

Useful Tips for Handling and Creating Special Characters in SAS®

Bob Hull, SynteractHCR, Inc., Carlsbad, CA
Robert Howard, Veridical Solutions, Del Mar, CA

ABSTRACT

This paper will discuss various ways of creating and dealing with special characters in SAS. Many people experience difficulty when reading in excel files and discover that strange "boxes" appear in the data. What these are and how they can be dealt with will be discussed. Can special characters be saved in the SAS program? How can these characters be typed if they aren't on the keyboard? We will also provide examples on how to include special characters like Greek letters (μ), less than or equal to (\leq), and registered trademark ($\text{\textcircled{R}}$) into your SAS programs and RTF output. This paper will help you better understand some ways that special characters can be used within SAS.

INTRODUCTION

It can be said that the relationship between "special characters" and SAS is a tenuous one. Sometimes problems or errors are encountered when trying to read in external data or even when simply accessing SAS datasets which have variables containing special characters. As a result, either the file cannot be accessed or unrecognizable characters appear.

After having to solve some real-life problems, we decided it would be best to summarize some of our solutions for handling these issues. In this paper, we'll first look at some solutions for reading in data containing special characters and look at some examples. Next, we'll go over some tricks for writing out special characters to either your SAS output or RTF files.

READING IN SPECIAL CHARACTERS

By looking at a few examples, we will provide practical solutions for handling issues caused by reading in data containing special characters.

UNICODE DATA ERROR WHEN READING IN SAS DATASETS

When reading in SAS datasets it's possible that special characters will prevent you from being able to use the data. Have you seen this transcoding error in your log?

ERROR: Some character data was lost during transcoding in the dataset DB.LABS. Either the data contains characters that are not representable in the new encoding or truncation occurred during transcoding.

The data has Unicode characters in it and your SAS session is not set up for Unicode even though it appears the same as other SAS datasets. Unicode allows for different languages that became available beginning in Version 9.1.3. See the Recommended Reading section for more info on Unicode from SAS.

The ideal solution is to read in the data using SAS with Unicode support. In doing so, the special characters will show up correctly. However, if that is not available then you will be able to successfully read in the data using the following code:

```
data temp;
  set db.labs (encoding='asciiany');
run;
```

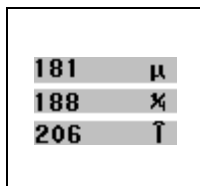
However, while we're now able to access the data, closer inspection reveals that the special characters appear in one of the variables (CUTOFF) which was the root of the problem. The Greek letter " μ " has been converted to " ¯¼". See Output 1 below for an example of how these values may appear.

Greater than 0.1	¯¼g/kg/min
Greater than 0.1	¯¼g/kg/min
Greater than 0.1	¯¼g/kg/min
Greater than 0.1	¯¼g/kg/min

Output 1. In this example the data is read in, but the Greek letter " μ " is converted to something indiscernible.

We can access a list of all available values in the current SAS session and their corresponding SAS byte value by executing the following code and looking at the log. Output 2 is a condensed screenshot of the log which has isolated three special characters of interest.

```
data _null_;
  do k=1 to 255;
    x=byte(k);
    put k +10 x;
  end;
run;
```

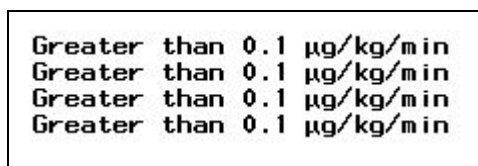


Output 2.
A screenshot of the log which isolates the special characters Î and ¼.

From the log, we can see that the byte codes 206 and 188 should be converted to a "μ", which we can see is identified as byte 181. In order to correctly display the value, we can use the byte function and tranwrd function to replace the special characters with the following code:

```
data temp;
  set db.labs (encoding='asciiany');
  cutoff=tranwrd(cutoff,byte(206),' ');
  cutoff=tranwrd(cutoff,byte(188),byte(181));
run;
```

The updated variable will now appear with the correct value. See Output 3 below.



Output 3: After transforming the special characters, the data and values can be used.

If the character attempting to be read in is not available in the list of 255, then the final solution may not work out as nicely as this situation did. However, the "asciiany" method will still enable you to at least read the data in and work with it, replacing the undesired value with something resembling what you need.

READING IN DATA FROM EXCEL

When reading special characters in from excel using SAS you may get unexpected results. Consider the following data in Excel, which is then read into SAS using proc import in Table 1 below.

<table border="1"> <thead> <tr> <th></th> <th>A</th> <th>B</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Var1</td> <td>Var2</td> </tr> <tr> <td>2</td> <td>abc</td> <td>def</td> </tr> <tr> <td>3</td> <td>ab©</td> <td>xΔz</td> </tr> <tr> <td>4</td> <td>a bc</td> <td>xz z</td> </tr> <tr> <td></td> <td>a</td> <td></td> </tr> <tr> <td></td> <td>b</td> <td></td> </tr> <tr> <td>5</td> <td>c</td> <td>xμz</td> </tr> </tbody> </table> <p style="text-align: center;">Original Excel file</p>		A	B	1	Var1	Var2	2	abc	def	3	ab©	xΔz	4	a bc	xz z		a			b		5	c	xμz	<table border="1"> <thead> <tr> <th>Obs</th> <th>VAR1</th> <th>VAR2</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>abc</td> <td>def</td> </tr> <tr> <td>2</td> <td>ab©</td> <td>x?z</td> </tr> <tr> <td>3</td> <td>a bc</td> <td>x=z</td> </tr> <tr> <td>4</td> <td>a□b□c</td> <td>xμz</td> </tr> </tbody> </table> <p style="text-align: center;">SAS output after proc import</p>	Obs	VAR1	VAR2	1	abc	def	2	ab©	x?z	3	a bc	x=z	4	a□b□c	xμz
	A	B																																						
1	Var1	Var2																																						
2	abc	def																																						
3	ab©	xΔz																																						
4	a bc	xz z																																						
	a																																							
	b																																							
5	c	xμz																																						
Obs	VAR1	VAR2																																						
1	abc	def																																						
2	ab©	x?z																																						
3	a bc	x=z																																						
4	a□b□c	xμz																																						

Table 1.
Side-by-side comparison of original Excel file with character returns in cell A5 and resulting SAS dataset created using proc import.

Some characters were not read in successfully. Notice that the delta and \geq sign were changed. When reading in a large dataset this may go unnoticed. However, when you are aware of this, these cases can be changed in the Excel file before importing as one solution. If it is not imported correctly, you will only be able to change it to something close like shown above in the Unicode section. My session does not have the greater than equal sign as one of the 255 possible characters, so it will not read in from an external file correctly.

The issue that arises when the returns are in excel presents a slightly different problem. The “box” that appears in SAS after importing is probably not desired. A return is not the only character that causes these boxes to appear. Unfortunately, there is no way to distinguish them. All boxes can be replaced with a space using the following code:

```
do aa=1 TO 29, 31, 127, 129, 141 to 144, 157, 158;
  var1=tranwrd(var1,byte(aa), ' ');
end;
```

So, where did this list of numbers come from in the do loop? Using the same code (displayed below again) from the first example, you can obtain all the corresponding byte values for the special box characters.

```
data _null_;
  do k=1 to 255;
    x=byte(k);
    put k +10 x;
  end;
run;
```

WRITING OUT SPECIAL CHARACTERS

WITHIN SAS OUTPUT

In some cases you may want to display special characters in your SAS output. By executing the code mentioned earlier, and repeated here, a list of values that can be inserted with the byte function is displayed in the log:

```
data _null_;
  do i=1 to 255;
    byte=byte(i);
    put i +10 byte;
  end;
run;
```

By reviewing the log, we see that byte 181 refers to mu (μ) and byte 174 refers to the registered trademark symbol (®). To use these special characters in your SAS program, you can use the BYTE function to translate these numeric codes into meaningful values. Note that the BYTE function returns a character value.

In a practical example, we can create the variable UNIT which displays “ $\mu\text{mol/L}$ ” using the special character. The following code produces the dataset in Output 4.

```
data unit;
  unit=byte(181) || "mol/L";
run;
```

unit
$\mu\text{mol/L}$

Output 4. Dataset with variable UNIT which contains the string “ μ ”.

You can now display “ $\mu\text{mol/L}$ ” in both SAS output and RTF documents. A similar approach would be used to create and display variables with the registered trademark (®) and degrees Celsius ($^{\circ}$).

WITHIN RTF OUTPUT

Within the RTF destination the use of the ODS escapechar (`ods escapechar='~';`) gives the user a wide range of special formatting tools to modify the output. While making a result bold is not a special character, it may be useful to some readers to see examples that alter the font in addition to inserting special characters. The following code provides several practical examples in one exercise. Below you will see how easy it is to underline, italicize, or bold your text, as well as insert a character return and use insert the special characters for “greater than or equal to” and “less than or equal to” signs.

```
ods escapechar='~';
title "~S={font_weight=bold}RTF Syntax ~S={}";

data example;
  set sashelp.class;
  if _n_ <= 4;
  *example to show plain, italic and bold text;
  if _n_ = 1 then
    a="plain ~S={font_style=italic} italic ~S={font_weight=bold}Bold";
  *create a character return display a greater than or equal to sign;
  if _n_ = 2 then a="Force ~n Break ~{unicode 2264}";
  *underline text;
  if _n_ = 3 then a="Plain {\ul UnderlineMe} Plain";
  *change the font color to red;
  if _n_ = 4 then a="~S={foreground=red}Red!~S={}";
run;

ods rtf file="s:\temp\rtf.rtf" style=ctdv9;

proc report data=example nowindows style(header column)=[protectspecialchars=off];
  columns a (" \brdrb\brdrs\brdrw1 spanning " name weight) ;
  define a      /display " " style(column)=[cellwidth=1.5 in];
  define name   /display "Name";
  define weight /display "~{unicode 2265} Weight" style(column)=[font_weight=bold];
run;

ods rtf close;
```

Executing the code above produces the RTF file which is found in Output 5 below.

RTF Syntax		
	spanning	
	Name	≥ Weight
plain <i>italic</i> Bold	Alfred	112.5
Force	Alice	84
Break ≤		
Plain <u>UnderlineMe</u> Plain	Barbara	98
Red!	Carol	102.5

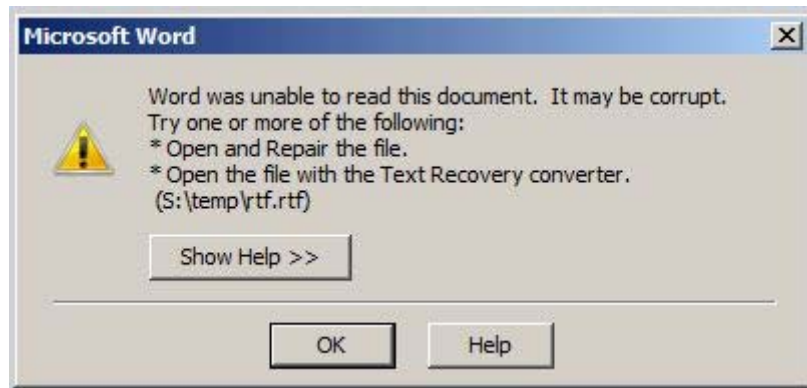
Output 5. The RTF table which displays the results after executing the code above.

The special characters, ≤ and ≥ are added in both the data cell and column header, respectively using the unicode statement. Unicode 2264 is used for ≤, and Unicode 2265 is used for ≥. See the Recommended Reading section below for a complete list of unicode characters and their codes.

A couple of other “special characters” are also inserted in this code. Note in the columns statement the text string “\brdrb\brdrs\brdrw1” is used to get a spanning underline of the joined columns. Also note the “~n” in the “_n_=2” line forces the next text in the cell to go down to the next line (character return). As you can see from these two simple examples, there are many ways to insert RTF language into SAS output.

BROKEN RTF FILES

When inserting these special characters or occasionally when working with data it is possible that your RTF file becomes corrupted. What has happened is that the syntax of the RTF file is no longer well-formed and the word processor cannot translate the file. This is almost always because the data has unbalanced braces, i.e., there is a “{” without a “}” following it. If the length of the variable is not long enough to keep the closing “}” then it may be truncated and the file won’t open. When trying to open it you may receive this message:



If you are inserting the erroneous code, then it is easy to correct it by fixing the length of the variable or fixing the typo that caused the closing “}” to be missing. However, if your data has this occurrence, then a little more effort is involved to fix this. You could exclude half your data and try running your file. If it works then the problem data is in the other half. Repeat this until the bad record is found and change the data. The suggested fix is to change “{” to “[“ as this usually solves the problem quickly and does not change the integrity of the output. This can be done using the tranwrd function on all variables going to the output if you aren’t inserting special codes/characters. Alternatively, you could print out all observations that contain a “{” in all vars going to the output and find the one that doesn’t have a “}” The best solution for you will depend on your data.

CONCLUSION

While dealing with special characters, both when reading in data or creating data, can be tricky, there are a lot of different ways of getting the results you need. This paper has demonstrated several different techniques to handle special characters. The examples show the tools the authors use to accomplish these tasks and will allow you to find your specific solution. The presentation of common error messages may prove useful when you least expect it. The authors used SAS 9.2 on Windows and Citrix during testing. Results may vary depending on your specific setup.

RECOMMENDED READING

- For more information on Unicode in SAS, visit: <http://support.sas.com/resources/papers/unicode913.pdf>
- For a complete list of Unicode Codes and their corresponding special characters, visit: http://en.wikipedia.org/wiki/List_of_Unicode_characters

CONTACT INFORMATION

Thank you for your time and interest. If you have any comments or questions please feel free to contact us at:

Name: Bob Hull
Enterprise: SynteractHCR, Inc.
Address: 5759 Fleet Street, Suite 100
City, State ZIP: Carlsbad, CA 92008
Work Phone: 760.268.8003
E-mail: rhull@synteract.com
Web: www.synteracthcr.com

Name: Rob Howard
Enterprise: Veridical Solutions
Address: P.O. Box 656
City, State ZIP: Del Mar, CA 92014
Work Phone: 858.205.8284
E-mail: rob.howard@veridicalsolutions.com
Web: www.veridicalsolutions.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.