PharmaSUG China 2016

The Impact of Change from wlatin1 to UTF-8 Encoding in SAS Environment

Hui Song, PRA Health Sciences, Blue Bell, PA, USA
Anja Koster, PRA Health Sciences, Zuidlaren, The Netherlands

ABSTRACT

As clinical trials become globalized, there has been a steadily strong growing need to support multiple languages in the collected clinical data. The default encoding for a dataset in SAS is “wlatin1”. wlatin1 is used in the “western world” and can only handle ASCII/ANSI characters correctly. UTF-8 encoding can fulfill such a need.

UTF-8 is a universal encoding that can handle characters from all possible languages, including English. It is backward compatible with ASCII characters. However, UTF-8 is a multi-byte character set while wlatin1 is a single-byte character set. This major difference of data representation imposes several challenges for SAS programmers when (1) import and export files to and from wlatin1 encoding, (2) read in wlatin1-encoded datasets in UTF-8 SAS environment, and (3) create wlatin1-encoded datasets to meet clients’ needs.

In this paper, we will present concrete examples to help the readers understand the difference between UTF-8 and wlatin1 encoding and provide practical solutions to address the challenges above.

INTRODUCTION

The default encoding for a dataset in SAS is “wlatin1” (or “wlatin1 Western (windows)”). wlatin1 is used in the “western world” and does not suppose Asian characters. With clinical trials become globalized, there has been a steadily strong growing needs to support multiple languages in the collected clinical data. UTF-8 is a universal encoding that can handle characters from all possible languages.

In UTF-8, ASCII was incorporated into the Unicode character set as the first 128 symbols, so the 7-bit ASCII characters have the same numeric codes in both encoding sets (ASCII and UTF-8). This allows UTF-8 to be backward compatible with the 7-bit ASCII. As such, a UTF-8 file containing only ASCII characters is identical to an ASCII file containing the same sequence of characters. It is similar for wlatin1, in which the first 128 symbols have the same numeric codes as in ASCII.

Thus, as long as you only use the first 128 symbols of ASCII (code 000 till 127, your normal keyboard characters like 0, 1, 2, … 9, a, b, … z, A, B, … Z, {, [, },] ,|, \, etc.), there will be no problem at all. However, notice that for example, the character ü is ASCII code 129 and µ is ASCII code 230. When transcoding is done from wlatin1 into UTF-8, then some single byte-characters from wlatin1 (such as ü and µ) might become 2 or 3 byte UTF-8 characters. What happens then?

This paper describes the consequences of changing to a new default encoding in SAS for SAS programmers.

WHAT ARE THE DIFFERENCES BETWEEN WLATIN1 AND UTF-8

As said, UTF-8 can handle all kinds of characters. It is a multi-byte character set, while wlatin1 is a single-byte character set. It is important to realize the differences of processing data in a single-byte versus a multi-byte environment. One UTF-8 character can be 1 byte, 2 bytes, 3 bytes, or even 4 bytes.

To process data in UTF-8 in SAS, you have to use the ***DBCS*** string functions (also known as ***K*** functions). To use K functions properly, you need to understand the difference between byte-based offset and character-based offset (or length-based). Most of the K functions require character-based offset.

In the following, we will use three examples to illustrate the impact of UTF-8 encoding.

EXAMPLE 1 NOT USE LENGTH STATEMENT

To get a better understanding of what this means, see the following simple SAS program:

```
data temp;
   unit = 'mmol/L'; output;
   unit = 'µmol/L'; output;
run;
proc print; run;
```

Note that, unless stated otherwise, it is assumed that we are running in a UTF-8 encoding SAS environment for all the SAS programs. In the example above, the result of the print is probably not what you expect:

```
Obs     unit
 1     mmol/L
 2     µmol/
```

Output 1. Output for Example 1

The unit `µmol/L` is not correct. However, the length of unit seems to be 6 for both mmol/L and µmol/L.

EXAMPLE 2 USE LENGTH STATEMENT

Now let us change the program into:

```
data temp;
   length unit $ 7;
   unit = 'mmol/L'; output;
   unit = 'µmol/L'; output;
run;
proc print; run;
```

Results in the output (which is now as expected):

```
Obs     unit
 1     mmol/L
 2     µmol/L
```

Output 2. Output for Example 2

This is what happened. wlatin1 is a single-byte character set, meaning that each character could be stored in one byte. But UTF-8 is a multi-byte character set, meaning that characters need 1, 2, 3 or even 4 bytes to be stored. The special character µ, needs 2 bytes with UTF-8 encoding.

EXAMPLE 3 LENGTH FUNCTION VS. KLENGTH FUNCTION

Let us look at another example:

```
data temp;
   length unit $ 7;
   unit = 'mmol/L'; output;
   unit = 'µmol/L'; output;
run;
data temp;
   set temp;
   len  =  length(unit);
   klen = klength(unit);
run;
proc print; run;
```

The output is as follows:

```
Obs     unit      len     klen
 1     mmol/L      6        6
 2     µmol/L      7        6
```

Output 3. Output for Example 3

Notice that the LENGTH function returns different results: the length of ‘mmol/L’ is 6 and the length of ‘µmol/L’ is 7. So in other words, ***the length function returns the number of bytes, not the number of characters in the string.***

When you use the K-function of function LENGTH, you see that KLENGTH(unit) is the same for ‘mmol/L’ and ‘µmol/L’, both return the number of characters, which is 6.

It is important to understand that when you process multi-byte data that you should use the DBCS string functions (K functions) instead of our ‘normal’ functions. The K functions do not make assumptions about the size of characters (number of bytes) in a string; it is a character-based offset function, while the ‘normal’ functions are byte-based.

A byte-based offset assumes that the starting position specified for a character is the byte position of that character in the string. For single-byte data, since one character is always one byte in length, you can assume that the second character in the string begins in byte two of the string. However, if the data in the string is multi-byte, the data in the second byte can be one of the following, depending on the data and encoding of the data:

- the second character in the string, or
- the second byte of a 2-byte character, or
- the first byte of the second multi-byte character in the string.

Example of other K-functions includes but not limited to:

KLEFT, KRIGHT, KLENGTH, KLOWCASE, KUPCASE, KREVERSE, KSCAN, KSTRIP, KSUBSTR, KTRANSLATE.

See Reference [2] for more information.

EXAMPLE 4 TRANSLATE FUNCTION VS. KTRANSLATE FUNCTION

One last example to demonstrate the difference between function TRANSLATE and KTRANSLATE:

```
data temp;
   unit='µmol/L';
   unit_translate  = translate (unit,'u','µ'); ** repl. µ with u-> unexpected outcome;
   unit_ktranslate = ktranslate(unit,'u','µ'); ** repl. µ with u-> expected outcome;
run;
proc print; run;
```

The output is as follows:

```
                          unit_         unit_
Obs      unit       translate     ktranslate
 1      µmol/L      u mol/L         umol/L
```

Output 4. Output for Example 4

READING AND WRITING DATA BETWEEN WLATIN1 AND UTF-8

Now that we know the differences between wlatin1 and UTF-8 encoding, next let us look at the consequences for importing and creating wlatin1-encoded files in SAS UTF-8 environment.

IMPORT EXTERNAL FILES

SAS reads and writes external files using the current session encoding. This means that the system assumes that the external file uses the same encoding as the SAS session, which does not have to be the case.

Suppose we have a file TT.TXT, which contains the following three records:

```
01;DAY 1
02;DAY 1;The subject had a fever of 39.6°C.
02;DAY 2;You can earn €2500,- when you participate in a study
```

And run the following code:

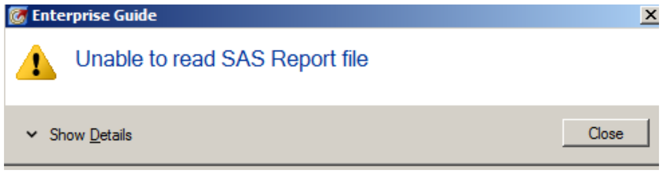
```
data temp;
   length subject $ 4 visit $ 10 com $ 100;
   infile '\\fileserver\userid\tt.txt' dlm=";" missover lrecl=200;
   input  subject $ visit $ com $;
run;
proc print; run;
```

The result is as follows:

```
Obs     subject    visit     com
 1        01       DAY 1
 2        02       DAY 1     The subject had a fever of 39.6�C.
 3        02       DAY 2     You can earn �2500,- when you participate in a study
```

Output 5. Output for Importing a Text File with Special Characters

Notice that the special characters ° and € are not imported. The log file shows no errors or warning for this. If you are using SAS Enterprise Guide (EG) and when you have "SAS report" selected in SAS EG as a 'Result Format' (see Tools Options, category Results general) then an error will show up, notifying you there are issues.



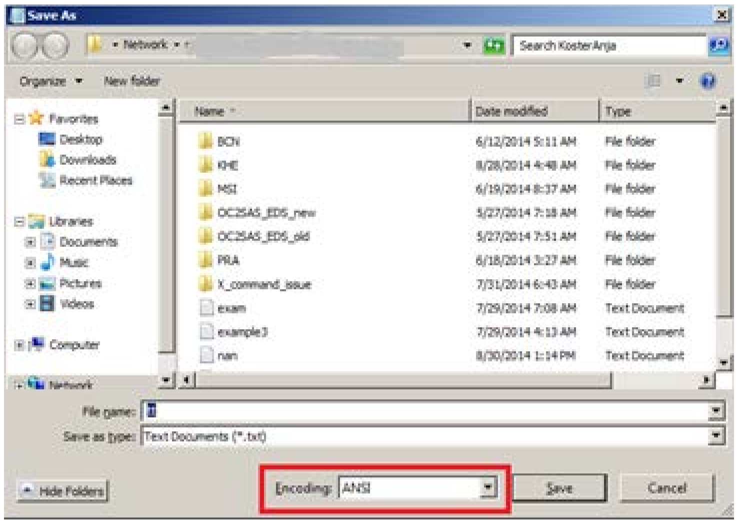
Unable to read SAS Report file

Figure 1. SAS Enterprise Guide Error When Importing a Text File with Special Characters

This problem is caused by the fact that file TT.TXT uses another encoding than your SAS session (i.e., UTF-8). By default, .txt files will be created in ANSI encoding. ANSI is an extension of ASCII characters. It includes all ASCII characters with an additional 128 characters, like µ.When the encoding of the TT.TXT file is UTF-8 this problem will not occur. See Section FAQ on how you can recognize the encoding of a text file.

There are a few solutions for this import issue:

1) Change the encoding of the file to UTF-8. Open the file in Notepad, go to File, Save as. Then change to encoding from ANSI into UTF-8 and click button [Save]



Name	Date modified	Type
BCN	6/12/2014 5:11 AM	File folder
KHE	8/28/2014 4:48 AM	File folder
MSI	6/19/2014 8:37 AM	File folder
OC2SAS_EDS_new	5/27/2014 7:18 AM	File folder
OC2SAS_EDS_old	5/27/2014 7:51 AM	File folder
PRA	6/18/2014 3:27 AM	File folder
X_command_issue	7/31/2014 6:43 AM	File folder
exam	7/29/2014 7:08 AM	Text Document
example3	7/29/2014 4:13 AM	Text Document
nan	8/30/2014 1:14 PM	Text Document

File name:
Save as type: Text Documents (*.txt)
Encoding: ANSI

Figure 2. Choose Encoding of Text File

2) When file is received from sponsor, it is not recommended to edit or re-save the file. In SAS we can use the ENCODING= option. That option tells SAS what the encoding is of the original file and SAS can transcode this to UTF-8 automatically. Just add encoding='wlatin1' (not ansi) to the infile statement:

```
data temp;
   length subject $ 4 visit $ 10 com $ 100;
   infile '\\fileserver\userid\tt.txt' dlm=";" missover lrecl=200 encoding='wlatin1';
   input subject $ visit $ com $;
run;
proc print; run;
```

The file is now imported correctly:

```
Obs     subject    visit    com
 1        01       DAY 1
 2        02       DAY 1    The subject had a fever of 39.6°C.
 3        02       DAY 2    You can earn €2500,- when you participate in a study
```

Output 6. Output for Importing a Text File with Encoding Option

3) Another solution is to use the KCVT function after you did the import in SAS without the ENCODING= option.

The KVCT function is of the form:

```
outstring = KVCT (instring, enc_in, enc_out);
```

Where,
instring - Input character string
enc_in - Encoding of instring
enc_out – Encoding of out string
outstring – Results of transcoding instring from enc_in to enc_out.

This function translates a variable from one encoding into another:

```
data temp;
   length subject $ 4 visit $ 10 com $ 100;
   infile '\\fileserver\userid\tt.txt' dlm=";" missover lrecl=200;
   input subject $ visit $ com $;
run;
proc print; run;

data temp;
   set temp;
   com=KCVT (com, 'wlatin1','UTF-8');
run;
proc print; run;
```

The file is now imported correctly:

```
Obs    subject    visit    com
 1       01       DAY 1
 2       02       DAY 1    The subject had a fever of 39.6�C.
 3       02       DAY 2    You can earn �2500,- when you participate in a study

Obs    subject    visit    com
 1       01       DAY 1
 2       02       DAY 1    The subject had a fever of 39.6°C.
 3       02       DAY 2    You can earn €2500,- when you participate in a study
```

Output 7. Output for Importing a Text File before and after KCVT Function

The encoding option has the advantage that it works for the complete file; while the KCVT function has to be used for each variable that contains 'strange' characters.

Importing excel sheets usually do not have this issue. This because the encoding value is saved in the excel sheet itself, so SAS can automatically convert this to its session encoding.

For example, the following code will import tt.xls:

```
proc import replace datafile='\\fileserver\userid\tt.xls' dbms=excelcs out=temp; run;
proc import replace datafile='\\fileserver\userid\tt.xls' dbms=excel out=temp; run;
```

The following code will import tt.xlsx:

```
proc import replace datafile='\\fileserver\userid\tt.xlsx' dbms=excel out=temp; run;
proc import replace datafile='\\fileserver\userid\tt.xlsx' dbms=excelcs out=temp; run;
proc import replace datafile='\\fileserver\userid\tt.xlsx' dbms=xlsx out=temp; run;
```

However, keep in mind that ***DBMS=XLS does not support the multiple-byte characters!*** Thus, you cannot create (or import) via PROC EXPORT/PROC IMPORT a .XLS file with characters like µ in UTF-8 encoding SAS environment.

CREATE EXTERNAL FILES

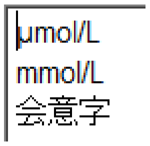
Creating a .TXT or an Excel file (using PROC EXPORT and DBMS=XLSX) from a UTF-8 encoded dataset is no problem.

Creating TEXT File

See the following example programs.

```
data temp;
   length unit $ 20;
   unit = 'µmol/L';  output;
   unit = 'mmol/L'; output;
   unit = '会意字';   output;
run;
** This creates a correct text file outp1.TXT (with encoding is UTF-8);
data _null_;
   set temp;
   file '\\fileserver\userid\outp1.txt';
   put @1 unit $ ;
run;
```

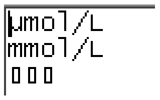
When opening a TXT file it also depends on server environment/browser if all characters are displayed correctly. For example, when you open the file outp1.TXT on Unicode file server (notepad) you will see:



```
µmol/L
mmol/L
会意字
```

Figure 3. The Special Characters Displayed Correctly in Unicode System

However, when you open the same file (with notepad) from wlatin1 file server, it will not display correctly:



```
µmol/L
mmol/L
▯▯▯
```

Figure 4. The Special Characters Cannot be Displayed in wlatin1 System

Creating Excel File

One can use the following code to create a XLSX file.

```
** This creates a correct xlsx file outp1.XLSX;
proc export data=temp dbms=xlsx outfile='\\fileserver\userid\outp1.xlsx' replace; run;
```

To create a correct XLS file, you can create a XLSX first. Then open in Excel 2010 and save as type 'Excel 97-2003 Workbook'. The code below will not be able to create a correct XLS file.

```
** This creates an incorrect xls file outp1.XLS;
proc export data=temp dbms=xls  outfile='\\fileserver\userid\outp1.xls'  replace; run;
```

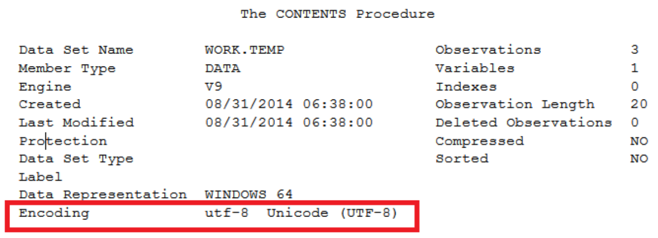
Or, one can use the KCVT function to create single-byte characters like we have in wlatin1 (as long as all characters can be converted to single byte in wlatin1):

```
data temp;
   set temp;
   unit=KCVT (unit,'UTF-8','wlatin1');
run;
** This creates a correct text xls file outp2.XLS (for the µ sign);
proc export data=temp dbms=xls
     outfile='\\fileserver\Userid\outp2.xls' replace;
run;
```

Note that `'会意字'` can not be represented as a single-byte character. Thus, to get this Asian character in XLS, you need to create XLSX first and then save as 'Excel 97-2003 Workbook'.

WORKING WITH SAS DATASETS WITH WLATLIN1 ENCODING

Sometimes, we may have SAS dataset encoded in wlatin1, for example, received from vendor. To see the encoding for a SAS dataset, run PROC CONTENTS for the dataset.



```
                              The CONTENTS Procedure

Data Set Name        WORK.TEMP                  Observations          3
Member Type          DATA                       Variables             1
Engine               V9                         Indexes               0
Created              08/31/2014 06:38:00        Observation Length    20
Last Modified        08/31/2014 06:38:00        Deleted Observations  0
Protection                                      Compressed            NO
Data Set Type                                   Sorted                NO
Label
Data Representation  WINDOWS_64
Encoding             utf-8  Unicode (UTF-8)
```

Figure 5. Use PROC CONTENTS to Check the Encoding of a SAS Dataset

When a SAS dataset has encoding wlatin1, you can deal with it in UTF-8 environment. In fact, SAS will automatically convert it to UTF-8. However, be careful when the dataset contains characters like µ. When the length of the variable is not long enough, data will be truncated. When this happens, SAS will give as error message in the log.

Let us look at one example. Run the following program in a wlatin1 environment.

```
libname x '\\fileserver\userid';

data x.temp;
   unit ='µmol/L';
run;
proc print; run;
```

The result will be:

```
Obs     unit
 1     µmol/L
```

Output 8. Output for a SAS Dataset Created in wlatin1 Environment

Now run the following code in the UTF-8 environment.

```
libname x '\\fileserver\Userid';
proc print data=x.temp;
run;
```

The result is:

```
Obs     unit
 1     µmol/
```

Output 9. Output for a wlatin1 Dataset Opened in UTF-8 Environment

The log tells us:

```
NOTE: Data file X.TEMP.DATA is in a format that is native to another host, or the file
encoding does not match the session encoding. Cross Environment Data Access will be
used, which might require additional CPU resources and might reduce performance.

WARNING: Some character data was lost during transcoding in the dataset X.TEMP. Either
the data contains characters that are not          representable in the new encoding
or truncation occurred during transcoding.
```

The note tells us that the dataset X.TEMP is not UTF-8 (which is correct, because it is wlatin1).

The warning tells us that something goes wrong. In this case truncation occurred.

You need to increase the variable length to prevent truncation during transcoding of the data to UTF-8.

To convert a wlatin1 dataset X to a UTF-8 dataset Y in a UTF-8 environment, there are a few options and some pitfalls.

OPTION 1 USE DATA STEP

```
data y;
   set x;
run;
```

Creates Y in UTF-8, while X is created in wlatin1. SAS gives a message in the LOG that dataset x is in another encoding. If will give an error when data cannot be migrated from wlatin1 to UTF-8. Never ignore these error messages.

Please note that the statements below will NOT change the encoding (if the same dataset name is used). When X is in wlatin1, x stays in wlatin1.

```
data x;
   set x;
run;
```

You can force the transcoding by specifying that it needs to become UTF-8, using the dataset option ENCODING=.

```
data x(encoding='UTF-8');
   set x;
run;
```

OPTION 2 USE PROC DATASETS

The second approach is to use PROC DATASETS as below:

```
proc datasets lib=libname;
   modify x/correctencoding='UTF-8';
run;
```

However, this way is NOT recommended: it only changes the encoder indicator but not actually translate the data itself!

OPTION 3 USE PROC MIGRATE

When you would like to convert multiple SAS datasets from wlatin1 into UTF-8, you can use PROC MIGRATE.

```
proc migrate in=inlib out=outlib;
run;
```

This migrates all SAS datasets in libname inlib to libname outlib. It retains SAS datasets labels as well. Note that inlib and outlib should be two different locations.

CREATE SAS DATASETS WITH WLATLIN1 ENCODING IN UTF-8 ENVIRONMENT

In other times, we might need to deliver wlatin1 encoded datasets to client. However, by default the SAS datasets will get encoding UTF-8 in a UTF-8 environment. In the following, we will discuss two ways to create wlatin1 encoded datasets within a UTF-8 SAS system.

OPTION 1 USE THE OUTENCODING OPTION IN THE LIBNAME

For example:

```
libname y '\\fileserver\userid' outencoding='wlatin1';
data y.temp_;
x='µmol/L';
run;
proc contents; run;
proc print; run;
```

Note that the dataset y.temp_ is created with encoding wlatin1 and not UTF-8. The variable length of unit is automatically set to 7.

However, you still get note in log:

```
    NOTE: Data file Y.TEMP_.DATA is in a format that is native to another host, or the
    file encoding does not match the sessionencoding. Cross Environment Data Access
    will be used, which might require additional CPU resources and might reduce
    performance.
```

OPTION 2 USE THE ENCODING OPTION WHEN YOU CREATE THE DATASET

For example:

```
libname y '\\fileserver\userid';
data y.temp (encoding='wlatin1');
x='µmol/L';
run;
proc contents; run;
proc print; run;
```

This creates the same output and log as first example.

Be careful with datasets and changing encoding: run the following program in SAS UTF-8 environment:

```
data hello;
   a=1;
run;
proc contents data=hello; run;
data hello (encoding='wlatin1');
   b=1;
run;
proc contents data=hello; run;
```

The listing shows us that the first dataset hello is in UTF-8 and the second dataset is in wlatin1, as expected.

But now run:

```
proc datasets;
   delete hello;
quit;
data hello (encoding='wlatin1');
   a=1;
run;
proc contents data=hello; run;
data hello ;
   b=1;
run;
proc contents data=hello; run;
```

And see that the first dataset hello is in wlatin1, and the second is also in wlatin1. You need to explicitly say:

`data hello (encoding='UTF-8');` for the second dataset, to become UTF-8.

But when the first dataset is created in wlatin1 environment, this is not needed. See example below.

Run the following code in wlatin1 environment.

```
libname x '\\fileserver\userid';
data x.hello;
   a=1;
run;
proc contents data=x.hello; run;
```

Then run the following code in UTF-8 environment.

```
libname x '\\fileserver\userid';
proc contents data= x.hello; run;
data x.hello ;
   b=1;
run;
proc contents data= x.hello; run;
```

And see that the first dataset hello is in wlatin1 and the second one is UTF-8 (created in UTF-8 system).

XPT FILES AND ENCODING

Suppose an XPT file with one or more datasets is created in wlatin1 environment, with encoding wlatin1.

What happens when you create SAS datasets from the wlatin1 XPT file in the UTF-8 environment?

UTF-8 SAS system will automatically create the SAS dataset into UTF-8, but will NOT convert the data automatically!

Run the following code in wlatin1 environment.

```
libname x '\\fileserver\userid';
data x.temp;
   length unit $10; *<-- this is important to have;
   unit ='µmol/L';
run;

libname outdat xport '\\fileserver\userid\test.xpt';
proc copy in=x out=outdat memtype=data;
   select temp;
run;
```

This creates a TEST.XPT file with one dataset temp.

Then run the code below in UTF-8 environment:

```
libname xptfile xport '\\fileserver\userid\test.xpt';
libname y '\\fileserver\userid\r';
proc copy in=xptfile out=y memtype=data; run;
proc contents data=y.temp; run;
proc print data=y.temp; run;
```

Notice that y.temp is created with the default encoding UTF-8, but that the print says (µ is missing)

```
Obs     unit
 1     mol/L
```

Output 10. Output for y.temp

To tell SAS that to convert or transcode the data, add and use the INENCODING at the libname. In other words, replace

```
libname y '\\fileserver\userid\r';
```

with

```
libname y '\\fileserver\userid\r' inencoding='wlatin1';
```

to get the data converted to UTF-8 as well.

```
Obs     unit
 1     µmol/L
```

Output 11. Output for y.temp with INENCODING= Option

Similar things can happen when an XPT file is created in UTF-8 environment and then SAS datasets are extracted in wlatin1 system. Then use

```
libname y '\\fileserver\Userid\r' inencoding='UTF-8';
```

FREQUENTLY ASKED QUESTIONS – FAQ

Some of the frequently asked encoding questions are listed below.

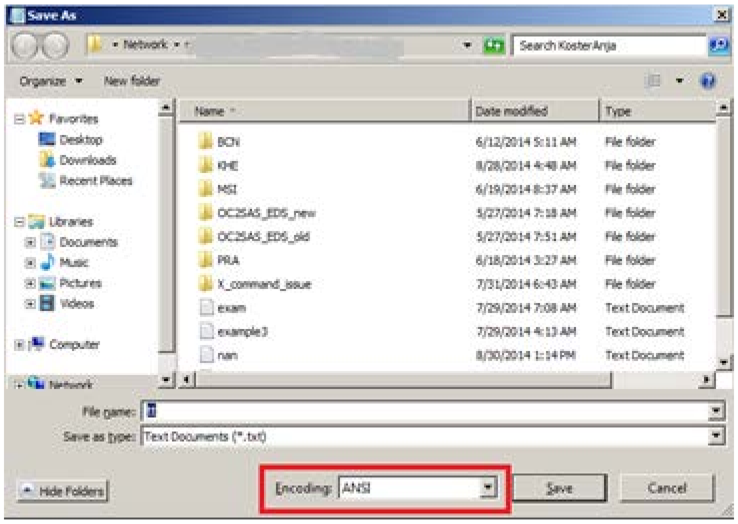
HOW TO VERIFY IF THE ENCODING OF A TEXT FILE IS ANSI OR UTF-8?

Although a text file in UTF-8 and ANSI encoding can look the same, there is a difference. You need to know how a text file is encoded, to be able to import it in SAS correctly.

When the text contains Asian characters, it is clear that it cannot be ANSI. But a ‘normal’ text file with only ASCII and extended ASCII (like µ) will look the same.

The difference is that, in a UTF-8 text file, a BOM-code is inserted (Byte Order Marker) immediately at the beginning of the file. This BOM is invisible when you open the text file in Notepad or WordPad. The BOM is represented via Byte Sequence 0xEF, 0xBB, 0xBF and some browsers display this as ï»¿

There is a way to verify the encoding using Notepad. Open the .txt with Notepad and in menu go to File, Save as. The encoding of the file is displayed at the bottom.



Name	Date modified	Type
BCN	6/12/2014 5:11 AM	File folder
KHE	8/28/2014 4:48 AM	File folder
MSI	6/19/2014 8:37 AM	File folder
OC2SAS_EDS_new	5/27/2014 7:18 AM	File folder
OC2SAS_EDS_old	5/27/2014 7:51 AM	File folder
PRA	6/18/2014 3:27 AM	File folder
X_command_issue	7/31/2014 6:43 AM	File folder
exam	7/29/2014 7:08 AM	Text Document
example3	7/29/2014 4:13 AM	Text Document
nan	8/30/2014 1:14 PM	Text Document

File name:
Save as type: Text Documents (*.txt)
Encoding: ANSI

Figure 6. Finding the Encoding of a Text File

HOW TO CHECK WHAT DEFAULT ENCODING IS USED IN YOUR CURRENT SAS SESSION?

Use the following statement in SAS:

```
proc options option=encoding; run;
```

The results will be something like

```
ENCODING=wlatin1  Specifies default encoding for processing external data.
```

or

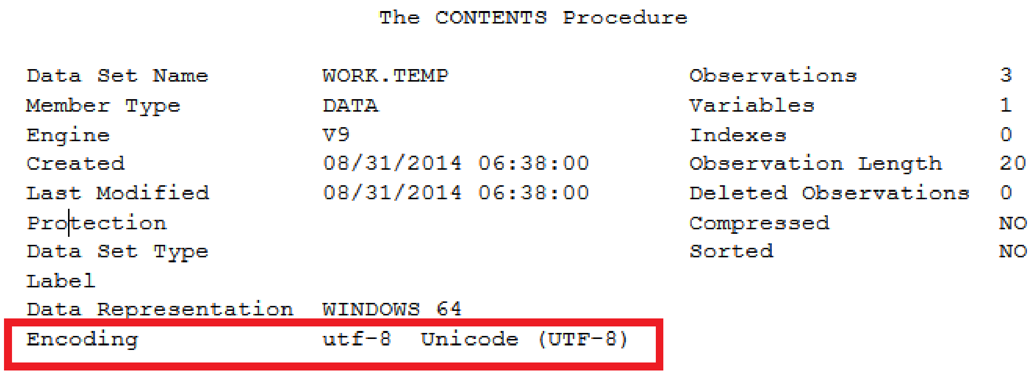
```
ENCODING=UTF-8    Specifies the default character-set encoding for the SAS session.
```

HOW TO CHECK WHAT THE ENCODING IS OF A SAS DATASET?

Use the following statement in SAS:

```
proc contents data=<dataset>; run;
```

Which results in (part of the output)



```
                          The CONTENTS Procedure

Data Set Name          WORK.TEMP                  Observations          3
Member Type            DATA                       Variables             1
Engine                 V9                         Indexes               0
Created                08/31/2014 06:38:00        Observation Length    20
Last Modified          08/31/2014 06:38:00        Deleted Observations  0
Protection                                        Compressed            NO
Data Set Type                                     Sorted                NO
Label
Data Representation    WINDOWS_64
Encoding               utf-8  Unicode (UTF-8)
```

Figure 7. Check the Encoding of a Dataset

However, even though the dataset has encoding UTF-8, this does not necessarily mean the data itself is in UTF-8.

HOW CAN I SEE HOW MANY BYTES A CHARACTER CONSISTS OF?

Suppose you like to know how many bytes it takes to store the euro-sign € in UTF-8.

In SAS run the following statement.

```
      %put %length(€);
```

Then the length, which is 3, will be printed in the log.

HOW CAN I SEE IF THERE ARE MULTI-BYTE CHARACTERS IN A VARIABLE?

If you have a dataset X with a variable COM and you like to see quickly if there are multi-byte characters in it:

```
data _null_;
   set x;
   if length(com) ne klength(com)
   then put 'WARNING, multi bytes characters present in: ' com;
run;
```

This will show the record(s) in the log. If true, never decrease the length of the COM variable than the maximum of all LENGTH(com) to prevent truncation.

CONCLUSION

In this paper, we have discussed the impact of changing the encoding from wlatin1 to UTF-8. UTF-8 is a multi-byte character set while wlatin1 is a single-byte character set. When moving data between these two encodings, SAS will automatically transcode the data into the current SAS session's encoding. However, SAS may not be able to do the transcoding correctly all the time due the different data representation in the two encodings. For example, if the data are not handled carefully and correctly, truncation or miscoding may happen.

We have used concrete examples to show how to handle the datasets and files between UTF-8 and wlatin1 encodings. Since UTF-8 can represent much more characters than ASCII characters, some special characters cannot be transcoded from UTF-8 into wlatin1. Thus, we might need to have some (automatic) mechanisms to replace them so as to be compatible with FDA's submission guidelines for tabulation data.

REFERENCES

[1] http://support.sas.com/resources/papers/Multilingual_Computing_with_SAS_94.pdf

[2] SAS(R) 9.4 National Language Support (NLS): Reference Guide, Third Edition, Internationalization Compatibility for SAS String Functions. Available at http://support.sas.com/documentation/cdl/en/nlsref/67399/HTML/default/viewer.htm#p1pca7vwjjwucin178l8qddjn0gi.htm

CONTACT INFORMATION

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

Name: Hui Song
Enterprise: PRA Health Sciences
Address: 731 Arbor Way, Suite 100
City, State ZIP: Blue Bell, PA 19422
Work Phone: 215-591-1138
E-mail: songhui@prahs.com

Name: Anja Koster
Enterprise: PRA Health Sciences
Address: PO Box 200, 9470 AE
City, State ZIP: Zuidlaren, Netherlands
Work Phone: +31 50 402 2671
E-mail: kosteranja@prahs.com