### Paper CC34

# The Mystery of Automatic Retain in a SAS® Data Step

Huei-Ling Chen, Merck & Co., Inc., Kenilworth, NJ USA
Hong Zhang, Eli Lilly and Company, Bridgewater, NJ
Patricia Guldin, Merck & Co., Inc., Kenilworth, NJ USA

## ABSTRACT

The data step is the most frequently used programming process in the SAS System. As programmers we should be very familiar with it. However, sometimes we write a piece of code, but the output is not our expectation. Is our code incorrect or are there mysteries inside the data step? This paper will focus on one of the mysteries - automatic retain in a data step. We will investigate how variables are automatically retained yet no retain statement is specified. Examples are provided to demonstrate the pitfalls one can experience when constructing a data step. Being cautious can avoid unexpected results. This paper uses a PUT _ALL_ statement to demonstrate how automatic retain variables can be retained.

## KEYWORDS

RETAIN, AUTOMATIC RETAIN, PUT _ALL_ STATEMENT, IF-THEN/ELSE STATEMENT, DATA STEP

## AUTOMATIC RETAIN

The RETAIN statement is a useful statement in a SAS DATA step. People use it to carry over values across observations, perform calculations, re-order the variables, and some other possible uses. Plenty of user guides or help documents are available online to provide SAS users with the fundamental understanding on the use of RETAIN statements with clear examples and explanations.

Unlike the RETAIN statement, the concept of automatic retain is less recognized and understood by SAS users. There are variables not coded in a RETAIN statement but whose values are still being automatically retained. Howard (2005) observes that during the SAS data step, four types of variables are retained: variables in the RETAIN statement, SAS special automatic variables, variables read with a SET, MERGE, or UPDATE statement, and accumulator variables in a SUM statement. Except the variables specified in the RETAIN statement, all other three types of variables are automatic retained.

- SAS special automatic variables
- Variables read with a SET, MERGE or UPDATE statement
- Accumulator variables in a SUM statement

This paper focuses on one of the automatically retained variables, variables read with a SET, MERGE or UPDATE statement. The PUT _ALL_ statement is utilized to illustrate how an automatic retained mechanism can cause unexpected results.

## EXAMPLE 1

Dataset A has variables ID, X, and Y. Dataset B has variables ID, X, but not Y.

Dataset A

| ID | X | Y |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 2 | 3 |
| 3 | 1 | 4 |

Dataset B

| ID | X |
|---|---|
| 4 | 1 |
| 5 | 2 |
| 6 | 2 |

Datasets, A and B, are to be set together. In addition, variable Y will be replaced with a new value 9 when variable X equals to 1, otherwise Y remain the same value.

Expected New Dataset

| ID | X | Original Y | New Y |
|---|---|---|---|
| 1 | 1 | 2 | 9 |
| 2 | 2 | 3 | 3 |
| 3 | 1 | 4 | 9 |
| 4 | 1 | . | 9 |
| 5 | 2 | . | . |
| 6 | 2 | . | . |

An initial coding approach is to combine the datasets using a simple set statement and carry out the manipulation in one single DATA step. The logic is straightforward and the following code seems to be sufficient to carry out the task.

**< Incorrect Coding >**

```
data C;
    set A B;
    if X=1 then Y=9;
run;
```

Surprisingly, the output dataset is not what we expect.

Dataset C from above code

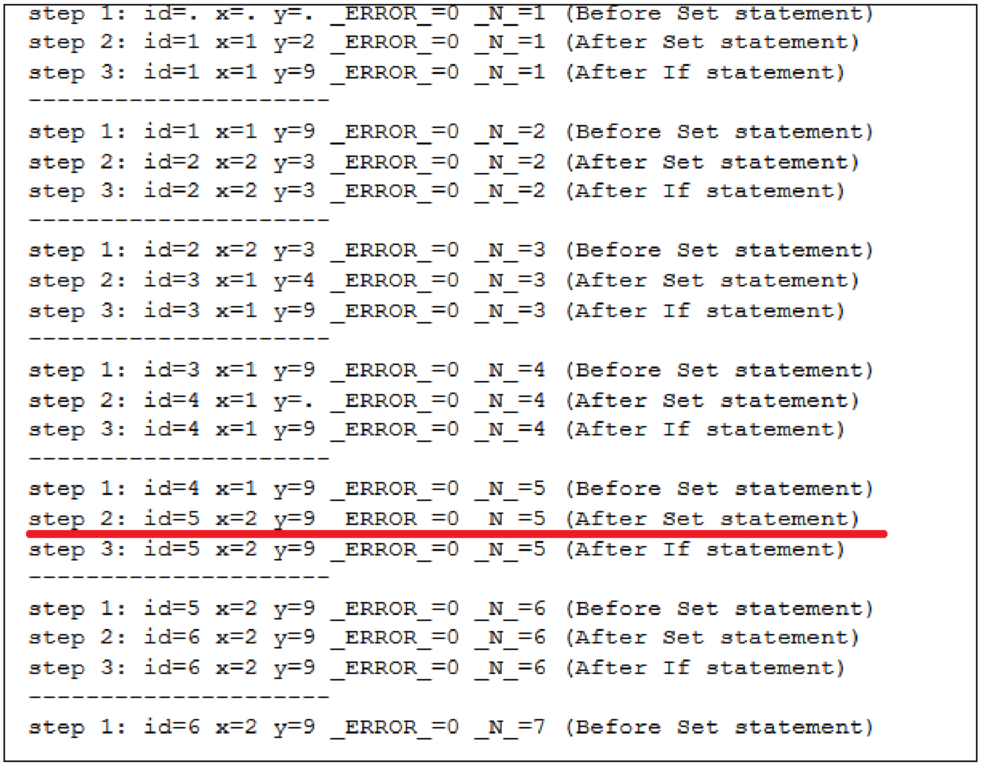
| ID | X | Original Y | New Y |
|---|---|---|---|
| 1 | 1 | 2 | 9 |
| 2 | 2 | 3 | 3 |
| 3 | 1 | 4 | 9 |
| 4 | 1 | . | 9 |
| 5 | 2 | . | **9** |
| 6 | 2 | . | **9** |

Notice that the ID 5 and 6 records supposed to have variable Y with a missing value now are filled with value 9. Where does the '9' come from?

**Use PUT _ALL_ Statement to Debug**

```
data C;

    put 'step 1: ' _all_ '(Before Set statement)';

    set A B;

    put 'step 2: ' _all_ '(After Set statement)';

    if X=1 then Y=9;

    put 'step 3: ' _all_ '(After If statement)' ;
    put '---------------------';

run;
```

**Log Output**



```
step 1: id=. x=. y=. _ERROR_=0 _N_=1 (Before Set statement)
step 2: id=1 x=1 y=2 _ERROR_=0 _N_=1 (After Set statement)
step 3: id=1 x=1 y=9 _ERROR_=0 _N_=1 (After If statement)
---------------------
step 1: id=1 x=1 y=9 _ERROR_=0 _N_=2 (Before Set statement)
step 2: id=2 x=2 y=3 _ERROR_=0 _N_=2 (After Set statement)
step 3: id=2 x=2 y=3 _ERROR_=0 _N_=2 (After If statement)
---------------------
step 1: id=2 x=2 y=3 _ERROR_=0 _N_=3 (Before Set statement)
step 2: id=3 x=1 y=4 _ERROR_=0 _N_=3 (After Set statement)
step 3: id=3 x=1 y=9 _ERROR_=0 _N_=3 (After If statement)
---------------------
step 1: id=3 x=1 y=9 _ERROR_=0 _N_=4 (Before Set statement)
step 2: id=4 x=1 y=. _ERROR_=0 _N_=4 (After Set statement)
step 3: id=4 x=1 y=9 _ERROR_=0 _N_=4 (After If statement)
---------------------
step 1: id=4 x=1 y=9 _ERROR_=0 _N_=5 (Before Set statement)
step 2: id=5 x=2 y=9 _ERROR_=0 _N_=5 (After Set statement)
step 3: id=5 x=2 y=9 _ERROR_=0 _N_=5 (After If statement)
---------------------
step 1: id=5 x=2 y=9 _ERROR_=0 _N_=6 (Before Set statement)
step 2: id=6 x=2 y=9 _ERROR_=0 _N_=6 (After Set statement)
step 3: id=6 x=2 y=9 _ERROR_=0 _N_=6 (After If statement)
---------------------
step 1: id=6 x=2 y=9 _ERROR_=0 _N_=7 (Before Set statement)
```

The log above illustrates how SAS compiles and executes the data step by step.

- The first record
    - At step 1: Variables are initialized to missing at the beginning.
    - At step 2: SAS reads in the record. Variables ID, X, and Y are overwritten by new values when the first observation is read by the SET statement.
    - At step 3: SAS execute the IF statement. The variable Y is replaced with value 9, as the criteria of the IF statement, X equals 1, is satisfied.

- The second record
    - At step 1: Rather than being reinitialized to missing, the values are retained from the previous execution. The second observation here hence shows ID=1 X=1 Y=9, the value of first observation.
    - At step 2: SAS reads in the record. Variables ID, X, and Y are overwritten by new values when the second observation is read by the SET statement. The observation here is renewed to ID=2 X=2 Y=3.
    - At step 3: SAS execute the IF statement. The criteria of the IF statement is not met. The variable Y stays with value 3.

This process goes well and repeats till the fifth record.

- The fifth record
    - At step 1: the values are retained from the previous execution. The fifth observation here hence shows ID=4 X=1 Y=9, the value of fourth observation.
    - At step 2: SAS is supposed to replace the value with the read in value. Variables ID and X are overwritten by new values ID=5 and X=2. Notice that there is no variable Y from dataset B to be read. Variable Y cannot be replaced. Hence, the value of variable Y remained as 9 which is a retained value from previous record.
    - At step 3: SAS execute the IF statement. The criteria of the IF statement is not met. The variable Y stays with value 9.

## Suggested Solutions

Often programmers advocate efficient coding. One perception is that less coding steps is efficient. This is true only when the output result is correct. In the example above, the SAS code should be written more carefully.

One solution is to split the single DATA step into two DATA steps. First step is to stack the datasets. And carry out the deriving task in the second DATA step.

**< Correct Coding >**

```
data D;
    set A B;
run;

data E;
    set D;
    if X=1 then Y=9;
run;
```

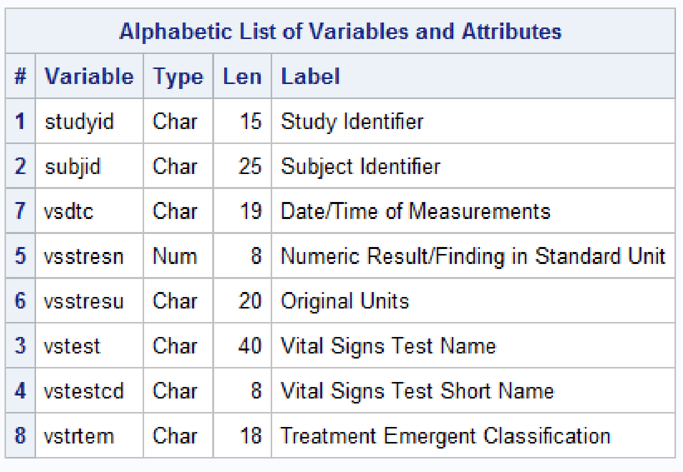
## EXAMPLE 2 – PRE-SPECIFIED VARIABLE ATTRIBUTES

The SDTM and ADaM standards are designed to support submission by a sponsor to a regulatory agency. To further help sponsors implement these standards, the implementation guides, SDTMIG and ADaMIG, were created. Both implementation guides specify standard dataset structures, variables, dataset naming conventions, variable values algorithm, etc. When creating SDTM and ADaM datasets, variables should comply with these pre-specified standard attributes such as label, format, and length.

One approach is to create an empty dataset equipped with the same variables and attributes such as length, label, and format. Then the empty dataset can be appended to the existing datasets, this way the pre-specified variable attributes will be passed on to the existing datasets.

Here this paper takes an example from a clinical trial dataset and demonstrates how the automatic retained mechanism may produce an unexpected output.

**An Empty Dataset DEFINE_VS with the Required Attributes (PROC CONTENTS Output) referenced from the SDTMIG implementation guide**



Alphabetic List of Variables and Attributes

| # | Variable | Type | Len | Label |
|---|---|---|---|---|
| 1 | studyid | Char | 15 | Study Identifier |
| 2 | subjid | Char | 25 | Subject Identifier |
| 7 | vsdtc | Char | 19 | Date/Time of Measurements |
| 5 | vsstresn | Num | 8 | Numeric Result/Finding in Standard Unit |
| 6 | vsstresu | Char | 20 | Original Units |
| 3 | vstest | Char | 40 | Vital Signs Test Name |
| 4 | vstestcd | Char | 8 | Vital Signs Test Short Name |
| 8 | vstrtem | Char | 18 | Treatment Emergent Classification |

Dataset DEFINE_VS

| STUDYID | SUBJID | VSDTC | VSSTRESN | VSSTRESU | VSTEST | VSTESTCD | VSTRTEM |
|---|---|---|---|---|---|---|---|

Raw Dataset VITAL (present essential variables RFSTDTC VSDTC RFENDTC only)

| RFSTDTC | VSDTC | RFENDTC |
|---|---|---|
| 2013-09-24T13:53 | 2013-10-22T10:11 | 2014-01-14 |
| 2013-09-24T13:53 | 2014-02-11 | 2014-01-14 |
| 2013-09-24T13:53 | 2013-08-27T10:44 | 2014-01-14 |
| 2013-09-24T13:53 | 2013-11-21T10:40 | 2014-01-14 |
| 2013-09-24T13:53 | 2014-01-15T11:00 | 2014-01-14 |

An SDTM domain VS (Vital Sign) will be created based on a raw dataset called VITAL. An empty dataset (dataset DEFINE_VS) is appended on the top of the raw dataset (VITAL) in order to preserve the pre-specified variable attributes.

Assume that a new variable, VSTRTEM, which not existing in the raw dataset VITAL, needs to be derived in the DATA step. Variable VSTRTEM is flagged with value 'Y' when variable VSDTC is inside the boundary of variable RFSTDTC and RFENDTC.

Expected Output Dataset

| RFSTDTC | VSDTC | RFENDTC | VSTRTEM |
|---|---|---|---|
| 2013-09-24T13:53 | 2013-10-22T10:11 | 2014-01-14 | Y |
| 2013-09-24T13:53 | 2014-02-11 | 2014-01-14 | |
| 2013-09-24T13:53 | 2013-08-27T10:44 | 2014-01-14 | |
| 2013-09-24T13:53 | 2013-11-21T10:40 | 2014-01-14 | Y |
| 2013-09-24T13:53 | 2014-01-15T11:00 | 2014-01-14 | |

The following simple code may initially seem correct to carry out this algorithm.

**< Incorrect Coding >**

```
data vs;
    set define_vs vital;
    if rfstdtc <= vsdtc <= rfendtc then vstrtem = 'Y';
run;
```

However, the above code produces this unexpected output.

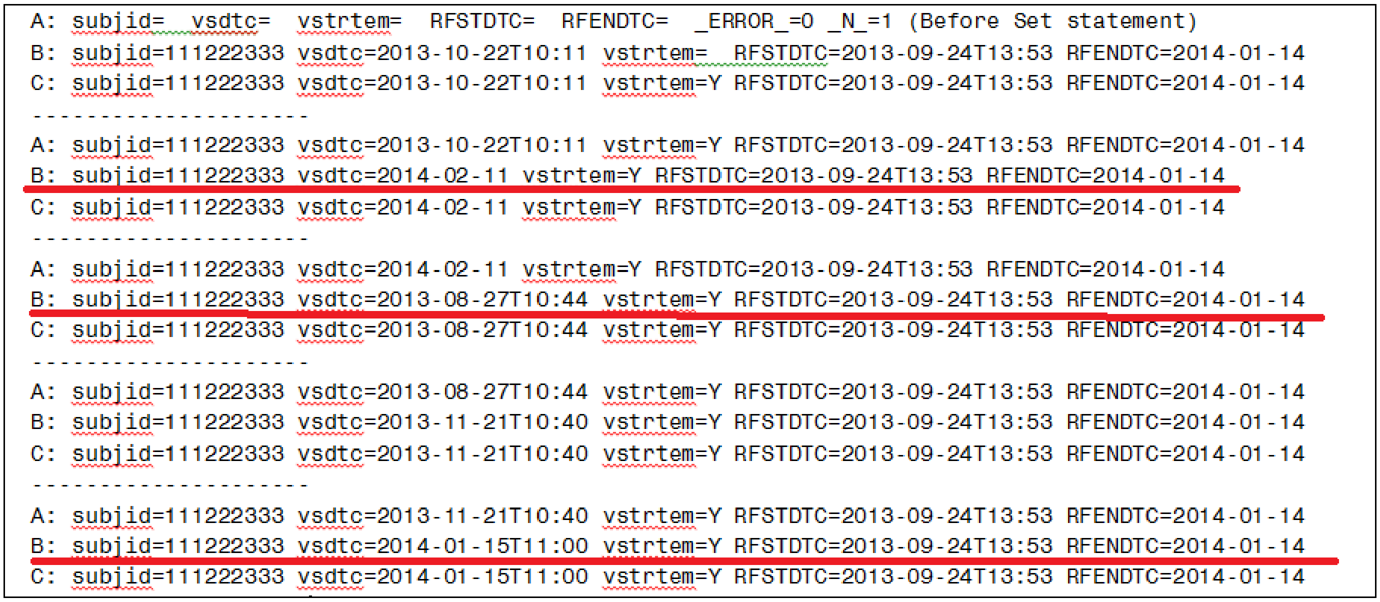
**Unexpected Output**

| RFSTDTC | VSDTC | RFENDTC | VSTRTEM |
|---|---|---|---|
| 2013-09-24T13:53 | 2013-10-22T10:11 | 2014-01-14 | Y |
| 2013-09-24T13:53 | 2014-02-11 | 2014-01-14 | **Y** |
| 2013-09-24T13:53 | 2013-08-27T10:44 | 2014-01-14 | **Y** |
| 2013-09-24T13:53 | 2013-11-21T10:40 | 2014-01-14 | Y |
| 2013-09-24T13:53 | 2014-01-15T11:00 | 2014-01-14 | **Y** |

### Use PUT _ALL_ Statement to Debug

```
*** add put _all_ to understand where it went wrong ;
data vs;

    put 'A: ' _all_ '(Before Set statement)';

    set define_vs vital;

    put 'B: ' _all_ '(After Set statement)';

    if rfstdtc <= vsdtc <= rfendtc then vstrtem = 'Y';

    put 'C: ' _all_ '(After If statement)' ;
    put '---------------------';
run;
```

**Log Output**



```
A: subjid=  vsdtc=  vstrtem=  RFSTDTC=  RFENDTC=  _ERROR_=0 _N_=1 (Before Set statement)
B: subjid=111222333 vsdtc=2013-10-22T10:11 vstrtem=  RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
C: subjid=111222333 vsdtc=2013-10-22T10:11 vstrtem=Y RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
--------------------
A: subjid=111222333 vsdtc=2013-10-22T10:11 vstrtem=Y RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
B: subjid=111222333 vsdtc=2014-02-11 vstrtem=Y RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
C: subjid=111222333 vsdtc=2014-02-11 vstrtem=Y RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
--------------------
A: subjid=111222333 vsdtc=2014-02-11 vstrtem=Y RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
B: subjid=111222333 vsdtc=2013-08-27T10:44 vstrtem=Y RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
C: subjid=111222333 vsdtc=2013-08-27T10:44 vstrtem=Y RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
--------------------
A: subjid=111222333 vsdtc=2013-08-27T10:44 vstrtem=Y RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
B: subjid=111222333 vsdtc=2013-11-21T10:40 vstrtem=Y RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
C: subjid=111222333 vsdtc=2013-11-21T10:40 vstrtem=Y RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
--------------------
A: subjid=111222333 vsdtc=2013-11-21T10:40 vstrtem=Y RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
B: subjid=111222333 vsdtc=2014-01-15T11:00 vstrtem=Y RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
C: subjid=111222333 vsdtc=2014-01-15T11:00 vstrtem=Y RFSTDTC=2013-09-24T13:53 RFENDTC=2014-01-14
```

The log described above illustrates how SAS compiles and executes the data step by step.

- The first record
  - At step A: Variables are initialized to be missing at the beginning.
  - At step B: SAS reads in the record. SUBJID, VSDTC, RFSTDTC and RFENDTC are overwritten by new values when the first observation is read by the SET statement. VSTRTEM remains missing as this variable does not exist in the dataset VITAL.
  - At step C: SAS executes the IF statement. The variable VSTRTEM is replaced with value ‘Y’, as the criteria of the IF statement, variable VSDTC inside the boundary of variable RFSTDTC and RFENDTC is satisfied.

- The second record
  - At step A: The values are retained from the previous execution. The second observation here hence shows the value of first observation, VSTRTEM equals ‘Y’.
  - At step B: SAS reads in the record. Variables SUBJID, VSDTC, RFSTDTC and RFENDTC are overwritten by new values when the second observation is read by the SET statement. There is no VSTRTEM from dataset VITAL to be read. The variable VSTRTEM value remains as ‘Y’, which is an automatically retained value from previous record.
  - At step C: SAS executes the IF statement. The criteria of the IF statement is not met. The variable VSTRTEM remains with value ‘Y’.

The process goes on and the VSTRTEM stays with the value ‘Y’ no matter the criteria truly met or not.

### Suggested Solutions

To avoid this unexpected result, it is safer to break the DATA processes into two steps: 1) using SET statement to append the datasets to one dataset; 2) working on the variable derivation in another new DATA step.

**< Correct Coding >**

```
data vs;
    set define vital;
run;

data vs1;
    set vs;
    if rfstdtc <= vsdtc <= rfendtc then vstrtem = 'Y';
run;
```

Another solution, in this particular example, is to write a complete IF-THEN/ELSE statement. The ELSE statement has to cover all the possible scenarios so that variable VSTRTEM can be re-defined in this step. Variable VSTRTEM is set to be blank value when VSDTC is not in the boundary of variable RFSTDTC and RFENDTC.

**< Correct Coding >**

```
data vs;
    set define vital;

    if rfstdtc <= vsdtc <= rfendtc then vstrtem = 'Y';
    else vstrtem= ' ';
run;
```

## CONCLUSION

The concept of automatic retain is less recognized and understood by SAS users. There are variables not coded in a RETAIN statement but their values are still automatically retained. Without understanding the automatically retained feature, we might unintentionally create the programming bugs. In the examples demonstrated in this paper, although the logs are fine without warning or error messages, the output result is not correct. This paper illustrates how such cases could happen, and demonstrates how to avoid this kind of bugs.

Being cautious can avoid the unexpected results. A clean log cannot guarantee that our coding is correct. In addition, we should always exam the output to avoid mistake.

## REFERENCES

Gorrell, Paul. (1999), "The RETAIN Statement: One Window Into the SAS® Data Step," Proceedings of the NorthEast SAS Users Group (NESUG) Conference 1999.
Howard, Neil. (2005), "How SAS Thinks," Proceedings of the SouthEast SAS Users Group (SESUG) Conference 2005.
Dunn, Toby and Chang Y. Chung, (2005), "Retaining, Lagging, Leading, and Interleaving Data," Proceedings of the PharmaSUG Conference 2005.
Tian, Yunchao. (2007), "The Power and the Danger of Automatic Retain," Proceedings of the NorthEast SAS Users Group (NESUG) Conference 2007.

## ACKNOWLEDGMENTS

The authors would like to thank the management teams for their advice on this paper/presentation.

## CONTACT INFORMATION

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

Huei-Ling Chen
c/o Merck & Co., Inc.
126 Lincoln Avenue
P.O. Box 2000
Rahway, NJ 07065
Phone: 732-594-2287
e-mail: Huei-Ling_Chen@merck.com

Hong Zhang
Eli Lilly and Company
440 Route 22
Bridgewater, NJ 08807
Phone: 908-243-3138
e-mail: zhang_hong@lilly.com

Patricia Guldin
c/o Merck & Co., Inc.
MAILSTOP UG1D-10
351 North Sumneytown
North Wales, PA 19454

Phone: 267-305-8242
e-mail: patricia_guldin@merck.com

## TRADEMARK