Paper SAS4561-2020

The SAS® Encoding Journey: A Byte at a Time

Mickaël Bouedo, SAS Institute

ABSTRACT

UTF-8 is becoming the most dominant character encoding for the World Wide Web. It supports a great number of characters from many languages, including English, and is compatible with ASCII characters. While all characters are one byte in WLATIN1, the same familiar characters might take several bytes in UTF-8. This major difference in data representation imposes several challenges for SAS programmers. Transcoding errors, truncation, or garbage characters such as ? might appear unexpectedly when your data is processed. To help you in this endeavor to overcome these challenges, this paper explains the basics of character encoding and how it is handled in SAS®. It defines some common terms such as ASCII, single-byte character set (SBCS), multi-byte character set (MBCS), and Unicode. It also introduces SAS functions and macros that are available to detect potential problematic characters and to make it easier to fix the problem.

INTRODUCTION

Mr. and Mrs. Doe just bought a brand-new car to replace their 15-year-old car and they go to the gas station to fill the tank. They go to their favorite pump, the pump number 1 which only provides unleaded gas 93. Mr. Doe fills the tank like he did for many years with his old car. However, Mr. and Mrs. Doe forgot that the new car has a different engine and expects a different type of fuel. They leave the gas station and after a couple of miles, the car stops with the dashboard lighting up like a Christmas tree with many red, blue, and green lights.

We could easily draw two analogies with Mr. and Mrs. Doe's story to a user of SAS®.

First, we could see SAS as the car, the SAS session encoding as the engine, and the data's encoding as the type of fuel. If you start your SAS session with a UTF-8 encoding but provide WLATIN1-encoded data, then, like Mr. and Mrs. Doe, your SAS log could be as colored as their car's dashboard and you could run into some trouble.

The second analogy is related to the type of fuel aka the data's encoding. For many years, WLATIN1 and LATIN1 were the dominant encoding for English and other Western languages. This is not the case anymore. UTF-8 data is now the most used encoding on the web, and UTF-8 encoding is becoming the default encoding for many applications, including SAS Viya. If your industry still requires data in US-ASCII, using UTF-8 is still compatible as it will be described in this paper.

Hopefully, unlike the car, SAS has many options to help avoid the trouble. You can easily change your session encoding to match your data's encoding, or you can easily transcode the data to be in the expected encoding of your SAS session.

You will want to look under the hood to find problems that you could run into by processing data that does not match your session encoding and learn the various way to avoid them. First, it's important to understand a little bit about character encoding.

WHAT IS A CHARACTER ENCODING?

In order to debug or avoid any transcoding issues within SAS, it is important to understand the characteristics of an encoding and the differences among the different character encoding schemes.

Text data is made of characters. Examples of characters include the letters from the Latin alphabet e, é, ë, or the Japanese characters コ ン サ ー ト from the katakana syllabary. Other examples of characters include punctuation such as the exclamation mark (!) or symbols such as the Greek letter micro (µ).

Characters that are used by a language or group of languages are grouped into a character set (also called a repertoire). Examples of a character set are the Western European alphabets and the Japanese kanji and kana syllabaries.

Computers only work with bits and a group of 8 bits called a byte or an octet, which can be represented as numbers under different formats such as binary, decimal, or hexadecimal. So, in order to convert a sequence of characters into bytes and vice versa, computers and software need an encoding scheme, or encoding for short.

An encoding maps each character in a character set to a unique numeric representation, which results in a table of all code points. This table is referred to as a code page, which is an ordered set of characters in which a numeric index (or code point value) is associated with each character.

There are many different character sets and character encodings where the mapping between bytes, code points, and characters is also different.

FROM 7-BIT ASCII TO UTF-8 – A LITTLE BIT OF HISTORY

A single-byte character set (or SBCS) is an encoding where each character is encoded with one byte. ASCII and extended ASCII encoding are SBCS encodings. When more than one byte is needed to represent a character, like in the UTF-8 encoding, the character set is called a multibyte character set (MBCS).

ASCII OR US-ASCII

ASCII, an acronym for American Standard Code for Information Interchange, is one of the first standard encodings adopted in the early 1960s by computers to represent characters. ASCII is also called sometimes lower ASCII or 7-bit ASCII because only the first 7 bits in a byte are used. The usage of the first 7 bits allows a byte to represent a maximum of 128 code points or characters in the decimal range 0 to 127, or [00-7F] in hexadecimal representation. Despite this relatively small number, it has enough room to include and represent all lower-case and upper-case Latin letters, along with each numerical digit, common punctuation marks, spaces, tabs, and other control characters, all used in English alphabet. US-ASCII is the SAS name for this encoding.

ASCII EXTENSIONS

The 7-bit ASCII encoding with its 128 characters could not satisfy everyone's needs because it does not contain national characters such as ä, è, ç, Σ, ж, and so on. However, a byte using 8 bits became the norm for computers, and storage for characters was soon expanded to make use of that 8th bit. By using this extra bit in a byte, a new range is available from decimal range 128 to 255, or [80-FF] in hexadecimal representation. It's possible to support an additional 128 code points or possible characters to the existing 128 7-bit ASCII ones.

The characters supported in this new range are often referenced to as the upper ASCII characters. Alternative names include extended ASCII characters, 8-bit ASCII characters, or high ASCII. The encodings that extend the 7-bit ASCII are called extended ASCII encodings (or 8-bit ASCII encodings).

The extra code points in the new code range [80-FF] are mostly used to represent foreign or national characters.

Extended ASCII encodings are platform- and locale-dependent. Many extended ASCII encodings have been created, such as the OEM/DOS encodings, ISO 8859 family encodings, Windows encodings, or encodings for the iOS Mac system. Table 1 shows various encoding values existing on different platforms and supporting different languages. The name between parenthesis is the SAS name of the encoding.

Supported Languages	OEM/DOS Encodings	ISO Encodings	Windows Encodings
Western European	CP437 (PCOEM437) CP850 (PCOEM850)	8859-1 (LATIN1) 8859-15 (LATIN9)	Windows-1252 (WLATIN1)
Eastern European	CP852 (PCOEM852)	8859-2 (LATIN2)	Windows-1250 (WLATIN2)
Russian	CP866 (PCOEM866)	8859-5 (CYRILLIC)	Windows-1251 (WCYRILLIC)
Greek	CP737 (MSDOS737)	8859-7 (GREEK)	Windows-1253 (WGREEK)
Hebrew	CP862 (PCOEM862)	8859-8 (HEBREW)	Windows-1255 (WHEBREW)

Table 1. Examples of Extended ASCII Encoding Names

LATIN1 or ISO 8859-1

The SAS encoding called LATIN1, Latin1, ISO 8859-1, or Latin part 1 is one of these extended ASCII encodings. Other possible alias names for this encoding include ibm-819, IBM819, cp819, 8859_1, csISOLatin1, iso-ir-100, ISO_8859-1:1987, l1, 819, or Windows-28591. This encoding is used throughout the Americas and Western Europe on Unix-based systems and has the following characteristics:

- Lower ASCII characters are in the range [00-7F].
- Like all the ISO encodings, LATIN1 reserves the range [0x80-0x9F] to control characters.
- The remaining code points, range [A0-FF], contain characters used in languages originated from Western European countries. However, some characters needed for French, Dutch, and Finnish were missing. It was corrected by introducing LATIN9 (ISO 8859-15).

LATIN9 or ISO 8859-15

LATIN9 and LATIN1 are very similar and easily confused, which leads to common issues. LATIN9 differs from LATIN1 by supporting eight different characters, most notably the euro character €. Other characters include Š, Ž, Œ, and their lowercase versions. LATIN9 is often used on Unix systems in Europe.

WLATIN1 or Windows-1252

WLATIN1 or code page 1252 is a superset of LATIN1. Only the characters assigned to the code range [80-9F] are different and contain additional characters. Display 1 shows all the characters in that range.

It includes the eight characters that were added to LATIN9 but at different code positions. It also includes several punctuation marks, arithmetic symbols, and quotation marks.

The quotation marks in that range are also called "smart" quotation marks, as these characters are often used by an autocorrecting feature in word processors and some text editors. The nice-looking shape is preferred to the straight 7-bit ASCII version that is typed from your keyboard. These characters often confuse SAS users and may lead to unexpected transcoding issues, which are explained in the next section as well as later.

80	€ 20AC		‚ 201A	ƒ 0192	„ 201E	… 2026	† 2020	‡ 2021	ˆ 02C6	‰ 2030	Š 0160	‹ 2039	Œ 0152		Ž 017D	
90		‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014	˜ 02DC	™ 2122	š 0161	› 203A	œ 0153		ž 017E	Ÿ 0178

Display 1. Code Range [80-9F] for WLATIN1 Encoding

UNICODE AND UTF-8 TO THE RESCUE

However, unlike the lower 7-bits ASCII, the characters defined in the upper range of the extended ASCII encodings were never standardized. For example, the euro currency sign (€) has a different code point in LATIN9 (0xA4), WLATIN1 (0x80), and PCOEM858 (0xD5).

Also, there is no easy way to use two or more non-English alphabets in the same document, and alphabets with more than 256 characters like Chinese and Japanese had to use entirely different encoding schemes.

In the late 1980s, a new standard was proposed: Unicode. Unicode is the universal character standard that includes the characters in most of the world's writing systems. The latest version of Unicode contains a repertoire of over 137,000 characters.

In Unicode, each character is represented by its own unique number, which is officially written in hexadecimal preceded by U+. The euro sign is always U+20AC and the Greek alpha is U+03B1.

Unicode also defines the rules for mapping each of those characters to a numeric value. UTF-8, UTF-16, and UTF-32 are the three most-known forms of encodings to process characters defined in Unicode. They only differ in how many bytes they use to encode each character.

UTF-8, which is by far the most dominant encoding on the world wide web, uses one to four bytes to encode a character. Table 2 shows the number of bytes that are required to encode characters used in different languages.

Another important characteristic of UTF-8 is backward compatibility with the lower ASCII characters. It means that the first 128 characters available in UTF-8 match the characters in ASCII and have the exact same byte. In other words, ASCII maps 1:1 onto UTF-8. Any character not in ASCII takes up two or more bytes in UTF-8. Table 2 below shows a few characters encoded in different encodings.

Character	Description	Unicode value	UTF-8		WLATIN1 (1 byte)	CP437 (1 byte)
			Code points	# of Bytes		
A	Capital letter A	U+0041	0x41	1	0x41	0x41
€	Euro sign	U+20AC	0xE2 0x82 0xAC	3	0x80	n/a
•	Bullet point	U+2022	0xE2 0x80 0xA2	3	0x95	0x07
α	Small Greek alpha	U+2013	0xE2 0x82 0x93	3	n/a	0xE0
à	Small A-grave	U+00E0	0xC3 0xA0	2	0xE0	0x85

Table 2. Example of Characters Encoded in Different Encodings

HOW DO THESE ENCODINGS AFFECT SAS?

SAS, like any software dealing with character data, needs to know what encoding to use when processing these characters. The SAS session encoding, like the engine in a car, is a very important option of SAS. The data's encoding is like the type of fuel we provide to SAS. It's recommended that both encodings match; otherwise transcoding happens.

SAS SESSION ENCODING

The session encoding establishes the environment to process SAS syntax and SAS data sets, and to read and write characters in external files. The SAS session encoding is set using the ENCODING system option. It is a startup option only.

How to Determine Your SAS Session Encoding?

You can use one of the following methods to determine your session encoding:

```
proc options option=encoding; run;
%put SESSION ENCODING is &sysencoding;
%put SESSION Encoding=%sysfunc(getoption(encoding));
```

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

Output 1. Output from the OPTIONS Procedure and &SYSENCODING Macro

DATA ENCODING

Data characters come from different sources, including SAS data sets or external files such as a simple text file or columns from a database. The characters in a file or other data sources are stored using a character encoding to represent the data.

External Files

The encoding for some files, such as database tables, is available programmatically. However, data from external sources does not always have an encoding attribute. A visual inspection of the file might be needed to detect the encoding.

Data Sets

Data sets created in SAS®9 and SAS Viya have an encoding attribute stored in the data set header. That attribute or property is set when the data set is created. The default value is set with the same encoding as the session encoding in the SAS session used to create the data set.

How to Determine the Encoding of a Data Set?

The data set's encoding can be determined by using the CONTENTS procedure as follows. The result is shown in the Output 2 below, where we can see the encoding of the data set is UTF-8.

```
proc contents data=u8.symbols;
run;
```

The CONTENTS Procedure

Data Set Name	U8.SYMBOLS	Observations	3
Member Type	DATA	Variables	2
Engine	V9	Indexes	0
Created	02/11/2020 14:09:49	Observation Length	33
Last Modified	02/11/2020 14:09:49	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_64		
Encoding	utf-8 Unicode (UTF-8)		

Output 2. Output from PROC CONTENTS

TRANSCODING

The encoding of a data set or external file being read may differ from the session encoding. In that case, the data set must be transcoded into the session encoding for SAS to work with it.

Transcoding is the process of converting data from one encoding to another encoding. In SAS, transcoding is necessary if the session encoding and the file or data set encoding are not compatible.

SAS Files

For SAS data sets, transcoding is usually done automatically with CEDA (Cross-Environment Data Access). A note like the one below is usually printed to the log to indicate the usage of CEDA.

```
NOTE: Data file WLT1.SYMBOLS.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.
```

Output 3. Note Indicating CEDA Is Used

CEDA enables a SAS file that was created in a directory-based operating environment (for example, UNIX or Windows) to be processed in an incompatible environment or under an incompatible session encoding.

WHAT TO DO WHEN ERRORS ARE HAPPENING?

Problems, with or without error messages, may arise when transcoding data. If SAS detects an error, an error message like the following error message is displayed in the log:

```
ERROR: Some character data was lost during transcoding in the data set
WLT1.SYMBOLS. Either the data contains characters that are not
representable in the new encoding or truncation occurred during
transcoding.
```

Output 4. Transcoding Error Message Example

The error message guides you to the problem. It gives some clues as to why the issue is happening. The error message suggests two possible reasons:

1. Characters that are not representable means that the data contains characters that do not exist in the new encoding and therefore can't be represented.
2. Truncation means that a variable in the new data set does not have enough room to receive the transcoded data and therefore truncation occurred.

TROUBLESHOOTING A TRANSCODING ISSUE

The error message "`the data contains characters that are not representable in the new encoding`" indicates that the output encoding does not support some of the characters from the input encoding.

Unsupported Characters

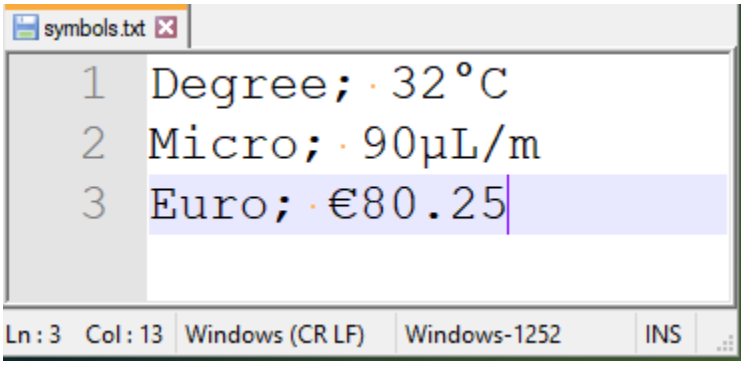
This error is common when transcoding data from UTF-8 to ASCII or any extended ASCII encodings such as WLATIN1, because ASCII encodings support a very limited set of characters compared to UTF-8. For example, if your UTF-8 data contains Japanese characters, and you attempt to read it under a WLATIN1 SAS session encoding, the transcoding will fail with the error message displayed in Output 4.

Incorrect Data Set Encoding

Another common issue that could generate this error message happens when the data set encoding attribute does not reflect the way the data is encoded. For example, the data set header says UTF-8 but the data is encoded in WLATIN1 and contains characters in the upper ASCII range. If all the data is in the lower ASCII range, it is not an issue since UTF-8 is compatible with that range.

Creating a data set with the incorrect encoding is a common mistake and it can happen without any error message as shown in the following example.

Suppose we have a WLATIN1 encoded external file, symbols.txt, which contains a few data representing unit symbols. Display 2 below shows the data in Notepad++ Text Editor. The encoding used by the text Editor appears in the taskbar. Here it says Windows-1252 (aka WLATIN1) and we can easily verify that the data displays correctly.



Display 2. External File Symbols.txt in Notepad++ Text Editor

We read the file and create a data set in a UTF-8 SAS session encoding:

```
libname mylib "path to a library";
data mylib.symbols;
  length name $10 example $ 10;
  Infile "Symbols.txt" dlm=";" missover;
  input name $ example $;
  run;
proc print;
run;
```

While the log indicates the import is successful, the resulting HTML output shows unexpected characters:

The SAS System

Obs	name	example
1	Degree	32�C
2	Micro	90�L/m
3	Euro	€80.25

Output 5. ODS Output from the PRINT Procedure

What happened? The file symbols.txt contains data encoded in WLATIN1 encoding where the symbol characters (degree, micro, and euro) are one-byte encoded characters in the upper ASCII range with the respective code points of 0xB0, 0xB5, and 0x80 in WLATIN1. Since SAS does not know the encoding of the data in the file, it defaults to the session encoding, UTF-8, to read the file.

However, in the UTF-8 encoding, the characters in that range are either the first byte of a multi-byte character or an undefined code point. Therefore, the single byte read in the file becomes invalid.

The HTML document generated by ODS and PROC PRINT replaces the invalid character with a special character called replacement character (�). It has the Unicode value of U+FFFD. In other forms of output, the invalid character may appear as an empty square box.

How to Correct the Problem?

If you still have the external source file, the easiest way to correct the problem is to specify the ENCODING= option on the INFILE or FILENAME statement to tell SAS what encoding to use to read the file:

```
infile "Symbols.txt" dlm=";" missover encoding="wlatin1";
```

If you do not have the source file anymore, and only the resulting data set is available, there are several possibilities to correct the problem. All the suggested solutions described below are from a UTF-8 SAS session encoding.

Option 1: You could use the CORRECTENCODING= option of the DATASETS prcedure to update the encoding property of the data set:

```
proc datasets library=mylib;
   modify symbols / correctencoding=wlatin1;
quit;
```

WARNING: Changing the encoding attribute of a data set does NOT transcode the data itself to the new encoding. Having a mismatch of the encoding attribute and the encoding of the data is often source of a problem such as:

- Data could be lost during transcoding.
- Unnecessary transcoding could take place.
- Unexpected results from procedures or functions could occur.

Option 2: Re-create the data set by using the ENCODING= option to tell SAS to use a different encoding to read the data set.

```
libname u8 "path to a library";
data u8.symbols; /* a new data set is created */
    set mylib.symbols(encoding=wlatin1);
 run;
```

Option 3: Use the INENCODING option on a LIBNAME statement. This option is similar to option 2. The INENCODING= option tells SAS to read any data set from that library with the specified encoding.

```
libname mylib "path to a library" inencoding="wlatin1";
libname u8 "path to a library";
data u8.symbols; /* new data set */
    set mylib.symbols;
run;
```

How to Detect and Prevent the Problem?

Not all encoding can be easily verified programmatically. Extended ASCII encodings, for example, could potentially contain any characters in the range [00-FF]. A visual inspection of the source file is usually recommended for these encodings.

However, US-ASCII or UTF-8 have specific characteristics that allow verification with tools.

There are various tools and functions that can help to verify that a data set is correctly encoded to one of the two encodings mentioned above.

The %VALIDCHS Macro: Validates the Character(s) Encoding Compatibility

The %VALIDCHS macro validates that the encoding of the characters in the data set are compatible with encoding property of the data set or a different encoding that can be specified as a parameter. The %VALIDCHS macro is available since version 9.4m5 and is in the AUTOCALL macro library.

For example, consider our incorrect data set, u8.symbols, which contains WLATIN1 encoded data but with a property header set to UTF-8. You can run the macro as follows to detect the issues, which are printed in the log as shown in Output 6.

```
%VALIDCHS(dsnm=symbols, libnm=mylib) ;
```

```
WARNING: Found invalid character in the variable: symbol at the observation
number:1.
WARNING: Found invalid character in the variable: symbol at the observation
number:2.
WARNING: Found invalid character in the variable: symbol at the observation
number:3.
ERROR: Detected the issue in the dataset: u8.symbols (encoding: utf-8).
```

Output 6. Output from the Macro %VALIDCHS

The KPROPDATA Function: Detects and Replaces Unwanted Characters

The KPROPDATA function can help to detect, remove, or convert unwanted characters. The syntax is as follows:

```
str=KPROPDATA(<instr> (<options>, <input encoding>, <output encoding>));
```

By default, this function converts the input data string to the current SAS session encoding and removes or replaces non-supported characters based on the specified option. The function allows you to change the input and output encoding.

KPROPDATA provides different options about what to do with the unwanted characters. For example, if you simply want to replace unsupported characters with a space character, use the BLANK option as the second argument to the function. You can also replace the characters with different character format such as Unicode escape notation, hexadecimal, or numeric character reference (NCR).

The SAS program below reads the UTF-8 data set symbols and replaces characters that are not supported in US-ASCII encoding with different formats. The resulting data set is shown using a PROC PRINT in Output 7.

Incompatible characters are replaced or deleted based on the option specified.
"UESC": converts the character using a Unicode escaped notation.
"TRIM": removes the character, and no replacement character is used.
"HEX": replaces the character with a hexadecimal representation.
"NCR": replaces the character using NCR notation.

```
data symbols_ascii;
  set mylib.symbols;
  uesc = kpropdata(example,"UESC", "UTF-8", "US-ASCII" );
  trim = kpropdata(example,"TRIM", "UTF-8", "US-ASCII" );
  ncr = kpropdata(example,"NCR", "UTF-8", "US-ASCII" ) ;
  hex = kpropdata(example,"HEX", "UTF-8", "US-ASCII" ) ;
run;
proc print; run;
```

The SAS System

Obs	name	example	uesc	trim	ncr	hex
1	Celcius	32°C	32\u00b0C	32C	32°C	32\xc2\xb0C
2	Micro	90µL/m	90\u00b5L/m	90L/m	90µL/m	90\xc2\xb5L/m
3	Euro	€80.25	\u20ac80.25	80.25	€80.25	\xe2\x82\xac80.25

Output 7. Results from the KPROPDATA Function

In SAS Viya 3.5, a new option "PUNC" has been added to the KROPDATA function. This option is useful to remove special characters called smart quotation marks and punctuation. See the section about WLATIN1 for a description of these smart quotation marks.

The complete list of smart quotation marks and punctuation characters processed with this option is available in the *SAS 9.4 National Language Support (NLS): Reference Guide*.

The following program converts some of these smart characters. The result is in Output 8.

```
data _null_;
 smart = ' ‹ “ – — … • ” ';
 ascii = kpropdata(smart,"PUNC");
 put smart= / ascii=;
run;
```

```
smart=‹ “ – — … • ”
ascii=< " - - ... * "
```

Output 8. Output from KPROPDATA and the PUNC Option

The BASECHAR Function: Converts Characters to Base Characters

The BASECHAR function is another way to convert some types of unwanted characters like accented characters found in the extended ASCII range.

The syntax is

```
STR=BASECHAR(<instr>(,<Unicode format type>))
```

The BASECHAR function reads the characters in the string and converts them to the base character form without the accents, based on Unicode specifications. If an output format is specified, other characters without a base character are represented with that format.

For example, the following code shows the BASECHAR function processing accented characters in a UTF-8 session encoding. It uses the format PAREN (Unicode parenthesis escape) to represent the other characters. The result is shown in Output 9.

```
data _null_;
    nc = "à é å ç ë ö µ ж";
    bc = basechar(nc,"PAREN");
    put nc= / bc=;
run ;
```

```
nc=à é å ç ë ö ж
bc=a e a c e o <u0436>
```

Output 9. Output from the BASECHAR Function Using the PAREN option

Encoding ASCIIANY: Disables the Transcoding

There are cases in which you might want to suppress the transcoding of your data. For example, a good reason would be that your data contains only ASCII characters. The ENCODING= option values ASCIIANY or ANY allow you to import your data without any transcoding.

```
data wlt1.symbols(encoding=asciiany);
   set u8.symbols;
run;
```

Using this encoding to "fix" or bypass the error message would be a bad reason and could result in data loss or corruption if you work with the data.

AVOIDING DATA TRUNCATION BY USING THE CVP ENGINE

A truncation occurs during the transcoding process because the variable's length that receives the data is too small. Truncation issues are most common when importing data from one of the extended ASCII encodings to UTF-8. For example, you import a WLATIN1 data set containing extended ASCII characters into a UTF-8 SAS session.

To demonstrate the issue, let's create a WLATIN1 SAS data set in a WLATIN1 SAS session. The data set contains a single variable defined with a length of 1 as shown in Output 10.

```
data wlt1.units;
  input unit $1.;
  datalines;
°
µ
€
run;
proc contents; run;
```

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	Unit	Char	1

Output 10. PROC CONTENTS Output Showing Length of 1

Now, in a UTF-8 SAS session, let's try to read and import this data set.

```
data u8.units;
   set wlt1.units;
run;
```

Since the data set encoding is different from our SAS session encoding, we receive in the log the CEDA note (Output 3) indicating transcoding will happen and then followed by the error message seen in Output 4.

The length specification is the number of bytes for storing character and numeric variables. In an SBCS environment such as a WLATIN1 SAS session, one byte is equivalent to one character. The format $1 means the variable can hold only one byte or one character with an SBCS encoding.

However, in UTF-8, these symbols are encoded on three bytes. The variable receives only the first byte of the three, which results in a truncated character.

In order to avoid this issue, you must increase the length of the variable receiving the data. This can be done automatically by using the CVP engine. The CVP engine expands the character variable lengths so that character data truncation does not occur. In SAS 9, the default expansion rate is 1.5. To change the rate, the CVP engine option CVPMULTIPLIER= can be specified.

```
libname wlt1 cvp "<path to library>" cvpmultiplier=3;
data u8.units;
  set wlt1.symbols;
run;
proc contents; run;
```

The resulting data is created with a length of $3 for the Unit variable as shown from PROC CONTENTS in Output 11.

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	Unit	Char	3

Output 11. Output from PROC CONTENTS Showing Length Is 3

CONCLUSION

Technologies have changed over the past decades, and the world is more connected than ever. Data comes from all around the world. Unicode and its UTF-8 encoding are a perfect and consistent way to support all the characters from these different languages and cultures.

Like Mr. and Mrs. Doe, adapting to a new environment is not always without mistakes. The transition from an SBCS environment such as WLATIN1 to UTF-8 is no different and transcoding problems may happen.

SAS provides many different solutions. This paper covers some of them through a specific case and examples. Each problem is different, and the expectations on the solution are also different.

However, understanding the mechanism of transcoding, the basic characteristics of encodings, and the concept of session encoding in SAS are critical. The concepts explained in this paper will help you to identify and solve these types of problems in a faster way and with great confidence.

REFERENCES

Unicode Consortium. "Frequently Asked Questions." Available at https://www.unicode.org/faq/basic_q.html.

SAS Institute Inc. 2016. *SAS 9.4 National Language Support (NLS): Reference Guide*. 5th ed. Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

I would like to express my gratitude to the people who helped me to write this paper: Elizabeth Bales for the multiple ideas, suggestions, and reviews. Harold Weinbrecht and Liz Simon for the technical review.

RECOMMENDED READING

- Kiefer, Manfred. 2012. *SAS Encoding: Understanding the Details.* Cary, NC: SAS Institute Inc.
- Xie, Edwin (You). 2020. "Your Data Will Go On: Practice for Character Data Migration." *Proceedings of the SAS Global Forum 2020 Conference*. Cary, NC: SAS Institute Inc.
- Bales, Elizabeth, and Wei Zheng. 2017. "SAS® and UTF-8: Ultimately the Finest." *Proceedings of the SAS Global Forum 2017 Conference*. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

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

Mickaël Bouedo
SAS Software Globalization R&D
+1 919-531-4835
mickael.bouedo@sas.com
www.sas.com