PhUSE 2016

Paper PP10

Non-Printable and Special Characters? … BYTE me!

Louise Sims, Phastar Ltd., London, UK

ABSTRACT

As a form of computer character encoding, non-printable and special characters (NPSC) can be tricky to interpret, often appearing as blank spaces or unfathomable squares. In the pharmaceutical industry, the popularity of global work-sharing and data exchange has meant NPSC are appearing more frequently within clinical trial data and can be perplexing for the unsuspecting programmer. By taking the time to understand exactly what NPSC are allows us to create more efficient methods of programming for them and enable our data to become more meaningful.

This paper gives a brief introduction to NPSC, highlighting some of the programming difficulties which can be encountered from them, including importing external spreadsheets containing NPSC into SAS®, managing NPSC within SAS data and forcing NPSC to appear in outputs when needed. Solutions to these common problems are also given, so the reader need never be intimidated when faced with square boxes again.

INTRODUCTION

Non-printable and special characters originate as a result of computer encoding. Computers can only interpret strings of numbers and therefore encoding is used as a method to represent character symbols, called character (or coding) sets, through numbers, called coding points. A Bit is the smallest unit of data and a Byte consists of 8 bits. Coding sets can either be Single Byte Coding Sets (SBCS) which mean that each coding point is represented in 1 byte or less, or Multi-Byte Coding Sets (MBCS), where multiple bytes are required in order to include all of the coding points in the coding set.

American Standard Code for Information Interchange (ASCII) is a 7-bit encoding set (and therefore a SBCS), which is a standard way of representing characters. It contains 128 character points (0 to 127) within its coding set which represent characters such as the English alphabet, numbers, punctuation and control characters (see Appendix 1.1). The extended ASCII coding set is an 8-bit encoding set which follows on from the first 128 characters of ASCII, containing character points 128 to 255. These additional coding points include letters with accents used in European languages such as French and German. However, there are different versions of the Extended ASCII coding set; see Appendices 1.2 and 1.3 for two different versions.

Extended Binary Coded Decimal Interchange Code (EBCDIC) is another 8-bit encoding set which is only used in some IBM machines. EBCDIC represents characters in coding points 0 to 255, in the same way as ASCII, but different characters are represented at different coding points; therefore the two coding sets are not immediately compatible. ASCII is used much more widely as a standard compared to EBCDIC.

The Unicode set is a much larger character set than both ASCII and EBCDIC, with each character between 8-bits and 32-bits in size, and it is a MBCS. The Unicode set consists of characters which can be used in most of the world's languages, including Traditional Chinese and Cyrillic. Within the Unicode set, there are different methods of encoding, named as Unicode Transformation Formats (UTF).

1. UTF-8 – this is a variable-length encoding method used for characters between 8-bits and 32-bits in size. So if a character can be represented in 8-bits, only 1 byte will be used. If a character requires 16-bits then 2 bytes will be used. This variable-length method ensures additional space is not wasted when characters could be represented by a smaller amount of code. The first 128 characters of the UTF-8 are the same as ASCII to allow compatibility.

2. UTF-16 – this is also a variable-length encoding method for character points between 16-bits and 32-bits in size. So again, this saves space as only the number of bytes needed to represent the character is used.

3. UTF-32 – this is a 32-bit, non-variable encoding form. Therefore all characters encoded in this set use 4 bytes for each coding point, meaning that this coding method takes up a lot more space than UTF-8 and UTF-16. However, it is a useful coding method for characters which require 4 bytes to be identified.

HOW DOES CHARACTER ENCODING CAUSE NPSC?

There are three main categories of character points in the ASCII set; non-printable characters (points 0 – 31), printable characters (points 32 – 127) and special characters in the extended ASCII code (points 128 – 255).

The non-printable characters consist of control characters which were originally designed for old Teletype machines to control where a Teletype would start printing from. Most of these characters are obsolete now and cannot be displayed correctly within data; however tabs, carriage returns and new line feeds are still used when inputting data today. Since there is no way of printing control characters, when these are present within data, they will often appear as blank spaces or represented by odd symbols, but they still can affect the appearance of a string of text. For instance, a carriage return will push text after the carriage return onto the next line. By contrast, printable characters can easily be interpreted and will appear in data as expected with no problems.

Special characters from the extended set can be interpreted fine when the same version of the extended set is used to input as is to read the data. However, since there are different versions of the Extended ASCII set, if a different version is used to input the data compared to that which reads the data, the character points will correspond to different characters, which can lead to strange symbols being displayed instead of the intended character.

The same concept applies to character points from the Unicode set. Depending on the choice of Unicode format used to encode data, again the source format may be different to the receiving format. For instance, within a SAS session, the encoding version used is defined at the initialization and is generally dependent on location. A SAS session ran in the UK is likely to be using the WLATIN1 encoding version, whereas a SAS session in Egypt is likely to be using WARABIC. If data was transferred from the two sessions using letters or characters only common to one of the two encoding versions, it would not appear correctly in the second session, and may appear instead as odd symbols or square boxes since the encoding version is unable to display the original character. This is common with global clinical trials being completed in sites across the world which use different languages and likely different encoding sets, particularly where manual data entry is permissible.

To determine the default encoding version being used within a SAS session, the following code can be used.

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

```
2911  proc options option=encoding;
2912  run;

    SAS (r) Proprietary Software Release 9.4  TS1M3

 ENCODING=WLATIN1  Specifies the default character-set encoding for the SAS session.
NOTE: PROCEDURE OPTIONS used (Total process time):
      real time           0.00 seconds
      user cpu time       0.00 seconds
      system cpu time     0.00 seconds
      memory              20.46k
      OS Memory           33532.00k
      Timestamp           08/01/2016 04:06:31 PM
      Step Count                        43  Switch Count  0
```

Figure 1: The SAS log window, displaying the encoding for the SAS session.

Checking the log window after submitting the code above shows the encoding version in the current SAS session is WLATIN1, which corresponds to the ASCII table in Appendix 1.1 and the extended ASCII table ISO 8859-1 in Appendix 1.2. For a full list of the different encoding sets within the Windows operating system, see Appendix 1.4. Encoding sets for other operating systems can be found in the SAS support documentation.

NPSC WITHIN SAS DATA

IDENTIFYING NPSC WITHIN DATA

It can be tricky to detect NPSC within data, simply because they are often present in long free-text fields and it can be hard to spot them just by looking at the data. One useful SAS function which helps to detect NPSC in data is the NOTPRINT function.

Syntax: NOTPRINT (“*character string*” <, *start*>)

Where *character string* is the text to search for non-printable characters, s*tart* is the starting position within the character string to start searching.

The NOTPRINT function searches for any non-printable characters within a specified string and returns the position of the first occurrence of a non-printable character. The optional argument start can be used to specify where the search commences from; a positive value of start means the search starts to the right and a negative value of start means the search starts to the left. If no value for start is used, the default start position is the beginning of the string.

For example, the code below can be used to check if any non-printable characters are present in the inclusion/exclusion criterion text variable, IETEST, from clinical trial data. A value of 0 in NTPRNT means the string in IETEST does not contain any non-printable characters. A value of greater than 0 in NTPRNT indicates the location of the first non-printable character in that string.

```
data ie;
set sdtm.ie;
  ntprnt=notprint(ietest);
run;
```

Once aware of the presence of a non-printable character, the RANK function can be used to identify which non-printable character is in the data.

Syntax: RANK (*expression*)
where *expression* is a character string

The RANK function returns the coding point in either the ASCII or EBCDIC coding set of a given character. The coding point returned depends on the operating environment being used, i.e. for an IBM machine it is likely to be using EBCDIC coding, whereas Windows, UNIX and Macintosh machines will likely be using ASCII coding.

Returning to our inclusion/exclusion example, we can use the following code to determine which non-printable characters are present in the data.

```
data ie;
set sdtm.ie;
  ntprnt=notprint(ietest);
  if ntprnt>0 then id=rank(substr(ietest,ntprnt,1));
run;
```

Since we have obtained the location in the string for where the first NPSC is within the NTPRNT variable, we can then use the SUBSTR function in conjunction with the RANK function to identify the non-printable character at that point in the string.

IETESTCD	IETEST	IECAT	NTPRNT	ID
EXCL206	Subject has recurrent or chronic infections, defined as >= 3 infections requiring anti-microbials over the past 12 months prior to screening.	EXCLUSION	0	
EXCL207	Subject has a significant concurrent medical condition, including: Type 1 diabetes; Poorly controlled type 2 diabetes (Hemoglobin A1c > 8.5); Symptomatic heart failure (New York Heart Association clas	EXCLUSION	0	
EXCL208	Subject is pregnant or breast feeding, or planning to become pregnant whileenrolled in the study, up to the subject last study visit including the 30-dayfollow-up period.	EXCLUSION	76	10
EXCL209	Female subject is not willing to abstain from sexual intercourse or use 2 highlyeffective forms of birth control for the duration of the study including the follow-upperiod (except women at least 3	EXCLUSION	81	10

Indicates the presence of NPSC and the position of the first instance.

Identifies the ASCII coding point of the first NPSC.

Figure 2: Identifying NPSC using the RANK function.

Running on a Windows OS, we can see the first non-printable character in both of the two values of IETEST above correspond to ASCII value 10. Checking Appendix 1.1, we can see this is the line feed control character. By copying and pasting the text from one of the values with the line feed we can see how the non-printable character is acting on the text.

```
Subject is pregnant or breast feeding, or planning to become pregnant while
enrolled in the study, up to the subject last study visit including the 30-day
follow-up period.
```

Figure 3: Line feeds within SAS data displayed in Editor window.

REMOVING NPSC WITHIN DATA

Once NPSC are detected within data, it is often required to remove them so they do not adversely affect the interpretation or appearance of the data. Where possible, the first step should generally be for Data Management to remove NPSC from the source data. However, if it is not possible for Data Management to remove the NPSC, the following methods can be used instead.

Method 1: Using the COMPRESS function

Syntax: COMPRESS (*source* <, *characters*><, *modifiers*>)

Where *source* is the character string in which characters are to be removed from, *characters* is the optional argument where certain characters are specified to be removed from the source expression and *modifiers* are constants which modify the COMPRESS function.

The COMPRESS function can be used to remove certain characters from a string. The modifiers k and w are particularly useful for our purposes. The k modifier keeps the characters in the string which satisfy the arguments. The w modifier adds printable characters to the list of the string. So using both modifiers in conjunction means the COMPRESS function would be modified to keep all of the printable characters in the string, meaning all of the non-printable characters would be removed.

In the example below, the COMMENT variable contains a few different NPSC which need to be removed.

CRFDT_INT	CRFNO_STD	COMMENT
25JUL2013:00:00:00.000	IPADMIN_1	PATIENT DID NOT COME◄TO HIS W32 ?VISIT IN THE SCHEDULED TIME WINDOW BETWEEN 10 JUL → 24 JUL 2013. WE PERFORMED THE W32 VISIT ON 25 JUL 2013, BUT NO IP WAS ASSIGNED TO THE PATIENT BY IWRS.

Figure 4: Multiple, different NPSC within a free-text field.

```
data co_raw;
  set raw.comments_all;
  cmnt=compress(comment, , "kw");
run;
```

CRFDT_INT	CRFNO_STD	COMMENT	CMNT
25JUL2013:00:00:00.000	IPADMIN_1	PATIENT DID NOT COME◄TO HIS W32 ?VISIT IN THE SCHEDULED TIME WINDOW BETWEEN 10 JUL → 24 JUL 2013. WE PERFORMED THE W32 VISIT ON 25 JUL 2013, BUT NO IP WAS ASSIGNED TO THE PATIENT BY IWRS.	PATIENT DID NOT COME TO HIS W32 VISIT IN THE SCHEDULED TIME WINDOW BETWEEN 10 JUL 24 JUL 2013. WE PERFORMED THE W32 VISIT ON 25 JUL 2013, BUT NO IP WAS ASSIGNED TO THE PATIENT BY IWRS.

Figure 5: NPSC removed in the CMNT variable using the COMPRESS function.

In Figure 5 we can see that by using the COMPRESS function with the k and w modifiers has removed all of the NPSC from the comment. This function is ideal to use when removing multiple, different NPSC.

Method 2: Using the BYTE function

Another method to remove unwanted NPSC is to use the BYTE function to directly pick a NPSC, using its ASCII/ EBCDIC coding value.

Syntax: BYTE (*n*)

Where *n* is a numeric value between 0 and 255 and represents the coding point in ASCII or EBCDIC coding sets.

The BYTE function returns the character represented by *n* in either ASCII or EBCDIC, depending on the operating environment.

In the example below, there are multiple soft hyphens which appear in the verbatim medication text in the concomitant medications data.

CMVERB	CMPREF	INDIC	INDIC_STD	INDICSP	DOSE
STREPSILS -- (ALCOHOL 2,4--DICHLOROBENZYLICUS, AMYLMETACRESOLUM)	STREPSILS /00200801/	Other	88	SORE THROAT	1.000

STREPSILS -- (ALCOHOL 2,4--DICHLOROBENZYLICUS, AMYLMETACRESOLUM)

Figure 6: Multiple of the same NPSC within a free-text field.

First we can use the NOTPRINT and RANK functions to find the ASCII coding value of the NPSC. Then we can use the BYTE function in conjunction with the TRANWRD and COMPBL functions to replace all occurrences of this NPSC with a single blank space.

```
data cm;
  set raw.conmeds;
  check=notprint(cmverb);
  if check>0 then id=rank(substr(cmverb,check,1));
  cmverb_=compbl(tranwrd(cmverb, byte(id), " "));
run;
```

CMVERB	CMVERB_	CMPREF	INDIC	INDIC_STD	INDICSP	DOSE
STREPSILS -- (ALCOHOL 2,4--DICHLOROBENZYLICUS, AMYLMETACRESOLUM)	STREPSILS (ALCOHOL 2,4 DICHLOROBENZYLICUS, AMYLMETACRESOLUM)	STREPSILS /00200801/	Other	88	SORE THROAT	1.000

Figure 7: NPSC removed using the BYTE function.

REPLACING NPSC WITHIN DATA

There may be occasions where the NPSC displayed in the data is clearly the wrong character. This can often occur where the encoding version used to input the data is different to the version used to read the data. For example, the text below contains an arrow, whereas this should really be an apostrophe.

Comment Reference	Date/Time of Comment	Comment
CONCOMITANT MEDICATION	2009-08-27	CONTINUATION OF PREVIOUS COMMENT: AND ALSO BECAUSE OF CONSIDERATION OF THE PATIENT→S RENAL FUNCTION

CONTINUATION OF PREVIOUS COMMENT: AND ALSO BECAUSE OF CONSIDERATION OF THE PATIENT→S RENAL FUNCTION

Figure 8: Misinterpreted non-printable character in a free-text field.

Using the RANK and SUBSTR functions, we can identify the coding point of the NPSC is ASCII character code 26, which represents the substitute control character. We can then use the BYTE function to replace any occurrences of this NPSC with the desired apostrophe.

```
data co;
  set sdtm.co;
  id=rank(substr(coval,83,1));
  coval_=tranwrd(coval, byte(id), "'");
run;
```

This then gives the following value in the COVAL_ variable, so the text now reads “PATIENT’S”.

COREF	CODTC	COVAL	COVAL1	POS	ID	COVAL_
CONCOMITANT MEDICATION	2009-08-27	CONTINUATION OF PREVIOUS COMMENT: AND ALSO BECAUSE OF CONSIDERATION OF THE PATIENT→S RENAL FUNCTION		83	26	CONTINUATION OF PREVIOUS COMMENT: AND ALSO BECAUSE OF CONSIDERATION OF THE PATIENT'S RENAL FUNCTION

Figure 9: Replacing a NPSC with another character using the BYTE and TRANWRD functions.

NPSC IN OUTPUTS

LACK OF SPECIAL CHARACTERS WHEN REQUIRED

There are sometimes cases where special characters are needed in outputs and they are not contained in the data, so we need to manually add these special characters in. For example, the units of some laboratory results contain Greek letters which are not contained in the WLATIN1 encoding set. For instance, the Greek letter µ represents micro in scientific units and the SI unit for Creatinine is µmol/L.

Method 1: Adding NPSC to the data

In the example below, a “u” is used instead of a “µ” for the value of PARAM; Creatinine (µmol/L).

LBSCAT	PARAMCD	PARAM	PARCAT1	AVAL
RENAL FUNCTION	CREAT_S	Creatinine (umol/L)	STANDARD UNIT	79.560
RENAL FUNCTION	CREAT_S	Creatinine (umol/L)	STANDARD UNIT	79.560
RENAL FUNCTION	CREAT_S	Creatinine (umol/L)	STANDARD UNIT	88.400

Figure 10: Laboratory unit using “u” instead of special character “µ”.

One solution is to add the special character to the data, so the correct units would carry through to the output. As long as we know the ASCII coding value of the special character and the encoding version of the current SAS session, we can add the special character to the data using the BYTE function.

Checking Appendix 1.2 in the ISO 8859-1 extended ASCII character set, we can see µ is represented by the coding point 181. We can then either use the TRANWRD function to directly replace any occurrences of "u" with "µ" or alternatively we can concatenate the special character with a substring of the unit data as below.

```
data adlb;
  set adam.adlb
  where paramcd="CREAT_S";
  si_unit=byte(181)||substr(param,14,5);
run;
```

This then creates the following values in the data which can be used as required in outputs.

LBSCAT	PARAMCD	PARAM	PARCAT1	AVAL	SIUNIT
RENAL FUNCTION	CREAT_S	Creatinine (umol/L)	STANDARD UNIT	79.560	µmol/L
RENAL FUNCTION	CREAT_S	Creatinine (umol/L)	STANDARD UNIT	79.560	µmol/L
RENAL FUNCTION	CREAT_S	Creatinine (umol/L)	STANDARD UNIT	88.400	µmol/L

Figure 11: Special character µ added into variable text using the BYTE function.

Note that if the special character which needed to be added was not in the ASCII coding set, but was instead part of the Unicode set, the SAS function UNICODE can be used instead of the BYTE function to refer to a Unicode character point.

Method 2: Adding NPSC to the output

If we had a case where we didn't need to change the data, but instead wanted to add a special character into part of a heading or label within an output, we could add the special character to the code used to generate the output.

For example, the output below has the SI units for Total Bilirubin as umol/L in both the output title and subheading, whereas this should be µmol/L.

Table 14a-7.4. Summary of Total Bilirubin (umol/L) Shifts from Baseline (One-way) by Treatment Group
Safety Analysis Set

	Maximum Grade for Increased Value						
Baseline Grade	NA n (%)	0 n (%)	1 n (%)	2 n (%)	3 n (%)	4 n (%)	Total n (%)
Total Bilirubin (umol/L)							
Placebo (N=33)							
NA	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
0	0 (0)	32 (97)	1 (3)	0 (0)	0 (0)	0 (0)	33 (100)
1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)

Figure 12: Output needing special characters to be added to the title and subheading.

For an ODS output, we can use the ODS escape character to display special characters in the output. First ensure the ODS escape character is defined before the PROC REPORT section of code in the output program as below.

```
ods escapechar = "^";
```

Then within the PROC REPORT section of the output program, the following code can be added which will display the special character µ in a line within the main body of the output. The same code can also be added to the title text to display the letter µ in the output's title. The key point to remember is to ensure the words "unicode" and "mu" are contained within curly brackets immediately after the ODS escape character.

```
compute before page / style={just=l};
  line @1 'Total Bilirubin (^{unicode mu}mol/L)';
endcomp;
```

Other special characters can be added in the same way, but by replacing "mu" with the keyword for the required special character. For example "alpha" for letter α and "beta" for letter β. Using the code above in the PROC REPORT generates the following output.

Table 14a-7.4. Summary of Total Bilirubin (µmol/L) Shifts from Baseline (One-way) by Treatment Group
Safety Analysis Set

Baseline Grade	Maximum Grade for Increased Value: NA n (%)	0 n (%)	1 n (%)	2 n (%)	3 n (%)	4 n (%)	Total n (%)
Total Bilirubin (µmol/L)							
Placebo (N=33)							
NA	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
0	0 (0)	32 (97)	1 (3)	0 (0)	0 (0)	0 (0)	33 (100)
1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)

Figure 13: Output with special characters added in using the ODS escape character.

THE NEED FOR MORE SPACE IN OUTPUTS

There may be situations where we need extra space between text in an output in order to improve the appearance, particularly in outputs with long text strings which need formatting, as in the example below.

Table 14-8.2.1. Summary of Cardiovascular Events Committee (CEC) Adjudicated Cardiac Events from First Dose of IP Through End of Study
Safety Analysis Set

Category / Type / Primary Cause	Drug A: Overall Variable Dosing (N = 179) n (%)	Overall 140 mg Q2W (N = 71) n (%)	Overall 210 mg Q2W (N = 398) n (%)	All Drug A (N = 648) n (%)
Death	1 (0.6)	0 (0.0)	6 (1.5)	7 (1.1)
Cardiovascular	1 (0.6)	0 (0.0)	3 (0.8)	4 (0.6)
Stroke	1 (0.6)	0 (0.0)	0 (0.0)	1 (0.2)
Sudden cardiac death	0 (0.0)	0 (0.0)	3 (0.8)	3 (0.5)
Non-cardiovascular	0 (0.0)	0 (0.0)	3 (0.8)	3 (0.5)
Hemorrhage (excluding hemorrhagic stroke or bleeding in setting of coronary revasc)	0 (0.0)	0 (0.0)	1 (0.3)	1 (0.2)
Suicide	0 (0.0)	0 (0.0)	2 (0.5)	2 (0.3)

Figure 14: Output with formatting issues

In Figure 14, the Primary cause row “Hemorrhage (excluding hemorrhagic stroke or bleeding in setting of coronary revasc)” spans two lines due to its length and the small amount of space in the output. To improve the appearance of this row, where the text wraps onto the next line, ideally we would want the text to be indented by the same amount as the first line to clearly show it is part of a primary cause row and not a category or type row.

One solution to this problem is to add a special character, often referred to as a “Hidden Dragon” or the “Invisible character” which creates a blank space within a text field. The Hidden Dragon character can be added using the shortcut key sequence Alt+255, which only works using the number pad on a keyboard. Note this is different to the blank space represented by ASCII code 32, as that blank space gets compressed when processed by SAS if is a leading or trailing blank, whereas the Hidden Dragon does not get compressed, making it very useful when adding spaces before the start or after the end of a character string.

All characters within the ASCII coding set can be input by using Alt and the corresponding ASCII coding value. Note that for the extended ASCII characters, the Alt+code uses the CP437 extended ASCII coding set (also referred to as the PC/OEM 437 extended set), displayed in Appendix 1.3. By looking at the CP437 extended ASCII set, we can see the coding value 255 corresponds to a blank space, which is why Alt+255 on the number pad gives us the Hidden Dragon. The ISO 8859-1 extended ASCII set can also be input using the shortcut key sequence, but by adding a zero in front of the coding value. For instance, if we wanted to add the special character ±, we would type Alt+0177. Whereas if we were to use the CP437 extended ASCII set, we would type Alt+241 to obtain the character ±.

Returning to our example above, we can use the Hidden Dragon multiple times to create an indent each time the text in the first column wraps onto a second line. In the code below, ORD=3 is assigned to each of the Primary Cause rows, ORD=2 is assigned to each of the Type rows and ORD=1 is assigned to each of the Category rows. For the Primary Cause rows, the text in the first column of the output is split up so the first 51 characters in the text are concatenated with 7 Hidden Dragon spaces (obtained by pressing Alt+255 seven times on the number pad), and the remainder of the text. An alternative to adding the Hidden Dragon using the Alt+ ASCII code point is to use the BYTE

function instead. This method was used for the Type row, which requires an indent of 4 spaces. Note when using the BYTE method, 4 separate BYTE functions are required for the concatenation, as below, since each BYTE function will return 1 character.

```
data final1;
  set final;

** For the primary cause row, 7 blank spaces are concatenated between the text **;
  if ord=3 then do;
    if length(txt)>51 then col1=substr(txt,1,51)||"       "||substr(txt,52);
    else col1=txt;
  end;

 ** For the Type row, the BYTE function adds 4 blank spaces for the indentation **;
  else if ord=2 then do;
    if length(txt)>51 then col1=substr(txt,1,51)||byte(160)||byte(160)||byte(160)||
    byte(160)||substr(txt,52);
    else col1=txt;
  end;
  ** For the Category row, no indentation is needed **;
  else col1=txt;
run;
```

Table 14-8.2.1. Summary of Cardiovascular Events Committee (CEC) Adjudicated Cardiac Events from First Dose of IP Through End of Study
Safety Analysis Set

Category Type Primary Cause	Drug A Overall Variable Dosing (N = 179) n (%)	Overall 140 mg Q2W (N = 71) n (%)	Overall 210 mg Q2W (N = 398) n (%)	All Drug A (N = 648) n (%)
Death	1 (0.6)	0 (0.0)	6 (1.5)	7 (1.1)
Cardiovascular	1 (0.6)	0 (0.0)	3 (0.8)	4 (0.6)
Stroke	1 (0.6)	0 (0.0)	0 (0.0)	1 (0.2)
Sudden cardiac death	0 (0.0)	0 (0.0)	3 (0.8)	3 (0.5)
Non-cardiovascular	0 (0.0)	0 (0.0)	3 (0.8)	3 (0.5)
Hemorrhage (excluding hemorrhagic stroke or bleeding in setting of coronary revasc)	0 (0.0)	0 (0.0)	1 (0.3)	1 (0.2)
Suicide	0 (0.0)	0 (0.0)	2 (0.5)	2 (0.3)

Figure 15: Output with formatting issues fixed through using the "Hidden Dragon" special character.

IMPORTING EXTERNAL DATA CONTAINING NPSC INTO SAS

One common source of NPSC within SAS data is from external data which is imported into SAS. If data is manually entered into a spreadsheet, for instance, at one location in the world, and then sent to be imported to another location, it is quite possible the two locations may be using different character encoding sets. Therefore, the external spreadsheet is likely to contain NPSC that the receiving site cannot interpret.

If external data is received which contains NPSC, many times the data will be able to be imported into SAS and the data can be cleaned up as mentioned in the sections above. However, there can sometimes be cases where SAS is unable to import the data correctly because of the presence of NPSC. In this case, we would need to clean the data before it can be read into SAS.

IMPORTING .CSV, .XLS AND .TXT FILES WITH NPSC INTO SAS

A common method of receiving external data is through spreadsheets, and in particular, in .csv files. Sometimes, .csv files which contain NPSC will not import into SAS in the required format. For example, as in the .csv file below.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2	Study Cod	Country	Site No	Enrol No	Rand No	Issue No	Category	Issue Des	Action	Created Date	Created B	Resolved	Closed	Closed Date
3	D5160C00	Australia	304	304301	1014	V16	Major Pro	9.2 -	Patient wi	25-Aug-15	Ms Amy Jo	Y	Y	25-Aug-15
4	D5160C00	Canada	1002	1002311	1095	V32	Major Pro	April 21,	site to	20-Jul-15	Ms. Susy F	N	N	
5	D5160C00	Canada	1003	1003301	1840	V7	Major Pro	Subject	Site to	4-Mar-15	Ms Sylvia	N	Y	24-Jun-15

Figure 16: A .csv file containing NPSC to be imported into SAS.

When trying to import this file into SAS, we get the following results.

	VAR2	VAR3	VAR4	VAR5	VAR6	VAR7	VAR8
Study Code	Country	Site No	Enrol No	Rand No	Issue No	Category	Issue Description
D5160C00003	Australia	304	304301	1014	V16	Major Protocol Deviation	"9.2 - Follow-Up tu (RECIST 1.1) until progression perfor (every 6 weeks du week).
Patient E0304301 CT schedule has been changed from 6 weekly to 12 weekly due to discomfort during the procedure; this is related to the cold contrast required to be consumed prior to scan. This was approved by Global Study Physician and Safety Team	along with the PI and also by St Vincent's Hospital Sydney HREC; approval letter received 11Aug2015.						

Figure 17: .csv file not imported correctly into SAS due to NPSC.

As we can see from Figure 17, the data has lost its structure during the import process and cannot be used in this current state. The cause of this loss of structure is due to the presence of carriage returns and line feeds contained in the original .csv file, which causes issues for SAS when trying to process them. To resolve this problem, one solution is to manually clean up the non-printable control characters in the .csv file before importing into SAS, using Notepad++.

Notepad++ has a Find and Replace feature which allows the user to search within a .txt file and replace any troublesome NPSC with a blank space, for example. Data received in either .csv, .xls or .txt files containing NPSC can all be cleaned up fairly easily within Notepad++, but it's worth mentioning the extended Find and Replace feature needed for this task is not available in the basic Notepad software.

Using either Ctrl+H or Search then Replace in the Toolbar within Notepad++ will bring up the following window.

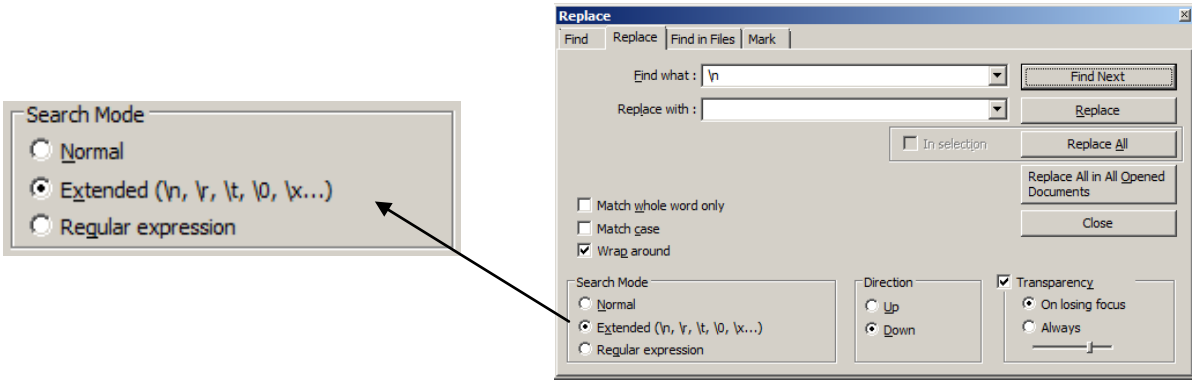

Figure 18: Find and Replace feature in Notepad++.

Note that it is important to ensure "Extended (\n, \r, \t, \0, \x…)" is selected in the Search Mode section. You can then search for different NPSC using the Notepad++ codes. For example, a line feed is represented by \n, a carriage return by \r, a tab by \t and a null space by \0. Other NPSC can be searched by their coding value, using \xddd where ddd is the ASCII/Unicode coding value.

If we edit our .csv file in Notepad++, we can see where some of the problems are. Within each subject's records, there are line feeds which are causing information from one subject to be split over multiple lines. Therefore, when this is imported SAS counts each line as a new record; hence the loss of structure as shown above.

```
1  ,,,,,,,,,,,,,,,
2  Study Code,Country,Site No,Enrol No,Rand No,Issue No,Category,Issue Description,Action,Created Date,Created By,Reso
3  D5160C00003,Australia,304,304301,1014,V16,Major Protocol Deviation,"9.2 - Follow-Up tumour assessments (RECIST 1.1)
4
5  Patient E0304301 CT schedule has been changed from 6 weekly to 12 weekly due to discomfort during the procedure; th
6
7  ",Patient will continue to have 12 weekly CT scans up to disease progression. The site will also warm the contrast
8  D5160C00003,Canada,1002,1002311,1095,V32,Major Protocol Deviation,"April 21, 2015: Subject has signed and dated the
9  - Optional cross over ICF (v 2mar2015) approved 19mar2015
10 - Main ICF  version 23feb2015
```

Figure 19: Data from .csv file in Notepad++, showing the presence of NPSC within the data.

Using Find and Replace, we can remove the occurrences of the line feeds by searching for \n and replacing with a blank space. This then gives us the following file in Notepad++.

```
1  ,,,,,,,,,,,,,,,
2  Study Code,Country,Site No,Enrol No,Rand No,Issue No,Category,Issue Description,Action,Created Date,Created By,Resol
3  D5160C00003,Australia,304,304301,1014,V16,Major Protocol Deviation,"9.2 - Follow-Up tumour assessments (RECIST 1.1)
4  D5160C00003,Canada,1002,1002311,1095,V32,Major Protocol Deviation,"April 21, 2015: Subject has signed and dated the
5  D5160C00003,Canada,1003,1003301,1840,V7,Major Protocol Deviation,"Subject was not using adequate contraceptive measu
6  D5160C00003,Canada,1005,1005301,1029,V5,Major Protocol Deviation,Subject 1005301-ePRO not completed at baseline and
7  D5160C00003,Canada,1005,1005302,1035,V6,Major Protocol Deviation,"Subject 1005302-neurological assessment not done a
8  D5160C00003,Canada,1005,1005301,1029,V7,Major Protocol Deviation,"Subject 1005301-chemotherapy at Cycle 2 given less
```

Figure 20: Data from .csv with line feeds removed using Notepad++.

If we now try importing the .csv file back into SAS with the line feeds removed, we can see the data is now imported successfully in the desired format.

STUDY_CODE	COUNTRY	SITE_NO	ENROL_NO	RAND_NO	ISSUE_NO	CATEGORY	ISSUE_DESCRIPTION
D5160C00003	Australia	304	304301	1014	V16	Major Protocol Deviation	9.2 - Follow-Up tumour assessments (RECIST 1.1) until objective disease progression performed outside of window (every 6 weeks during the study +/- 1 week).Patient E0304301 CT schedule has been changed from 6 weekly to 12 weekly due to discomfort during the procedure; this is related to the cold contrast required to be consumed prior to scan. This was approved by Global

Figure 21: Data from .csv file successfully imported into SAS after removing NPSC.

CONCLUSION

In summary, NPSC often occur within clinical trial data as either blank spaces or substitute symbols. They are caused due to variations in the character encoding methods used to input and read data, which is increasingly common with global clinical trials since different encoding sets can be used for different languages. Despite more universal forms of character encoding such as ASCII and Unicode in place, most computer systems run on encoding versions which follow the native language, as encoding sets which encompass multiple alphabets are extremely large and require longer processing times.

NPSC can cause problems within clinical trial data and can often be hard to spot within long, free-text fields. They can cause misinterpretation of data since the NPSC are not usually the intended character and often do not make sense within the context of the rest of the data. Furthermore, control characters in particular can cause problems when importing external data into SAS; causing data to lose its structure and become unusable.

Using SAS functions such as NOTPRINT and RANK can help to detect the presence of NPSC within data. When NPSC are found, the first port of call should generally be for Data Management to remove them from the source data. However, in cases where this is not possible, functions such as COMPRESS and BYTE can be used to clean the data within SAS. For external data which needs to be cleaned before importing correctly, the Find and Replace feature within Notepad++ is a useful tool for quickly removing NPSC.

Despite often needing to remove NPSC, there are some times where they can be useful; particularly when producing outputs. For instance, when needing to include characters from other languages not available on the keyboard, the BYTE or UNICODE functions can be used to display a character from other alphabets in data and outputs. Characters such as the “Invisible character” or “Hidden Dragon” can also be useful for adding spaces to improve the appearance of outputs.

Although the presence of NPSC within data can be frustrating as they can affect the meaningfulness of data and quite often hinder processes we are trying to complete, once we understand the cause of NPSC and how to deal with them, they become much more manageable and even useful in the right circumstances.

REFERENCES

[1] SAS 9.4 National Language Support (NLS) - http://support.sas.com/documentation/cdl/en/nlsref/67964/HTML/default/viewer.htm#titlepage.html
[2] http://superuser.com/questions/545461/replace-carriage-return-and-line-feed-in-notepad
[3] http://lookuptables.com
[4] http://www.w3schools.com/charsets/ref_html_8859.asp

ACKNOWLEDGMENTS

I would like to thank Lewis Meares and John McDade from Phastar Ltd. for sharing their experiences with NPSC, enabling me to provide more information and solutions to problems they have encountered.

CONTACT INFORMATION

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

Louise Sims
Phastar Ltd.
Unit 2A, 2 Bollo Lane,
London
W4 5LE
Email: louise.sims@phastar.co.uk

APPENDICIES

Appendix 1.1 ASCII Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Appendix 1.2 Extended ASCII Table (ISO 8859-1 version, also referred to as CP1252)

Dec	Chr	Dec	Chr	Dec	Chr	Dec	Chr	Dec	Chr	Dec	Chr	Dec	Chr	Dec	Chr
128	€	144		160		176	°	192	À	208	Ð	224	à	240	ð
129		145	‘	161	¡	177	±	193	Á	209	Ñ	225	á	241	ñ
130	‚	146	’	162	¢	178	²	194	Â	210	Ò	226	â	242	ò
131	ƒ	147	“	163	£	179	³	195	Ã	211	Ó	227	ã	243	ó
132	„	148	”	164	¤	180	´	196	Ä	212	Ô	228	ä	244	ô
133	…	149	•	165	¥	181	µ	197	Å	213	Õ	229	å	245	õ
134	†	150	–	166	¦	182	¶	198	Æ	214	Ö	230	æ	246	ö
135	‡	151	—	167	§	183	·	199	Ç	215	×	231	ç	247	÷
136	ˆ	152	˜	168	¨	184	¸	200	È	216	Ø	232	è	248	ø
137	‰	153	™	169	©	185	¹	201	É	217	Ù	233	é	249	ù
138	Š	154	š	170	ª	186	º	202	Ê	218	Ú	234	ê	250	ú
139	‹	155	›	171	«	187	»	203	Ë	219	Û	235	ë	251	û
140	Œ	156	œ	172	¬	188	¼	204	Ì	220	Ü	236	ì	252	ü
141		157		173		189	½	205	Í	221	Ý	237	í	253	ý
142	Ž	158	ž	174	®	190	¾	206	Î	222	Þ	238	î	254	þ
143		159	Ÿ	175	¯	191	¿	207	Ï	223	ß	239	ï	255	ÿ

Appendix 1.3: Extended ASCII Table (OEM 437 or also known as CP437)

128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Appendix 1.4 SAS Encoding Values in Windows Operating System

Single-Byte Encodings for Windows

Description	Windows ENCODING= Value	MS-DOS ENCODING= Value	IBM-PC ENCODING= Value
Arabic	warabic	msdos720	pcoem864
Baltic	wbaltic	msdos775	pcoem921
Central Europe	wlatin2	not applicable	pcoem852
Cyrillic	wcyrillic	not applicable	pcoem866
			pcoem855
Estonia	wbaltic	not applicable	pcoem922
European	not applicable	not applicable	pcoem858
Farsi	not applicable	not applicable	pc1098
French Canadian	wlatin1	not applicable	pcoem863
Greek	wgreek	msdos737	not applicable
Hebrew	whebrew	not applicable	pcoem862
Indian Script Code	not applicable	not applicable	pciscii806
Nordic	not applicable	not applicable	pcoem865
Portuguese	wlatin1	pcoem860	not applicable
Thai	not applicable	not applicable	pcoem874
Turkish	wturkish	not applicable	pcoem857
USA	wlatin1	not applicable	pcoem437
Vietnamese	wvietnamese	not applicable	not applicable
Western	wlatin1	not applicable	pcoem858

Windows Double-Byte Encodings

Description	PCMS ENCODING= Value	No Vendor ENCODING= Value
Traditional Chinese	ms-950	big5
Simplified Chinese	ms-936	not applicable
Japanese	ms-932	shift-jis
Korean	ms-949	not applicable

Note: Windows also supports the utf-8 Unicode encoding.