Paper 315-2013

Reading an Excel Spreadsheet with Cells Containing Line Endings

Larry Hoyle, Institute for Policy & Social Research, University of Kansas

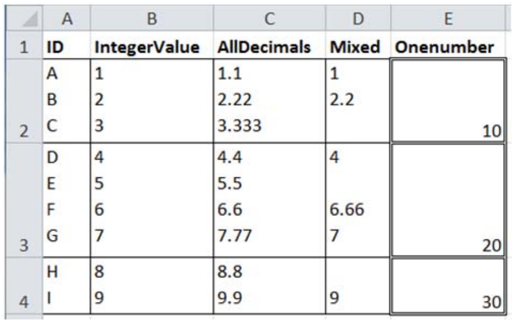
ABSTRACT

The creative ways people enter data into Excel spreadsheets can cause problems when trying to import data into SAS® data sets. This paper addresses the problem encountered when spreadsheet cells contain multiple lines (that is, the cells have embedded line endings). Several approaches to reading such data are described and compared.

INTRODUCTION

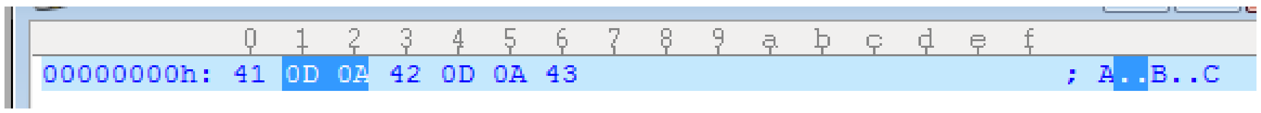
An Excel user may create multiline entries in a cell by pressing ALT Enter to embed a line ending in a string. The example spreadsheet, Sample3.xlsx, shown here in Figure 1, has 4 rows of cells and 5 columns, but actually needs to be read into a 9 row SAS dataset.

Copying and pasting the cell A2 into a text editor that displays the string in hexadecimal reveals that there are Carriage Return (0D) and Line Feed (0A) characters embedded within the string. The line ending between the A and B are shown highlighted in Figure 2 below.



	A	B	C	D	E
1	ID	IntegerValue	AllDecimals	Mixed	Onenumber
2	A B C	1 2 3	1.1 2.22 3.333	1 2.2	10
3	D E F G	4 5 6 7	4.4 5.5 6.6 7.77	4 6.66 7	20
4	H I	8 9	8.8 9.9	 9	30

Figure 1, Sample3.xlsx, An Example Spreadsheet With Embedded Line Endings in Some Cells

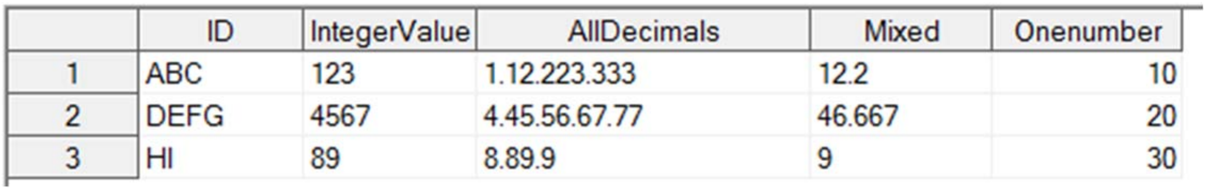


```
             0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 41 0D 0A 42 0D 0A 43                              ; A..B..C
```

Figure 2 – Cell A2, Hexadecimal Representation at Left Showing OD OA Delimiters

SAS IMPORT WIZARD

How do the import tools in SAS Display Manager and Enterprise Guide handle spreadsheets like this? The dataset imported from the file Sample3.xlsx in Display Manager (File…Import Data… Microsoft Excel) is shown in Figure 3 below.



	ID	IntegerValue	AllDecimals	Mixed	Onenumber
1	ABC	123	1.12.223.333	12.2	10
2	DEFG	4567	4.45.56.67.77	46.667	20
3	HI	89	8.89.9	9	30

Figure 3 - The Imported Dataset as Viewed in Display Manager with the Viewtable Facility

The SAS code generated by the Display manager import wizard follows:

```
   PROC IMPORT OUT= WORK.rawDataDM
       DATAFILE= "C:\DDRIVE\projects\sugs\SGF2013\paper\ReadingExcel2013\Sample3.xlsx"
       DBMS=EXCELCS REPLACE;  *  for 64 bit SAS with PC files server  ;
                             /* DBMS=EXCEL REPLACE;        *  for 32 bit SAS   ; */
       RANGE="rawdata$";
       SCANTEXT=YES;
       USEDATE=YES;
       SCANTIME=YES;
   RUN;
```

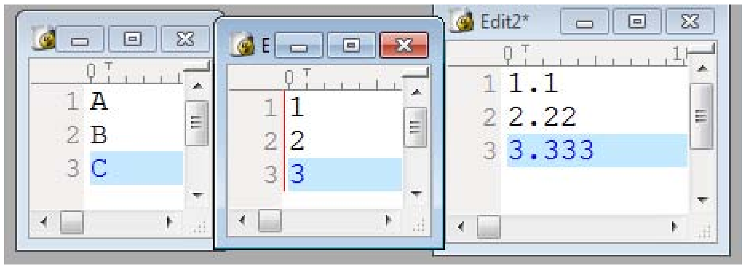


Figure 4 - - First Row Values Copied from the SAS Dataset for ID, IntegerValue, and AllDecimals.

In the Viewtable presentation (Figure 3), the line endings appear to have been discarded. Copying and pasting cells into a text editor reveals that the line endings are still in the strings (Figure 4). Displaying observation 1 for ID in hexadecimal also shows that it retains the same value as displayed in Figure 2. The Viewtable facility just ignores the line endings in the value.

PARSING THE CELLS

The dataset shown in Figure 3 needs to be parsed before it will be of much use. Complete code for that process is in Appendix A.

The first step in the process is to use PROC SQL against the dictionary tables to count the number of character variables in the table. (see ❶) Only character variables will be processed since line endings will not appear in other data types. This value is put into the macro variable NVARS. A second SQL query puts the list of character variable names into the macro variable VARLIST.

A DATA step creates the table work.ParsedData, reading from the original dataset, work.rawDataDM. Four arrays are defined. (see ❷) The array “allCharVars” points to the list of character variables contained in the macro variable VARLIST. The original values get copied into the temporary array “original”. The separated lines of data will be parsed back into the named variables, avoiding renaming. The third array contains a set of boolean values indicating whether each parsed column contains only numeric values. The fourth array contains a list of the names of the columns.

The spreadsheet row number is captured from the automatic variable “_n_”. THis may represent a grouping variable in the final result, where each group represents a set of observations in the SAS dataset that came from a single row of the spreadsheet.

As the variables are copied (❸), the COUNTW function finds the number of embedded lines. The maximum number of lines for the row is captured in the variable “maxlines”.

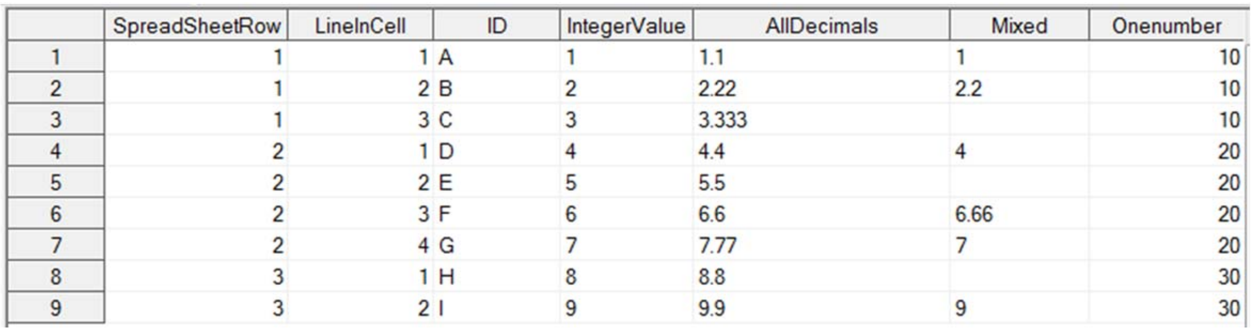
Once the original variables have been copied, the SCAN function breaks apart the original cell data. (❹) An outer loop iterates from line 1 through the maximum number of lines found for the spreadsheet row. Within that loop, each original value is scanned and the extracted line placed in that variable. If any column contains a non blank value then a logical variable, “goodrow”, is set to to true. Only rows with at least one non-blank value are output.

A subtlety in the use of options for both the COUNTW and SCAN is required. In Windows, the delimiter ending each line in the string is a pair of characters ‘0D0A’x. Successive occurrences of this delimiter indicate empty lines. The “M” option in the last argument of the function indicates that successive occurrences of any delimiter ***character*** indicate a missing value. The safe thing to do is to use one of the characters as the delimiter and then remove the other character once a line is split off. Since the Line Feed character (‘0A’x) appears in the line separater both in Windows and in Unix variants, it makes sense to use it for the delimiter and then delete any Carriage Return left over(‘0D’x).

```
   allCharVars{ixv}=compress(allCharVars{ixv},'0D'x);        * strip out Return;
```

Values with embedded line breaks are character strings, even if all of the characters in the string are numerics. The Do loop at ❹ also uses the INPUT function to check each parsed piece to see it it can be read as a number. The section of code at ❺ writes out a separate dataset (varname) with the names of the variables which are all numeric. These could then be converted to numeric variables (see, for example, SAS Usage Note 22147 http://support.sas.com/kb/22/147.html).

Figure 5 shows the resulting data. Note the missing values for the column “Mixed”. Trailing, successive, and leading, delimiters were detected as indicating missing data in rows 3, 5, and 8.



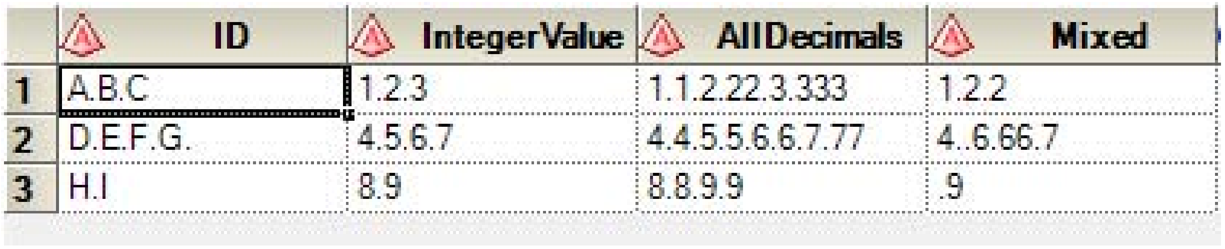
	SpreadSheetRow	LineInCell	ID	IntegerValue	AllDecimals	Mixed	Onenumber
1	1	1	A	1	1.1	1	10
2	1	2	B	2	2.22	2.2	10
3	1	3	C	3	3.333		10
4	2	1	D	4	4.4	4	20
5	2	2	E	5	5.5		20
6	2	3	F	6	6.6	6.66	20
7	2	4	G	7	7.77	7	20
8	3	1	H	8	8.8		30
9	3	2	I	9	9.9	9	30

Figure 5– The Parsed Dataset

OTHER SAS TOOLS

ENTERPRISE GUIDE

Enterprise Guide also has an import wizard for Excel Spreadsheets. Using that tool (File…import) on one of the sample spreadsheets yields the table seen in Figure 6. Here, the line endings have been converted to the character “.” (a period, ASCII 2E). This poses a real problem in terms of distinguishing line endings from decimal points. A workaround is possible if all numbers contain a decimal point.



	ID	IntegerValue	AllDecimals	Mixed
1	A.B.C	1.2.3	1.1.2.22.3.333	1.2.2
2	D.E.F.G.	4.5.6.7	4.4.5.5.6.6.7.77	4..6.66.7
3	H.I	8.9	8.8.9.9	.9

Figure 6 - The Spreadsheet as Imported by Enterprise Guide

EXPORTED DATA FROM EXCEL

Table 1 shows the spreadsheet exported from Excel as a comma delimited file. This file can be read, but not easily. Line endings inside quoted strings indicate multiline cells. There are also line endings between rows of the spreadsheet. Commas inside quotes are data. Commas outside quotes separate columns.

```
ID,IntegerValue,AllDecimals,Mixed
"A
B
C","1
2
3","1.1
2.22
3.333","1
2.2
3"
"D
E
F
G
","4
5
6
7","4.4
5.5
6.6
7.77","4
5
6.66
7"
"H
I","8
9","8.8
9.9","8
9"
```

Table 1 – The Spreadsheet Exported From Excel as Comma Delimited

XML ENGINE

Excel files are zipped XML files. It would also be possible to create a SAS XMLMap for xlsx files and import the spreadsheet using the XML libname engine. That option will not be explored here.

CONCLUSION

Proc IMPORT as shown in Appendix A is an effective approach to reading multiline Excel cells. With the ability to capture spreadsheet rows numbers and line number within cell, data can be joined across columns as needed for spreadsheets having structures other than the samples shown here.

CONTACT INFORMATION

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

Larry Hoyle

Institute for Policy & Social Research, University of Kansas

1541 Lilac Lane, Suite 607 Blake Hall

Lawrence, KS, 66045-3129

785 – 864 - 9110

LarryHoyle@ku.edu

http://www.ipsr.ku.edu/

APPENDIX A – SAS CODE FOR READING AND PARSING THE SAMPLE SPREADSHEET

```
/* Read a spreadsheet having cells with multiple lines */
PROC IMPORT OUT= WORK.rawDataDM
            DATAFILE=
"C:\DDRIVE\projects\sugs\SGF2013\paper\ReadingExcel2013\Sample3.xlsx"
            DBMS=EXCELCS REPLACE;  *  for 64 bit SAS with PC files server  ;
                       /* DBMS=EXCEL REPLACE;        *  for 32 bit SAS   ; */
     RANGE="rawdata$";
     SCANTEXT=YES;
     USEDATE=YES;
     SCANTIME=YES;
RUN;

/* Parse the cells containing multiple lines.*/
/* Output one dataset row for each line */
/* retain the last line for cells with fewer than the maximum number of lines */
/* first get the number of variables in the table  */

%LET MAXvlen=1024;    * the maximum length of any variable;

proc sql noprint;                                                      ❶
select count(name) as nvars into :NVARS
from dictionary.columns
where libname="WORK" and memname="RAWDATADM" and type="char";

select name into :VARLIST separated by ' '
from dictionary.columns
where libname="WORK" and memname="RAWDATADM" and type="char";

quit;
%put &NVARS;
%put &VARLIST;

data ParsedData(drop=_vname _n _vnumber)  varname(keep= _vname);
length SpreadSheetRow 8 LineInCell 8 _vname $ 200;
set work.rawdatadm end=last;                              ❷
drop ixv  maxlines nlines  goodrow;

array original{&NVARS} $ &MAXvlen _temporary_;
array allCharVars{&NVARS} &VARLIST;     *  only character variables will be parsed;
array numericCol{&NVARS} _TEMPORARY_;
array vnames{&NVARS} $ 200 _TEMPORARY_;

spreadSheetRow=_n_;
                                                               ❸
do ixv=1 to &NVARS;
                  /*  copy the original string  */
                  /*  the parsed pieces will be put back into the named variable */

  original{ixv} = allCharVars{ixv};
                     /* count the maximum number of lines, including null lines */
  nlines=countw(allCharVars{ixv},'0A'x,'M');
  if nlines>maxlines then maxlines=nlines;
end;
                  /*  scan for maxlines pieces of each original string   */
do LineInCell = 1 to maxlines;
  goodrow=0;                                                                    ❹
  do ixv=1 to &NVARS;
    numericCol{ixv}=1;   *  assume the column is numeric;
    vnames{ixv} = vname(allCharVars{ixv});
    allCharVars{ixv}=scan(original{ixv},LineInCell,'0A'x,'M');  * get a line;
    allCharVars{ixv}=compress(allCharVars{ixv},'0D'x);        * strip out Return;
    if allCharVars{ixv} ne ' ' then goodrow=1;            *something not null;

    _ERROR_=0;
    _n=input(allCharVars{ixv}, ? 32.);            * can this be read as a number? ;
    if _ERROR_ then numericCol{ixv}=0;      not then this is not a numeric column ;
  end;

if goodrow then output ParsedData;      * this eliminates all blank rows from the
output;
end;

if last then do;      * save a list of all the variables with only numeric values;
  do _vnumber=1 to &NVARS;

    if numericCol{_vnumber} then do;                           ❺
      _vname=vnames{_vnumber};
      output varname;
    end;  * numericCol{_vnumber};
  end;  * _vnumber=1 to &NVARS;
end;   *last;
run;
```