusting length based on ADaM domains	%genspec(inspec = &spec., adslib = adam, outspec = &outspec.);

Display 1 SAS® Macros in ADaM Automation

AUTOMATION TOOLS EMPLOYED FOR ADAM CREATION AND SPECIFICATION UPDATES

AUTOMATION 1: CREATE FORMATS AND INFORMATS FROM THE SPECIFICATIONS

The terminology for variables in ADaM datasets is described in the specifications. There are 2 codelists sources in the ADaM specifications: Variable Metadata (Display 2) and Codelist (Display 3) tabs. The term / decode values can be entered in either tab, but not both. Typically, it's advisable to keep repeated or lengthy codelists in the "Codelist" tab. The macro %genfmt is used to extract term/decode from both tabs in the specifications and create formats and informats when a codelists is applicable to a variable. For instance, let's look at the variable AGEGR1N (pooled age group 1(N)). When both term and decode are provided, the equal sign (=) is used as a separator. It's important to note that we put a space before and after the equal sign (=). In this case, the portion before the equal sign, '1' or '2' represents term, while the portion after the equal sign, "<65" or ">=65" is the decode. If the code or decode value contains a character string with "=" sign (e.g. "<=", ">="), do not add black spaces between characters to ensure that the "=" sign is interpreted as text, and not as a separator.

Dataset Name	Parameter Identifier	Parameter Identifier Variable	Variable Name	Variable Type	Variable Length	Variable Display Format	Codelist ID	Term/Decode
ADSL	*ALL*		COUNTRY	text	200		COUNTRY	
ADSL	*ALL*		AGEGR1	text	5		AGEGR1	< 65, >= 65
ADSL	*ALL*		AGEGR1N	integer	8		AGEGR1N	1 = < 65, 2 = >=65

... Dataset Metadata | Variable Metadata | Codelist | Codelist Name | Dictionary | Documents | Analysis Displa ...

Display 2 Term/Decode at Variable Metadata Sheet in The Spec

Codelist ID	Term	Decode
COUNTRY	ARG	ARGENTINA
COUNTRY	AUS	AUSTRALIA
COUNTRY	AUT	AUSTRIA
COUNTRY	BRA	BRAZIL
COUNTRY	BEL	BELGIUM

Cover Page | Info Page | Dataset Metadata | Variable Metadata | Codelist

Display 3 Codelist Sheet in The Spec

The macro %genfmt is used to create dual formats from the specifications. The macro call is shown as follows:

```
%genfmt(inspec=   /* the specification location */
        ,adsname= /* analysis dataset name  */
        );
```

The sample SAS® codes have the following major steps:

Step 1: Extract the codelists from the spec using PROC IMPORT

```
proc import file= "&INSPEC " dbms=xlsx out=_var replace;
     sheet="Variable Metadat";
     getnames=YES;
run;
```

Step 2: Split term and decode for codelist in "Variable Metadata" sheet

```
data _var2;
  set _var(keep=dataset_name variable_name variable_type term_decode
           rename=(dataset_name=DATASET variable_name=VARIABLE));
  vtype=input(upcase(variable_type), $vartyp.);
  nterms=countw(term_decode, '0D0A'x);
  length term decode $200;
  if index(term_decode, ' = ') >0  then
   do i=1 to nterms;
    this_term_decode=scan(term_decode, i, '0D0A'x);
    split=find(this_term_decode, ' = ');
    term=substr(this_term_decode, 1, split-1);
    decode=tranwrd(substr(this_term_decode, split+3),',',',');
     output;
   end;
  keep dataset variable vtype term decode;
run;
```

Step 3: Create dual formats for each variable using the following SAS® codes.

- SAS® code for creating formats: START = term; LABEL = decode,
- SAS® code for creating informats: START = decode; LABEL = term.

After executing the above codes, the intermediate dataset (See Display 4 for an example of what this dataset looks like) will be generated, encompassing both format and informat for variable 'RACEN'. Users can select and utilize either based on theirs needs.

VARIABLE	START	LABEL
RACEN_FMT	1	WHITE
RACEN_FMT	2	BLACK OR AFRICAN AMERICAN
RACEN_FMT	3	AMERICAN INDIAN OR ALASKA NATIVE
RACEN_FMT	4	ASIAN
RACEN_FMT	5	NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER
RACEN_FMT	6	UNKNOWN
RACEN_FMT	7	NOT REPORTED
RACEN_INFMT	WHITE	1
RACEN_INFMT	BLACK OR AFRICAN AMERICAN	2
RACEN_INFMT	AMERICAN INDIAN OR ALASKA NATIVE	3
RACEN_INFMT	ASIAN	4
RACEN_INFMT	NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER	5
RACEN_INFMT	UNKNOWN	6
RACEN_INFMT	NOT REPORTED	7

Display 4 Formats and Informats Dataset

Step 4: Create formats and informats dataset using cntlin in the FORMAT procedure

```
proc format cntlin = _final;
run;
```

AUTOMATION 2: ADD CORE VARIABLES TO ADAM DATASETS

The FDA advises to populate a set of basic subject level variables to all analysis datasets. These variables are called core variables. Core variables can include study ID, site ID, country, region, sex age, race ethnicity, treatment assignment, analysis population flags, and other important baseline characteristic variables. They will be identified from ADSL and populated into all analysis datasets. Display 5 shows that dataset name of core variables in the specifications is *ALL*.

Dataset Name	Parameter Identifier	Parameter Identifier Variable	Variable Name	Dataset Keys Order	Variable Label	Variable Type	Variable Length	Variable Display Format	Codelist ID
ALL	*ALL*		COUNTRY		Country	text	200		COUNTRY
ALL	*ALL*		COUNTRYL		Country Long Name	text	200		COUNTRYL
ALL	*ALL*		REGION1		Geographic Region 1	text	200		REGION
ALL	*ALL*		REGION1N		Geographic Region 1 (N)	integer	8		REGIONN
ALL	*ALL*		AAGE		Analysis Age	integer	8		
ALL			AAGEU		Analysis Age Units	text	200		AGEU
ALL	*ALL*		ASEX		Sex Decode	text	200		ASEX

Cover Page | Info Page | Dataset Metadata | Variable Metadata | Codelist | Codelist Name | Dictio ...

Display 5 Core Variables at Variable Metadata Sheet in The Spec

The macro %gencore serves the purpose of merging analysis datasets with ADSL to get core variables. The code for this process is straightforward, primarily involving the merging of ADSL with other ADaM datasets by the unique subject identifier (USUBJID) to populate subject-related information.

The macro call of %gencore is shown below:

```
%gencore(inspec= /* the specification location */
```

```
          ,adslib= /* analysis data library that has ADaM domains */
          ,indset=/* Name of the input dataset */
          ,outdset=/* Name of the output dataset */
          );
```

%gencore macro is comprised of three parts:

Part1: Import the specifications with the following SAS® code:

```
proc import file="&INSPEC" dbms=xlsx out=_var replace;
  sheet="Variable Metadata";
  getnames=YES;
run;
```

Part2: Create a macro variable “&corevar” that contains the list of core variables separated by a space

```
 proc sql noprint;
   select variable_name into: corevar separated by ' '
   from _var(where=(dataset_name=upcase("*ALL*")));
 quit;
```

Part3: Merge ADSL with other ADaM datasets by USUBJID to populate core variables

```
data &outdset.;
  merge &indset.(in=main) adsl(keep= STUDYID USUBJID &corevar.)
  by STUDYID USUBJID;
  if main;
run;
```

AUTOMATION 3: GENERATE VARIABLE ATTRIBUTES FROM THE SPECIFICATIONS

Every variable in an ADaM dataset is defined by attributes including label, length, display format. The macro %genattrib is used to retrieve this information from ADaM specifications, and then assign them to each variable in the ADaM dataset. The macro call of %genattrib is shown as follows:

```
%genattrib(inspec=   /* the specification location */
           ,adslib=  /* analysis dataset location */
           ,adsname= /* analysis dataset name */
           ,adslabel=/* analysis dataset label */
           );
```

Basically, the first step of the macro is to import the specification and extract variable attributes from variable metadata sheet. Then we would rename metadata as follows: variable_name to varnm, variable_label to varlb, variable_type to var_typ, variable_length to varlen, variable_display_format to varfmt. After this step, we can then use the following SAS® program to create a temporary variable named ‘allvar’ in intermediated dataset ‘_attrn’:

```
data _attrn;
  set _metadata4;
   length allvar $200.;
   if var_typ = 'text' then allvar = 'attrib ' ||compress(varnm)||' label =
"'||strip(varlb)||'"'||' length = '|| '$'||varlen||';';
   else if var_typ = 'integer'  then do;
     if varfmt = '' then allvar = 'attrib ' ||compress(varnm)||' label =
"'||strip(varlb)||'"'||' length = '||varlen||';';
     else allvar = 'attrib ' ||compress(varnm)||' label =
"'||strip(varlb)||'"'||' length = '||varlen||' format =
```

```
'||compress(varfmt)||'.'||';';
    end;
     else if var_typ = 'float' then do;
       if varfmt = '' then allvar = 'attrib ' ||compress(varnm)||' label =
"'||strip(varlb)||'"'||' length = '||varlen||';';
        else allvar = 'attrib ' ||compress(varnm)||' label =
"'||strip(varlb)||'"'||' length = '||varlen||' format =
'||'20.'||compress(varfmt)||';';
     end;
 run;
```

Next, we would create the macro variable &varall using the INTO statement in PROC SQL. The example of the value for macro variable "&varall" is following SAS® codes:

```
proc sql noprint;
    select allvar into :varall separated by ' ' from _attrn;
quit;
```

```
attrib COUNTRY label = "Country" length = $200;
attrib COUNTRYL label = "Country Long Name" length = $200;
attrib REGION1 label = "Geographic Region 1" length = $200;
attrib REGION1N label = "Geographic Region 1 (N)" length = 8  ;
attrib AAGE label = "Analysis Age" length = 8  ;
attrib AAGEU label = "Analysis Age Units" length = $200;
attrib ASEX label = "Sex Decode" length = $6  ;
```

Finally, put the macro variable &varall in the data step to add variable attributes.

```
data &adsname.;
 &varall.;
 set &adsname.;
run;
```

This macro avoids manually defining variable attributes for all the variables in all datasets, and therefore significantly reducing programming workload and human errors. It guarantees the consistency between the ADaM datasets and the specifications. Additionally, apart from assigning variable attributes, the macro also sorts data by key variables specified in the specifications, only containing all variables defined in the requirement specification, and ordering variables within the dataset as defined in the specifications. This macro is especially useful to ensure that the variable attributes are adjusted or updated in accordance with any changes made to the specifications. Running this macro is the only action required, which enhances cost-effectiveness.

AUTOMATION 4: TRIM THE LENGTH OF CHARACTER VARIABLES

In general, we assign the maximum possible length ($200) for the character variables in the specifications when maximum length for the variables is unknown. Setting all character variables to a length of 200 increases the size of a dataset, making it difficult for reviewers to work on. In addition, setting lengths larger than the actual value can lead to P21 issues (SD1082) 'Variable length is too long for actual data'. Therefore, it's necessary to reduce variable lengths in ADaM datasets creation.

To avoid truncation of values in the data process, our strategy involves utilizing a SAS® macro %genlen as a final step within each ADaM program. The primary objectives of this macro are outlined below:

- Reduce the length of character variables.
- Maintain the original length of numeric variables.
- Preserve the original order of variables within the dataset.

- Provide flexibility to choose the set of character variables for preserving lengths, such as predecessor variables.
- Conduct a comprehensive comparison to ensure that the reduced-size datasets remain identical to the original dataset, except for variable length modifications.

The macro call syntax:

```
%genlen ( adslib =   /* analysis dataset location */
         ,adsname=/* analysis dataset name */
         );
```

The implementation of %genlen macro is comprised of three key steps:

Step 1: Create the following macro variables using PROC CONTENTS

- n_char: number of character variables
- oriname1 - oriname&n_char: character variables
- allvar: all variables

```
proc contents data=&adslib..&adsname. out=_content1 varnum noprint;
run;
proc sql noprint;
  select compress(put(count(*), best.)) into :n_char from _content1 where
name not in (&excluvars.) and type=2;
  select name into :oriname1-:oriname&n_char from _content1 where name not
in (&excluvars.) and type=2;
  select name into:allvar separated by ' ' from _content1 order by varnum;
quit;
```

Step 2: Loop through the variables using PROC SQL and get the maximum length for each variable

```
proc sql;
  create table _mlength as
   select %do i=1 %to &n_char;
     max(length(&&oriname&i)) as &&oriname&i
     %if &i=&n_char %then %str();
     %else %str(,);
   %end;
   from &adslib..&adsname.;;
quit;

proc transpose data=_mlength out=_mlength_v;
     var _all_;
run;
```

Step 3: Reset the variable lengths in each dataset and ensure trimmed data has no anticipated changes

```
data _null_;
  set _mlength_v;
   call symputx(compress("charlen"||put(_N_, best.)), _name_||"
$"||strip(put(col1, best.)));
run;

data &adslib..&adsname.
     %if %length(&datasetlabel) %then (label="&datasetlabel");;
     retain &allvar;
     length
```

```
      %do i=1 %to &n_char;
       &&charlen&i
      %end;;
   set &adslib..&adsname.;
      informat _character_;
      format _character_;
run;
```

In order to allow maximum flexibility but minimal effort, we decided to reduce variable lengths via a SAS® macro as a post-processing step, which would determine in a dynamic fashion the maximum length needed for each variable based on the actual values present in the dataset. By taking this approach, we would not have to continuously monitor new data coming in to check for longer values and repeatedly update programming specifications and programs.

AUTOMATION 5: UPDATE THE SPECIFICATION BASED ON ACTUAL DATA

As mentioned previously, we assign the maximum potential length (up to $200) for each character variable in our ADaM specifications initially and then allow the macro %genlen to adjust the variable lengths as the final step within each ADaM program. Following database lock, we execute the %genspec macro to update variable lengths in the specifications based on actual data. The %genspec macro is typically called at the late stage of ADaM programming cycle to finalize the specifications. The call of macro %genspec is shown as the follows:

```
%genspec(inspec=  /* input specification */
         ,adslib= /* ADaM datasets location */
         ,outspec=/* output specification */
         );
```

The partial SAS® codes in %genspec macro is used to extract variable length from ADaM datasets.

Step 1: Get the list of datasets

```
data domainlist;
  set sashelp.vtable;
     where upcase(libname)=upcase("&adslib");
      keep memname memlabel order;
run;
```

Step 2: Create macro variables of domains (&domain1, &domain2 ...) and number of domains (&numdomain) that will be used in the next step

```
data_null_;
 set domainlist end=eof;
 call symputx(compress("domain"||put(_N_, best.)), memname);
 if eof then call symputx("numdomain", _N_);
run;
```

Step 3: Retrieve variable length from each dataset using a do...loop statement

```
%do nd=1 %to &numdomain;
  proc contents data=&adslib..&&domain&nd out=content2 noprint;
  run;
%end;
```

After obtaining the variable lengths for all variables from ADaM datasets, we will utilize this to update the values of variable length in the specifications. Later we will create Define.xml in P21E by directly importing the ADaM specifications to P21E.

CONCLUSION

In summary, ADaM dataset and define.xml are all generated from the programming specification. This paper introduces 5 automations tools to utilize the specifications for creating ADaM datasets and define file. The SAS® codes presented in this paper are just examples and there are other methods that have been publicly shared as well. We encourage you to research and evaluate various methods and consider what will work best with your process. This macro-based comprehensive approach not only maximizes efficiency but also champions the principle of the data consistency, integrity, and regulatory compliance.

REFERENCES

ADaM Implementation Guide (ADaM-IG v1.3)
https://www.cdisc.org/system/files/members/standard/foundational/ADaMIG_v1.3.pdf

PINNACLE21 ADaM Validation Rules https://www.pinnacle21.com/validation-rules/adam

FDA Study Data Technical Conformance Guide https://www.fda.gov/media/153632/download

CDISC Define-XML https://www.cdisc.org/standards/data-exchange/define-xml

ACKNOWLEDGMENTS

The authors would like to thank our manager Allison Covucci for her great support and valuable input of this paper.

CONTACT INFORMATION

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

Wei Shao
Bristol Myers Squibb
wei.shao2@bms.com

Xiaohan Zou
Bristol Myers Squibb
xiaohan.zou@bms.com