PharmaSUG 2015 - Paper BB16

Unpacking an Excel Cell: Dealing with Multi-Line Excel Cells in SAS

Lucheng Shao, Ivantis Inc., Irvine, CA

ABSTRACT

In clinical trials it is very common that a variable for a subject could have multiple observations for any particular visit. If the data are entered into Microsoft Excel, this could end up with having multiple observations in one cell. When an Excel file with multiple lines in one cell is imported into SAS® there can be problems because of unprintable (control) characters in the variables. This paper will show you examples of importing multiple line Excel cells and how to deal with them without the need to resort to cryptic macro code. This paper is intended for SAS users who are familiar with SAS BASE and want to learn more about how to create a new SAS function using the FCMP procedure.

INTRODUCTION

In a clinical study it is very likely that one subject could be taking more than one medication at any particular visit. As shown in Figure 1, Subject 1 was taking three medications (MedA, MedB, MedC), all of which were listed in one Excel cell with each one medication taking a separate line. You may expect the output dataset look pretty much the same after this original dataset being imported into SAS. Let's see what we will get using the IMPORT procedure.

Subject	Meds	Dose
	MedA	1.1
	MedB	1.2
1	MedC	0.9
2	MedD	2.2
	MedE	4.3
3	MedF	1.6

Figure 1. MEDS.xlsx. An Example Excel Dataset with Multiple Observations in One Cell

```
PROC IMPORT OUT= MEDS_FROMEXCEL
            DATAFILE= "C:\USERS\LSHAO\DESKTOP\LUCHENG\MEDS.XLSX"
            DBMS=EXCEL REPLACE;
            RANGE="SHEET1$";
            GETNAMES=YES;
            MIXED=NO;
            SCANTEXT=YES;
            USEDATE=YES;
            SCANTIME=YES;
RUN;

PROC PRINT DATA= MEDS_FROMEXCEL;
RUN;
```

Obs	Subject	Meds	Dose
1	1	MedA MedB MedC	1.1 1.2 0.9
2	2	MedD	2.2
3	3	MedE MedF	4.3 1.6

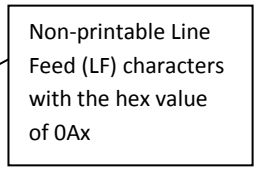
Output 1. MEDS_FROMEXCEL. Output of MEDS.XLSX from PROC IMPORT

In the output file MEDS_FROMEXCEL, Output 1, those multiple lines in one cell in the original MEDS.xlsx file were combined into a single line. It looks like blanks were also added as delimiter in the newly formed lines. However, those 'blanks' are not real blanks. They are non-printable Line Feed (LF) characters with the hex value of 0Ax, which are embedded when Enter + Alt was used to generate a new line of observation within the same cell in Excel. The following program shows characters in hexadecimal format and thus makes the Line Feed visible.

```
DATA FIND_DELIMITER(DROP=DOSE);
SET MEDS_FROMEXCEL;
IF _N_=1;
   DO POSITION = -1 TO 15;
      RESULT=CHAR(MEDS, POSITION);
      RESULT_HX=RESULT;
      FORMAT RESULT_HX $HEX10.;
      OUTPUT;
   END;
RUN;

PROC PRINT NOOBS DATA= FIND_DELIMITER;
RUN;
```

Subject	Meds	position	result	result_HX
1	MedAMedBMedC	-1		20
1	MedAMedBMedC	0		20
1	MedAMedBMedC	1	M	4D
1	MedAMedBMedC	2	e	65
1	MedAMedBMedC	3	d	64
1	MedAMedBMedC	4	A	41
1	MedAMedBMedC	5		0A
1	MedAMedBMedC	6	M	4D
1	MedAMedBMedC	7	e	65
1	MedAMedBMedC	8	d	64
1	MedAMedBMedC	9	B	42
1	MedAMedBMedC	10		0A
1	MedAMedBMedC	11	M	4D
1	MedAMedBMedC	12	e	65
1	MedAMedBMedC	13	d	64
1	MedAMedBMedC	14	C	43
1	MedAMedBMedC	15		20



Output 2. Output FIND_DELIMITER. It shows the Line Feed (LF) characters with hex value of 0Ax.

GET BUILDING BLOCKS - WRITE A SAS FUNCTION TO PARSE CELL WITH MULTIPLE LINES AND SAVE PARSED LINES AS ARRAY ELEMENTS

As we can see, PROC IMPORT combines multiple lines, including the non-printable Line Feed, into a single line in the SAS output. This 'everything in one line' format makes it difficult for data review and almost impossible for further data analysis. First and foremost, we need to parse the newly formed line into separate elements, so later we could manage them in the way we want them to be. Often times there will be multiple variables having this multi-line issue simultaneously because they are related. For example, in MEDS_FROMEXCEL dataset both **MEDS** and **DOSE** variables have this issue after being imported. Writing a SAS function could help improve efficiency for both the program and the programmer. PROC FCMP was used to create a SAS function to parse the cell (see code below).

Note that some authors have found both a Carriage Return (0Dx) and a Line Feed (0Ax) in their tables imported from Excel (see REFERENCE 1), however none of the workbooks tested for this paper had a Carriage Return character. If you do encounter other type of line break, you will need to parse on the line break accordingly in the subroutine. One nice thing to have a subroutine is that you would only need to change the parts that you would like to change (eg. to switch between different line breaks) once within the subroutine rather than going through the whole code and fixing them one by one.

```
PROC FCMP OUTLIB=WORK.FUNCS.TEST;
SUBROUTINE SPLITCELL(COLNAME $, NUMLINES, LINEARRAY[*] $);
   OUTARGS NUMLINES, LINEARRAY;
① ARRAYDIM = DIM(LINEARRAY);
   NUMLINES = COUNTW(COLNAME, '0A'X,'M');
   IF NUMLINES > ARRAYDIM
   THEN
      DO;
         NUMLINES = -1 * NUMLINES;
      RETURN;
      END;
②    DO I=1 TO NUMLINES;
         LINEARRAY{I}=SCAN(COLNAME, I, '0A'X, 'M');
      END;
   RETURN;
ENDSUB;
RUN;
```

The option **outlib=work.funcs.test** tells SAS where to store the function.

Subroutine defines a call routine (a type of SAS function named splitCell with three arguments –

1. **colName** – the column to be parsed
2. **lineArray** – the array of variables with the parsed values
3. **numLines** – the number of values found

Outargs tells SAS the value of numLines and LineArray can be altered and altered value will be returned.

In most clinical trials a subject will have a protocol-allowed maximum number of medications, and this number is represented by arrayDim in part ①. The value of numLines returns the actual number of medications that each subject has in the original Excel dataset. Part ① is used to check whether the actual number of medications exceeds protocol-allowed maximum number of medications. If it is, a negative value of numLines will be returned as a warning. We could make a subset of those subjects who has a negative value of numLines for further check.

In Part ②, multiples lines in one cell are parsed and each line is saved as an element of lineArray[*] .

Now we apply the subroutine that we just created to our MEDS_RAW dataset (see code below). Two datasets will be generated – a dataset called 'MEDS' that has parsed lines as array elements and a dataset called 'ERRORS' that collects all the observations with error (in this example would be either those subjects whose number of medications exceeds the maximum number allowed by protocol or those subjects who have medications with missing dose information).

```
OPTIONS CMPLIB=WORK.FUNCS;
%LET MAXLINES = 3; * THIS NUMBER EQUALS ALLOWED MAX NUMBER OF MEDS. SUPPOSE THIS
NUMBER IS 3;
%LET IMP='YES'; * IMP COULD BE 'YES' TO TRIGGER THE IMPUTATION PROCESS OR 'NO' TO
OUTPUT ALL OBSERVATIONS WITH MISSING DOSE INTO ERRORS DATASET;

DATA MEDS ERRORS;
   SET MEDS_FROMEXCEL;
   ARRAY MEDLIST(*)  $30. MED1 - MED&MAXLINES;
   ARRAY DOSELIST(*) $30. DOSE1 - DOSE&MAXLINES;
   ARRAY MEDLIST_TMP(&MAXLINES) $30. _TEMPORARY_;
   ARRAY DOSELIST_TMP(&MAXLINES) $30. _TEMPORARY_;
   ARRAY CNTS(*) NUMMEDS NUMDOSE;
   CALL MISSING(OF MEDLIST_TMP(*), OF DOSELIST_TMP(*), OF CNTS(*));
   CALL SPLITCELL(MEDS, NUMMEDS, MEDLIST_TMP);
   E_CODE=0;

* ERROR TYPE I: NUMBER OF MEDS EXCEEDS PROTOCOL-ALLOWED MAX NUMBER OF MEDS.
OUTPUT THESE OBSERVATIONS INTO DATASET ERRORS;

   IF NUMMEDS < 0
      THEN DO;
         E_CODE=1; *CALL THIS TYPE I ERROR;
         OUTPUT ERRORS;
      END;
   CALL SPLITCELL(DOSE, NUMDOSE, DOSELIST_TMP);

* ERROR TYPE II: MISSING DOSE FOR MEDS (ASSUMING TYPE I ERROR FREE). EITHER IMPUTE
MISSING DOSE WITH '0.0' OR OUTPUT THESE OBSERVATIONS INTO DATASET ERRORS;

   DOSE_M = CMISS(OF DOSELIST_TMP{*});
   MEDS_M = CMISS(OF MEDLIST_TMP{*});
   IF NUMMEDS>0 AND NUMDOSE>0 THEN DO; *ASSUME TYPE I ERROR FREE;
      * CASE (A): IMPUTATION;
      IF (DOSE_M ^= MEDS_M) AND &IMP='YES' THEN
         DO K= 1 TO &MAXLINES;
            IF MISSING(DOSELIST_TMP{K}) THEN DOSELIST_TMP{K}='0.0';
         END;
      * CASE (B): NO IMPUTATION;
      * INSTEAD, OUTPUT THESE OBSERVATIONS WITH MISSING DOSE INTO DATASET ERRORS;
         ELSE IF (DOSE_M ^= MEDS_M) AND &IMP='NO' THEN DO;
            E_CODE=2; *CALL THIS TYPE II ERROR;
            OUTPUT ERRORS;
         END;
   END;

   DO I=1 TO DIM(MEDLIST);
      MEDLIST{I}=MEDLIST_TMP{I};
   END;
   DO J=1 TO DIM(DOSELIST);
      DOSELIST{J}=DOSELIST_TMP{J};
   END;
   OUTPUT MEDS;
   DROP I J K NUMMEDS NUMDOSE DOSE_M MEDS_M;
RUN;
```

Three examples will be shown below.

	Name of Original file in Excel	Name of SAS dataset after using PROC IMPORT	Dataset MEDS. Final output after applying subroutine SPLITCELL	Dataset ERRORS. Final output after applying subroutine SPLITCELL	Notes
Example (a)	MEDS.XLSX (Figure 1)	MEDS_FROMEXCEL (Output 1)	(Output 3)	Empty output dataset	Original dataset is error free.
Example (b)	MEDS_E1.XLSX (Figure 2)	MEDS_R_E1 (Output 4)	(Output 5)	(Output 6)	Original dataset has number of medications exceeds the protocol-allowed maximum number of medications. Observations with errors are kept in the output MEDS dataset with the value of array elements being missing. At the same time, observations with errors are output to the ERRORS dataset.
Example (c1)	MEDS_E2.XLSX (Figure 3)	MEDS_R_E2 (Output 7)	(Output 9)	Empty output dataset	Original dataset has missing dose info. &IMP='YES'. In the output MEDS dataset missing dose will be imputed with '0.0'. No observation will be output to the ERRORS dataset.
Example (c2)	MEDS_E2.XLSX (Figure 3)	MEDS_R_E2 (Output 7)	(Output 10)	(Output 11)	Original dataset has missing dose info. &IMP='NO'. In the output MEDS dataset missing dose will be kept as missing. At the same time, observations with errors are output to the ERRORS dataset.

The original dataset MEDS.XLSX (Figure 1) is error free so the output dataset ERRORS has no observation. As you can see, multiple lines in MEDS.XLSX have been parsed with each line being stored as an array element in the output dataset MEDS (Output 3). These line elements (med1, med2, med3, dose1, dose2, dose3) are ready for further manipulation.

Obs	Subject	Meds	Dose	med1	med2	med3	dose1	dose2	dose3	e_code
1	1	MedA MedB MedC	1.1 1.2 0.9	MedA	MedB	MedC	1.1	1.2	0.9	0
2	2	MedD	2.2	MedD			2.2			0
3	3	MedE MedF	4.31.6	MedE	MedF		4.3	1.6		0

Output 3. Output MEDS from MEDS_FROMEXCEL

In the original dataset MEDS_E1.XLSX (Figure 2) the number of medications of subject 1 exceeds the protocol-allowed maximum number of medications.

```
NUMLINES = COUNTW(COLNAME, '0A'X,'M');
IF NUMLINES > ARRAYDIM
THEN
   DO;
      NUMLINES = -1 * NUMLINES;
   RETURN;
   END;
```

This part of code returns NUMLINES = -4. Since the array in the output dataset MEDS (Output 5) only allows three elements (med1-med3 and dose1-dose3), nothing was stored.

```
IF NUMMEDS < 0
   THEN DO;
      E_CODE=1; *CALL THIS TYPE I ERROR;
      OUTPUT ERRORS;
   END;
```

Instead, the line with error was output into the ERRORS dataset (Output 6) with e-code specified as 1.

Subject	Meds	Dose
	MedA	1.1
	MedB	1.2
	MedC	0.9
1	MedG	2.5
2	MedD	2.2
	MedE	4.3
3	MedF	1.6

Figure 2. MEDS_E1.XLSX

Obs	Subject	Meds	Dose
1	1	MedA MedB MedC MedG	1.1 1.2 0.9 2.5
2	2	MedD	2.2
3	3	MedE MedF	4.3 1.6

Output 4. MEDS_R_E1. Output of MEDS_E1.XLSX from PROC IMPORT

Obs	Subject	Meds	Dose	med1	med2	med3	dose1	dose2	dose3	e_code
1	1	MedA MedB MedC MedG	1.1 1.2 0.9 2.5							1
2	2	MedD	2.2	MedD			2.2			0
3	3	MedE MedF	4.3 1.6	MedE	MedF		4.3	1.6		0

Output 5. Output MEDS from MEDS_R_E1

Obs	Subject	Meds	Dose	med1	med2	med3	dose1	dose2	dose3	e_code
1	1	MedA MedB MedC MedG	1.1 1.2 0.9 2.5							1

Output 6. Output ERRORS from MEDS_R_E1

Subject	Meds	Dose
	MedA	1.1
	MedB	
1	MedC	0.9
2	MedD	2.2
	MedE	4.3
3	MedF	1.6

Figure 3. MEDS_E2.XLSX

Obs	Subject	Meds	Dose
1	1	MedA MedB MedC	1.1 0.9
2	2	MedD	2.2
3	3	MedE MedF	4.3 1.6

Output 7. MEDS_R_E2. Output of MEDS_E2.XLSX from PROC IMPORT

In the original dataset MEDS_E2.XLSX (Figure 3) the Dose information for MedB with subject 1 is missing.

```
DOSE_M = CMISS(OF DOSELIST_TMP{*});
MEDS_M = CMISS(OF MEDLIST_TMP{*});
```

DOSE_M calculates number of missing values for array DOSELIST_TMP{*}. MEDS_M calculates number of missing values for array MEDLIST_TMP{*}

If we print out MEDS_M and DOSE_M for the MEDS_R_E2 dataset we will get the following output.

Obs	Subject	Meds	Dose	med1	med2	med3	dose1	dose2	dose3	MEDS_M	DOSE_M
1	1	MedA MedB MedC	1.1 0.9	MedA	MedB	MedC	1.1		0.9	0	1
2	2	MedD	2.2	MedD			2.2			2	2
3	3	MedE MedF	4.3 1.6	MedE	MedF		4.3	1.6		1	1

Output 8. Output with MEDLIST_TMP{*}, DOSELIST_TMP{*}, MEDS_M and DOSE_M for MEDS_R_E2

```
IF NUMMEDS>0 AND NUMDOSE>0 THEN DO; *ASSUME TYPE I ERROR FREE;
   * CASE (A): IMPUTATION;
   IF (DOSE_M ^= MEDS_M) AND &IMP='YES' THEN
      DO K= 1 TO &MAXLINES;
         IF MISSING(DOSELIST_TMP{K}) THEN DOSELIST_TMP{K}='0.0';
      END;
   * CASE (B): NO IMPUTATION.
   * INSTEAD, OUTPUT THESE OBSERVATIONS WITH MISSING DOSE INTO DATASET ERRORS;
      ELSE IF (DOSE_M ^= MEDS_M) AND &IMP='NO' THEN DO;
         E_CODE=2; *CALL THIS TYPE II ERROR;
         OUTPUT ERRORS;
      END;
   END;
```

With &IMP='YES', in the output MEDS dataset (Output 9) missing dose with MedB of subject 1 was imputed with the value '0.0'. No observation was output to ERRORS dataset.

Obs	Subject	Meds	Dose	med1	med2	med3	dose1	dose2	dose3	e_code
1	1	MedA MedB MedC	1.1 0.9	MedA	MedB	MedC	1.1	0.0	0.9	0
2	2	MedD	2.2	MedD			2.2			0
3	3	MedE MedF	4.3 1.6	MedE	MedF		4.3	1.6		0

Output 9. Output MEDS from MEDS_R_E2 with &IMP='YES'

With &IMP='NO', in the output MEDS dataset (Output 10) missing dose was kept as missing. At the same time, observations with errors are output to the ERRORS dataset (Output 11).

Obs	Subject	Meds	Dose	med1	med2	med3	dose1	dose2	dose3	e_code
1	1	MedA MedB MedC	1.1 0.9	MedA	MedB	MedC	1.1		0.9	2
2	2	MedD	2.2	MedD			2.2			0
3	3	MedE MedF	4.3 1.6	MedE	MedF		4.3	1.6		0

Output 10. Output MEDS from MEDS_R_E2 with &IMP='NO'

Obs	Subject	Meds	Dose	med1	med2	med3	dose1	dose2	dose3	e_code
1	1	MedA MedB MedC	1.1 0.9							2

Output 11. Output ERRORS from MEDS_R_E2 with &IMP='NO'

If we have other types of errors, the corresponding code part could be adjusted accordingly in a similar way as shown in the above examples.

BUILDING BLOCKS READY – YOU CAN START BUILDING IN WHATEVER WAY YOU WANT

Once we had the ERRORS dataset in hand, each observation in the dataset could be closely examined and the corresponding part in the MEDS dataset could be corrected accordingly. Now this error-free MEDS dataset with parsed line saved as array elements will serve as a new starting point for you to formulate your final SAS output. Below, two examples are given as possible ways you may want to have the final output be like. There could be other preference for the way to formulate the final output depending on your purpose.

Suppose now our MEDS dataset is error-free (using the Output3 MEDS dataset as an example), for the purpose of further data analysis we would most likely want to have medication and dose taking different columns (Output 12).

For the purpose of data review, we may like to have the medication and dose information be combined in one column (Output 13).

```
DATA MEDS_NEW(KEEP=SUBJECT MED_NEW DOSE_NEW);
SET MEDS; * Using MEDS dataset in OUTPUT 3 as an example;
MED_NEW=MED1; DOSE_NEW=DOSE1; OUTPUT;
MED_NEW=MED2; DOSE_NEW=DOSE2; OUTPUT;
MED_NEW=MED3; DOSE_NEW=DOSE3; OUTPUT;
RUN;

DATA MEDS_NEW2;
SET MEDS_NEW;
IF MISSING(MED_NEW) OR MISSING(DOSE_NEW) THEN DELETE;
RUN;

PROC PRINT DATA=MEDS_NEW2;
RUN;
```

Obs	Subject	med_new	dose_new
1	1	MedA	1.1
2	1	MedB	1.2
3	1	MedC	0.9
4	2	MedD	2.2
5	3	MedE	4.3
6	3	MedF	1.6

Output 12. Final Output MEDS_NEW2 from MEDS (Output 3) with one observation taking a single line and with meds and dose kept in different columns

```
DATA MEDS_RW(KEEP=SUBJECT M_D_LIST);
RETAIN SUBJECT M_D_LIST;
LENGTH M_D_LIST $100;
SET MEDS_NEW2;
M_D_LIST=MED_NEW||' '||'DOSE:'||DOSE_NEW;
RUN;

PROC PRINT DATA=MEDS_RW;
RUN;
```

Obs	Subject	M_D_list
1	1	MedA Dose:1.1
2	1	MedB Dose:1.2
3	1	MedC Dose:0.9
4	2	MedD Dose:2.2
5	3	MedE Dose:4.3
6	3	MedF Dose:1.6

Output 13. Final Output MEDS_RW from MEDS (Output 3) with one observation taking a single line and with meds and dose information combined in one column

CONCLUSIONS

This paper shows that the critical step in solving the multiple-line issue is to parse the cell and save each single line as an array element. This gives you the greatest degree of freedom in manipulating those array elements in later steps to formulate the final SAS output that you want. This paper also shows how to create a SAS function by using PROC FCMP. The benefit of having a SAS function is that the function is constructed with familiar DATA steps and is reusable. Although SAS already has lots of built in functions, you may still need to create one to fit your own need as in the example shown in this paper.

REFERENCES

1. Reading an Excel Spreadsheet with Cells Containing Line Endings. Larry Hoyle, Institute for Policy & Social Research, University of KansasHow to Represent Missing Data: Special Missing Values vs. 999999999. Quentin McMullen, Westat, Rockville MD

2. A Cup of Coffee and Proc FCMP: I Cannot Function Without Them. Peter Eberhardt, Fernwood Consulting Group Inc, Toronto ON

ACKNOWLEDGMENTS

I would like to thank Peter Eberhardt for all his support and advice in editing this paper.

CONTACT INFORMATION

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

Lucheng Shao
Ivantis, Inc.
38 Discovery, Ste 150
Irvine, CA, 92618
lshao@ivantisinc.com