PharmaSUG 2017 - Paper AD04

A Vivid and Efficient Way to Highlight Changes in SAS Dataset Comparison

Jeff Xia, Merck & Co., Inc., Rahway, NJ, USA;

Lugang (Larry) Xie, Merck & Co., Inc., Rahway, NJ, USA

Shunbing Zhao, Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

How to find the updates/changes across periodic data extractions in an ongoing clinical trial is a very interesting topic. Many authors have papers discussing limitations and gray areas in dataset comparison using conventional methods. This paper presents a method to identify and highlight changes/updates in both the metadata and contents level of two corresponding datasets. Newly added variables or records, as well as the removed/updated ones are identified by a SAS® macro utility, and highlighted and color coded in an Excel spreadsheet. It provides a vivid and efficient way for SAS® programmers, data managers and statisticians to master every detail of the updates in datasets of different versions.

INTRODUCTION

As statistical SAS® programmers working on CSR tables/listings/graphs and NDA submission packages, we periodically receive data extractions of ongoing clinical trials from data management group, or external data from central labs, PK/PD groups, etc. Sometimes we even receive database updates after the database has been locked for a while: database unlock/relock does happen in many cases. To estimate the impact on our work cycle driven by these source data changes, it is an essential task for us to understand what the changes are in the data in a comprehensive manner, on both the metadata level and subject data contents level.

The SAS® procedure Proc Compare is widely used in the industry for dataset comparison. It is very popular among SAS® users because of its simplicity in syntax and usage. However, this procedure does have many limitations and blind spots. Joshua Horstman and Roger Muller (2016) presented a few interesting examples when using Proc Compare to compare the content of the datasets and their metadata, and concluded that the procedure output “NOTE: No unequal values were found. All values compared are exactly equal” could be very misleading in some circumstances.

The other drawback of the procedure Proc Compare is its lengthy output if mismatches do exist. It is very time consuming and labor extensive to find the information of interest, or pinpoint the exact changes in the dataset. This paper provides an efficient method to identify changes in the datasets, and presents the information of interest in a vivid way to draw reviewer’s attention.

DISCUSSION

Let‘s use an example to facilitate the discussion. We had a database which was locked a few months ago. All the tables/listings/graphs for CSR have been finalized for a while. Suddenly the data management group informed the team that the database had to be unlocked to enter a few SAE that were missed during the conduct of the trial, as well as some medical coding (MedDRA) updates based on further medical review. A few days later, we received the updated database in SDTM format. Since the timeline for CSR finalization was approaching, it became extremely critical for us to fully understand these changes in data, and be able to answer the following questions before we re-ran all CSR tables:

- Have the expected changes correctly made in the updated database?
- Are there any other unexpected changes in the updated database?

In order to answer these tough questions, we need to perform a thorough data comparison in SDTM datasets we received and their previous version. Firstly, we need to find out whether there are any changes in the metadata level, which include any previously existing variables got dropped? Any newly added variables? Any changes in the dataset label, variable label, variable type, length, format or

informat? These changes might affect SDTM compliance per the FDA published data checking rules, or critical variables that need to be incorporated in statistical analysis. Secondly, we have to find out what the changes in the data contents level are, including any records got removed from the dataset? Any records newly added to the dataset? How many records without any changes? For these records with changes in certain variable(s), what are these variables? What is the new value and what is the corresponding old value? Are these changes expected?

We have designed a list of macros to explore the data changes further and answer these questions step by step. The first macro is to rename all variables in a given dataset except these key variables. Key variables will be used for dataset merge between the two corresponding datasets, i.e., USUBJID, Sequence number carried from operational database, etc. The second macro is to compare the datasets and identify these changes in metadata and contents level, and record the change information in a variable pair ("VARLIST" and "STATUS") in the final dataset for output. The third macro is to produce a MS excel spreadsheet, and display the comparison results from the second macro in a vivid way by highlighting the updates in different colors.

1. MACRO RENAME.SAS

This macro %rename finds the name of all variables in a dataset, and then renames each one of these variables in a systematical way. Any variables specified in the macro variable "except" will keep their names without change; otherwise, the new name would be "OLD_" plus its original name, i.e., the variable "AESTDTC" in the dataset becomes "OLD_AESTDTC" after calling this macro. See below for logic in details, where macro parameter "lib" and "ds" are the library and input dataset name, "except" is the macro parameter for a list of variables that the user would like to keep their names without change.

```
proc sql noprint;
    select count(*) into :n
    from sashelp.vcolumn
    where libname=upcase("&lib") and
          memname=upcase("&ds");
    select distinct(name) into :var1-:var&n
    from sashelp.vcolumn
    where libname=upcase("&lib") and
    memname=upcase("&ds");
quit;
proc datasets library=&lib nolist;
    modify &ds;
    rename
    %do i=1 %to &n;
        %if %index(%upcase(&except), %upcase(&&var&i)) eq 0 %then
        %do;
            &&var&i=OLD_&&var&i.
        %end;
    %end; ;
quit;
```

2. MACRO COMPARE_DATASET.SAS

Firstly, this macro finds out the variable list in both the new and old datasets by using the procedure Proc Contents, and then it changes the variable name of the old dataset except the variable "name" by using the macro %rename, and then merges them using the key variable "name".

```
proc contents data=_&ds._new
              out=_var_new(keep = memname memlabel name
                                  label format length type varnum)
              noprint order=collate;
run;
proc contents data=_&ds._old
              out=_var_old(keep = memname memlabel name
```

```
                                    label format length type varnum)
            noprint order=collate;
run;
%rename(ds =_var_old, except = name);
data _metadata1;
   merge _var_new(in = a) _var_old(in = b);
   by name;
   length msg $200;
   if a and not b then msg = 'New variable added';
   else if b and not a then msg = 'Variable removed';
   else do;
       common = 'Y';
       if TYPE ne OLD_TYPE then msg = 'Variable type changed';
   end;
   if common ne 'Y' then common = 'N';
run;
```

Then the macro defines a list of macro variables to record results of the comparison by using the procedure Proc SQL, see sample code below:

Macro Name	Description
final_keep	List of variables to keep in the final output dataset, including all variables in the new dataset, and any variables in the old dataset but not in the new dataset.
common_var_c	List of character variables in the new dataset that we would like to compare with the old dataset.
common_var_n	List of numeric variables in the new dataset that we would like to compare with the old dataset.
common_var_c_old	List of character variables in the old dataset that we would like to compare with the new dataset.
common_var_n_old	List of numeric variables in the old dataset that we would like to compare with the new dataset.
in_new_only	List of variables in the new dataset but not the old dataset.
in_old_only	List of variables in the old dataset but not the new dataset.

Table 1. List of macro variables for recording comparison results.

```
proc sql noprint;
   select strip(name) /*char variable names in new dataset*/
   into: final_keep
   separated  by ' '
   from _metadata1
   where common ='Y' or msg = 'Variable removed' ;

   select strip(name) /*char variable names in new dataset*/
   into: common_var_c
   separated  by ' '
   from _metadata1
   where type = 2 and
         index( "&keyvars", upcase(strip(name))) = 0 and
         common ='Y';
```

```
    /*similar way to populate the rest macro variables in the
      following tables */
 quit;
```

2.1 PROCESS INFORMATION ON THE METADATA LEVEL

The next step is to separate the variables existing in both datasets from these of newly added or removed ones.

```
data _metadata1_1 _metadata1_2;
   set _metadata1;
   if msg in ('New variable added','Variable removed')
         then output _metadata1_2;
   else output _metadata1_1;
run;
```

And then find out any changes in the metadata level for these common variables in both datasets. Create another variable “varList” to record the list of variables with change in metadata, i.e., dataset label, variable label, type, length, and format, etc.

```
data _metadata1_1;
   set _metadata1_1;
   length varList $100;
   if _N_ = 1 and memlabel ne OLD_memlabel then do;
     msg = strip(msg)||' Dataset label;';
         varList = strip(varList)||' MEMLABEL';
   end;
   if label ne OLD_label then do;
     msg = strip(msg)||' Variable label;';
         varList = strip(varList)||' LABEL';
     end;
/*similar way to check updates in variable type, length, format, etc.*/
 run;
```

Finally put each record in metadata into one of the following four categories: “No Change”, “Updated”, “Added”, or “Removed”, and record this information in the variable “STATUS”.

```
data finalMeta ;
    label status = 'Status'
          varlist = 'Updated Variable List'
          memname = 'Dataset Name'
          memlabel = 'Dataset Label'
          name = 'Variable Name'
          label = 'Variable Label'
          type = 'Variable Type'
          length = 'Variable Length'
          format = 'Variable Format'
          varnum = 'Variable Order'
          msg = 'Note';
    set _metadata1_1(in = a) _metadata1_2(in = b);
    length status $10;
    memname = upcase("&ds");
    if a then do;
        if msg >'' then do;
            status = 'Updated';
            output;
```

```
            status = 'Old';
            varlist = '';
            memlabel = Old_memlabel;
            label = Old_label;
           type = Old_type;
           length = Old_length;
           format = Old_format;
           varnum = Old_varnum;
           output;/*create another record with the values in Old dataset*/
       end;
       else do;
           status = 'No Change';
           output;
       end;
   end;
   else if b then do;
       if msg = 'New variable added'  then status = 'Added';
       else if msg = 'Variable removed' then status = 'Removed';
       output;
   end;
run;

proc sort data = finalMeta(keep = status varList memname memlabel name
                                  label type length format varnum msg);
   by varnum;
run;  /*sorting for output at mete data level*/
```

2.2 PROCESS INFORMATION ON CONTENTS LEVEL

Similar way is used to process the contents level changes. Firstly, duplicate the old dataset for further process, and then rename all the variable names in the duplicated dataset except key variables, then merge it with the new dataset to identify whether a record is “Added”, “Removed”, or “Matched”.

```
data _&ds._old1;
    set _&ds._old
       %if %length(&typeChanged) >1 %then drop = &typeChanged;);;
run;
%rename(ds = _&ds._old1, except = &keyvars);
data _merged;
    merge _&ds._new(in = a
                %if %length(&typeChanged) >1 %then drop = &typeChanged;)
          _&ds._old1(in = b );
    by &keyvars;
    length status $10;
    if a and not b then status = 'Added';
    else if b and not a then status = 'Removed';
    else status = 'Matched';
run;
```

For these records with a status “Matched”, first we build two arrays C1 and C2, in which array C1 for common character variables in the new dataset, and array C2 for the character variables in the old dataset. Then go through each paired variables using a do loop and compare their values one by one, if any mismatch found, record the variable name in the variable “VarList” by using the SAS® function vname.

```
data _merged1;
    set  merged;
    length VarList   $400;
    %if %length(&common_var_c) ne 0 %then %do;
```

```
            array C1[*] &common_var_c;
            array C2[*] &common_var_c_old;
            if status = 'Matched' then do;
                do i = 1 to dim(C1);
                      if strip(C1[i]) ne strip(C2[i]) then do;
                            VarList = strip(VarList)||' '
                                    ||strip(vname(C1[i]));
                      end;
                end;
            end;
    %end;

    /*Build separate array N1 and N2 for numeric variabless*/
    if status = 'Matched' then do;
       if compress(VarList) >'' then status = 'Updated';
       else status = 'No Change';
    end;
run;

data _updated(keep = &keyvars);
    set _merged1 (where = (status in ( 'Updated')));
run;

data _old_update; /*Get the old values for variables with update*/
    merge _&ds._old(in = a ) _updated(in = b);
    by &keyvars;
    if a and b;
run;

data final; /*Include the records with old value in the final output*/
    length status $10;
    set _merged1(in = a where = (status ne 'Removed')) _old_update(in = b);
    if a then do;
       sort = 1;
    end;
    else if b then do;
       sort = 2;
       status = 'Old';
    end;
run;

data _removed(keep=&keyvars);/*Get the removed records from old dataset*/
    set _merged1 (where = (status in ( 'Removed')));
run;

data _old_removed;
    merge _&ds._old(in = a) _removed(in = b);
    by &keyvars;
    if a and b;
run;

data final_contents; /*Include the removed records in the final output*/
    set final(in = a) _old_removed(in = b);
    if b then do;
       sort = 3;
       status = 'Removed';
    end;
```

```
run;

%if %length(&inOldOnly) ne 0 %then %do;
   %let finalKeep = InOldOnly &finalKeep ;
   data final_contents;
      set final_contents;
      length InOldOnly $100;
      /*the variable InOldOnly is added for the purpose of
          highlighting in MS Excel spreadsheet*/
      InOldOnly = strip("&InOldOnly");
   run;
%end;

%if %length(&inNewOnly) ne 0 %then %do;
   %let finalKeep = InNewOnly &finalKeep ;
   data final_contents;
      set final_contents;
      length InNewOnly $100;
      /*the variable InNewOnly is added for the purpose of
        highlighting in MS Excel spreadsheet*/
      InNewOnly = strip("&InNewOnly");
   run;
%end;
```

For these records with updated values in certain variables, there will be two separate records in the final output for contents level change: the one with new value, and the other one with old value. To make sure the record with new value always appears on top of the one with old value, perform the following sorting by using the procedure Proc Sort.

```
proc sort data = final_contents;
    by &keyvars sort;
run;
```

3. MACRO DATA2XML.SAS

The SAS® feature of ExcelXP tagset is used to convert the dataset into a MS Excel spreadsheet with color highlights. The following code describes the logic in details. The macro variable “rptLoc” in below code stands for the location of the output, and “rptName” for the output file name.

```
ods html close;
ods listing close;
ods tagsets.excelxp file = "&rptLoc.\&rptName..xml" style = minimal;
ods tagsets.excelxp
       options (sheet_name = "&sheetname"
                embedded_titles = 'no'
                absolute_column_width = "&_columnwidth"
                autofilter = 'yes'
                zoom = '100'
                orientation='landscape'
                row_repeat = 'header'
                pages_fitheight = '100'
                center_horizontal = 'yes'
                center_vertical = 'no'
                autofit_height = 'yes'
                frozen_headers = 'yes');
```

If no column width is specified, SAS® ExcelXP tagset produces the output spreadsheet using the variable length as the default value. For some character variables with long strings, the column could be too wide

to fit the screen, which is very inconvenient for reviewers when looking for the whole value. The following logic has been used to reduce the width of these columns with long strings.

```
proc contents data = &indsn
              out = _indsn_contents(keep =name type length label varnum)
              varnum noprint;
run;

proc sort data = _indsn_contents;
    by varnum;
run;

data _indsn_contents;
   set _indsn_contents;
   name = upcase(name);
   if length >30 then length = 15;
   else if length <12 then length = 6;
   else length = length/2;   /*fine tune column width*/

   /*the following code puts the variable STATUS, VARLIST,
     INNEWONLY and INOLDONLY in the front of variable list*/
   if name = 'STATUS' then myorder = 1;
   else if name = 'VARLIST' then myorder = 2;
   else if name = 'INNEWONLY' then myorder = 3;
   else if name = 'INOLDONLY' then myorder = 4;
   else myorder = 100;
run;

proc sort data = _indsn_contents;
   by myorder varnum;
run;

proc sql noprint;
    %if %length(&width) = 0 %then %do; /*if no width is specified*/
       select length into :_columnwidth separated by ','
       from _indsn_contents;
    %end;
    %else %do; /*if width is specified for each column*/
       %let _columnwidth = &width;
    %end;
       select upcase(name) into :_columnvar separated  by ' '
       from _indsn_contents;
quit;

%do i=1 %to %length(&_columnvar);  /*find out how many variables*/
    %if %scan(&_columnvar, &i) ne %str() %then %do;
       %let _columnvartot=&i;
    %end;
%end;
%do i=1 %to &_columnvartot;
   /*assign the name of each variable to a macro variable*/
    %let _columnvar&i = %scan(&_columnvar, &i);
%end;
```

Ideally, the final output will not include the auxiliary variables such as VARLIST, INNEWONLY or INOLDONLY, the following code loops through the variable list and define them as “noprint”.

```
proc report data = &indsn nofs
     style(header)={font_weight=bold font_size=10pt just=center
                    protectspecialchars=off};
   column &_columnvar;
   %do i=1 %to &_columnvartot;
   /*do not include variable VARLIST INNEWONLY and INOLDONLY in output*/
       %if %upcase(&&_columnvar&i) = VARLIST or
           %upcase(&&_columnvar&i) = INNEWONLY or
           %upcase(&&_columnvar&i) = INOLDONLY %then %do;
          define &&_columnvar&i /noprint;
       %end;
       %else %do;
          define &&_columnvar&i /display;
       %end;
 %end;
```

The following code defines the background color of a given row based on the value in the variable STATUS. If the value is ‘OLD’ then set the color to dark gray; if the value is ‘REMOVED’ then set the color to green; if the value is ‘ADDED’ then set the color to yellow; otherwise no color is set, so the cell will be clear with no background color.

```
 %if %index(%upcase(&_columnvar), STATUS) > 0 %then %do;
     compute Status;
       if upcase(strip(Status)) in ('OLD') then do;
          call define(_row_,'style','style={background=gray}');
       end;
       else if upcase(strip(Status)) in ('REMOVED') then do;
          call define(_row_,'style','style={background=cxff6633}');
       end;
       else if upcase(strip(Status)) in ('ADDED') then do;
          call define(_row_,'style','style={background=yellow}');
       end;
     endcomp;
%end;
```

If a variable name is included in the variable “INNEWONLY’, which means it is a newly added row, the following code sets the background to yellow; alternatively if a variable name is included in the variable “INOLDONLY”, which means it is a variable that has been removed from the new dataset, then the background color will be set to green. For all the other common variables, the background color will be set to red if it is included in the value of the variable VARLIST.

```
%if %index(%upcase(&_columnvar), VARLIST) gt 0 %then %do;
    %let i = 1;
    %let tmp = %upcase(%scan(&_columnvar, &i));
    %do %while (&tmp ne %str());
      %if  %upcase(&tmp) ne STATUS and
           %upcase(&tmp) ne VARLIST and
           %upcase(&tmp) ne INNEWONLY and
           %upcase(&tmp) ne INOLDONLY  %then %do;
        compute &tmp;
          if findw(upcase(Varlist), upcase(strip("&tmp"))) gt 0 then do;
            call define("_c&i._",'style','style={background=red}')  ;
          end;
      %if %index(%upcase(&_columnvar), INNEWONLY) gt 0 %then %do;
           else if findw(upcase(InNewOnly), upcase(strip("&tmp"))) gt 0
            then do;
            call define(_col_,'style','style={background=yellow}')  ;
```

```
            end;
        %end;
        %if %index(%upcase(&_columnvar), INOLDONLY) gt 0 %then %do;
            else if findw(upcase(InOldOnly), upcase(strip("&tmp"))) gt 0
              then do;
              call define(_col_,'style','style={background=green}')  ;
            end;
        %end;
           endcomp;
 %end;
   %let i=%eval(&i + 1);/*process the next variable in the variable list*/
   %let tmp = %upcase(%scan(&_columnvar, &i));
%end;
run;
```

Type of changes	Color	Color Sample
Old value	Dark gray	
Updated value	Red	
Value with no change	Clear	
Removed record or column	Green	
Added record or column	Yellow	

Table 2. Color schema of the output MS Excel spreadsheet

4. SAMPLE OUTPUT

Below is a sample output of a comparison of dataset AE. We can see clearly that there are some changes on the metadata level. The dataset label was updated form ‘AE’ to ‘Adverse Event’, variable format ‘BEST’ was dropped from the new dataset; the SDTM expected variable AEENDTC was dropped from the new dataset, which might cause SDTM compliance issues. Furthermore, the variable AESEV was added into the new dataset.

	A	B	C	D	E	F	G	H	I	J
1	Status	Dataset Name	Dataset Label	Variable Name	Variable Label	Variable Type	Variable Length	Variable Format	Variable Order	Note
2	Updated	AE	Adverse Event	STUDYID	Study Identifier	2	255		1	Dataset label;
3	Old	AE	AE	STUDYID	Study Identifier	2	255		2	Dataset label;
4	No Change	AE	Adverse Event	DOMAIN	Domain Abbreviation	2	2		2	
5	No Change	AE	Adverse Event	USUBJID	Unique Subject Identifier	2	255		3	
6	Updated	AE	Adverse Event	AESEQ	Sequence Number	1	8		4	Variable format;
7	Old	AE	AE	AESEQ	Sequence Number	1	8	BEST	4	Variable format;
8	No Change	AE	Adverse Event	AESPID	Sponsor-Defined Identifier	2	255		5	
9	No Change	AE	Adverse Event	AETERM	Reported Term for the Adverse Event	2	255		6	
10	No Change	AE	Adverse Event	AEDECOD	Dictionary-Derived Term	2	255		7	
11	Added	AE	Adverse Event	AESEV	Severity/Intensity	2	8		8	New variable added
12	No Change	AE	Adverse Event	AESER	Serious Event	2	20		9	
13	No Change	AE	Adverse Event	AESTDTC	Start Date/Time of Adverse Event	2	19		10	
14	Removed	AE		AEENDTC		.	.		10.5	Variable removed

Display 1. Output of Excel spreadsheet of changes in metadata level

	A	B	C	D	E	F	G	H	I	J	K
1	Status	Study Identifi	Domain Abbreviation	Unique Subject Identifi	Sequence Numbe	Sponsor-Defined Identifie	Reported Term for the Adverse Event	Dictionary-Derived Term	Serious Event	Start Date Time of Adverse Event	End Date Time of Adverse Eve
6	No Change	V212-001	AE	SAMPLE-0001-0001	3.6872E+14	4	Mucositis	Mucosal inflammation	N	2012-08-08	
7	Added	V212-001	AE	SAMPLE-0001-0001	3.6872E+14	5	Edema in Extremities	Oedema peripheral	N	2012-08-09	
8	Updated	V212-001	AE	SAMPLE-0001-0001	3.6872E+14	6	Diarrhea-Intermittent	Diarrhoea	N	2012-08-11	
9	Old	V212-001	AE	SAMPLE-0001-0001	3.6872E+14	6	Diarrhea-Intermittent	Candida infec	N	2012-08-11	2012-08-31
14	No Change	V212-001	AE	SAMPLE-0001-0001	3.6872E+14	11	Hypomagnesemia	Hypomagnesaemia	N	2012-08-15	
15	Removed	V212-001	AE	SAMPLE-0001-0001	3.6872E+14	12	Intermittent Hypophosphatemia	Hypophosphataemia	N	2012-08-13	2012-12-04

Display 2. Output of Excel spreadsheet of changes in contents level

The changes on the contents level are also easy to see. The whole column of the variable AEENDTC was highlighted in green because it has been removed from the new dataset. Another column AESEV was highlighted in yellow (not captured in the display above because of screen size); the 7th row was highlighted in yellow since it was a new record, and the 15th row was highlighted in green since it was removed from the new dataset. The Dictionary Derived Term in the 8th row has been updated to "Diarrhoea" unexpectedly, which was highlighted in red to draw reviewer's attention. The corresponding old value can be found in the next row, and the entire row was highlighted in dark gray to remind reviewers that this row came from the old dataset, and was listed here just for reviewer's convenience in the process of data review.

5. FUTURE IMPROVEMENT

The output of the SAS® ExcelXP tagset is in the format of XML, which can be opened in MS Excel with full features. Compared with native Excel spreadsheet, the output in XML format has some drawbacks, such as its big file size, more time needed in opening it using MS Excel. SAS® version 9.4 provides a new feature of Excel destination, which can be used to generate the output in native Excel spreadsheet format (.xlxs). This paper still uses ExcelXP tagset for the purpose of back compatibility since there are significant amount of SAS® 9.3 users in the company. After all SAS® users migrate to SAS® 9.4, this macro can be re-written by utilizing Excel destination, therefore significantly reduces the output file size.

CONCLUSION

The method presented in this paper can produce dataset comparison output in MS Excel spreadsheet with color-coded highlights, which helps to draw reviewer's attention to the exact updates/changes between the new and old datasets. The highlighted changes displayed in the context of data listing make it easier for reviewer to understand the difference between the new and old values. We used this method to perform data check for database re-run, SDTM up-versioning from 3.1.1 to 3.1.3, etc. Compared with conventional data comparison tools such as the procedure Proc Compare, this tool significantly reduces the amount of manual check/data lookup, and greatly improves the efficiency in data review.

REFERENCES

Roger Muller and Josh Horstman, *2016.* Don't Get Blindsided by PROC COMPARE, WUSS Proceedings

Wendy Boberg, 2008. PROC REPORT in Color ... What's Your STYLE?, SAS® Global Forum 2008

Dylan Ellis, 2014. Absolute_Pixel_Width? Taming Column Widths in the ExcelXP Tagset. SAS® Global Forum 2014

ACKNOWLEDGMENTS

The authors would like to thank Cynthia He for her great support and valuable input of this paper.

CONTACT INFORMATION

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

Name: Jeff Xia
Enterprise: Merck
Address: 126 E. Lincoln Avenue
City, State ZIP: Rahway, NJ 07065-4607
Work Phone: 732-594-6439
Fax:
E-mail: jeff.xia@merck.com
Web: www.merck.com

Name: Lugang Xie (Larry)
Enterprise: Merck
Address: 126 E. Lincoln Avenue
City, State ZIP: Rahway, NJ 07065-4607
Work Phone: 732-594-1527
E-mail: lugang.xie@merck.com
Web: www.merck.com

Name: Shunbing Zhao
Enterprise: Merck
Address: 126 E. Lincoln Avenue
City, State ZIP: Rahway, NJ 07065-4607
Work Phone: 732-594-3976
E-mail: shunbing.zhao@merck.com
Web: www.merck.com