Paper SAS296-2017

SAS® and UTF-8: Ultimately the Finest. Your Data and Applications Will Thank You!

Elizabeth Bales and Wei Zheng, SAS Institute Inc.

ABSTRACT

SAS® with Unicode UTF-8 encoding is ready to help you tackle the challenges of dealing with data in multiple languages. In today's global economy, software needs are changing. Companies are globalizing and consolidating systems from various parts of the world. Software must be ready to handle data from social media, international web pages, and databases that have characters in many different languages. SAS makes migrating your data to Unicode a snap! This paper helps you move smoothly from SAS using other encodings to the powerful SAS Unicode environment with UTF-8 support. Along the way, you will uncover secrets to successfully manipulate your characters, so that all of your data remains intact.

INTRODUCTION

Data today is no longer simple. The sheer volume of data that most organizations must handle has increased exponentially. In addition to the additional amount of data that systems must manage, many of those same software applications must now be prepared to handle the characters of several languages, often in the same process. For example, if your company has been operating in the United States and Canada, you already support both English and French. If the company expands into China, your data now needs to include Chinese characters.

Many character encodings that are supported by SAS represent characters for a specific language or region. When the SAS session encoding, the encoding specified by the ENCODING system option, is one of those encodings, the variety of characters available in your data is limited. For example, WLATIN1 is a single-byte (SBCS) encoding that includes characters used by the English and French languages. WLATIN1 does not include any Chinese characters at all. Therefore, in the scenario where English, French, and Chinese data must be represented, the WLATIN1 encoding is not adequate to handle all of your data needs.

SAS with Unicode UTF-8 encoding is the answer! UTF-8 includes all of the characters available in modern software today. This paper will help you understand how to migrate your SAS programs, data, and environment from other character encodings to UTF-8.

Note: The SAS UTF-8 session is only supported on UNIX and Windows operating systems. You cannot run SAS on z/OS with the ENCODING system option set to UTF-8.

DEFINING UTF-8

Before explaining how to use SAS with a UTF-8 session encoding, it is helpful to introduce you to Unicode and UTF-8. Unicode is the universal character encoding standard that includes the characters in most of the world's writing systems. Unicode also defines the rules for mapping each of those characters to a numeric value.

UTF-8 is a multibyte encoding that represents all of the characters available in Unicode. UTF-8 is backward compatible with ASCII characters, which include the letters of the English alphabet, digits, and symbols frequently used in punctuation or SAS syntax. The 128 characters that make up the ASCII character set are each represented as one byte in UTF-8. Therefore, when the ASCII characters in your data are converted to UTF-8, the size of those characters does not change.

All of the other characters available in UTF-8 require 2, 3, or 4 bytes in memory. This includes many characters that are represented with a single byte of memory in the SBCS character encodings. For more information about the encodings that are supported by SAS, see the section "Encoding for NLS" in the *SAS® 9.4 National Language Support (NLS): Reference Guide*.

Many characters that can be represented by one byte in a single-byte (SBCS) encoding often must be represented by 2 or more bytes in UTF-8. For example, the Euro symbol, €, is represented by one byte in Windows code page 1252, also

referred to as WLATIN1 in SAS. In UTF-8, the Euro symbol must be represented by 3 bytes. Table 1 compares the byte values that are used to represent the Euro symbol in WLATIN1 and UTF-8.

Euro symbol	WLATIN1 representation	UTF-8 representation
€	80	E282AC

Table 1. Hexadecimal Representation of the Euro Symbol

"Table 4. UTF-8 Character Length by Language" shows the number of bytes that are required to represent most of the characters that are usually associated with a particular language.

HANDLING UTF-8 STRINGS IN SAS

SAS provides many string functions and CALL routines that let you manipulate the characters in your data. The traditional set of SAS string functions, such as SUBSTR, INDEX, and LENGTH, are byte based, which means that these functions assume that one character is always represented by one byte in memory. These SAS string functions work very well on your data if the SAS session encoding is a single-byte (SBCS) encoding. (The SAS session encoding system option is specified during SAS configuration by the ENCODING system option.) These functions also work fine in a SAS UTF-8 session if ***all*** of the characters in the data are ASCII characters. However, if your data contains any multibyte characters, the results from the traditional SAS string functions, such as SUBSTR, INDEX, and LENGTH, might be incorrect.

Note: The BYTE function only supports data in single-byte (SBCS) character sets. This function is not appropriate for use with UTF-8 data.

SAS provides a set of string functions that are character based in order to help you manage strings containing multibyte data. The K functions never make an assumption about the size of a character, so they handle all of your UTF-8 characters correctly. This section demonstrates the types of problems that the traditional SAS string functions can cause, and then shows how to use the K function in your SAS programs to solve those problems.

The first example uses the traditional string functions LENGTH and SUBSTR. The SAS code below assigns a string that contains a Euro character (€) to a character variable:

```
data test ;
   str= "€123" ;
   s=substr(str,1,1) ;
   sl=length(s);
   l=length(str) ;
   put str= $hex16. /s= sl= / s= $hex. /l=;
run ;
```

The string "€123" is assigned to the variable STR. If the SAS code runs in SAS with a WLATIN1 session encoding, the characters will each be represented as one byte. However, if the same SAS code is run in UTF-8, the Euro character will require 3 bytes. Table 2 shows the hexadecimal representation for these characters in WLATIN1 and UTF-8.

Character	€	1	2	3
WLATIN1	80	31	32	33
UTF-8	E282AC	31	32	33

Table 2. Hexadecimal Representation for WLATIN1 and UTF-8

Output 1. String Function Results in a WLATIN1 SAS shows the results you will see in the SAS log when you run the DATA step code in a WLATIN1 session.

```
str=80313233
s=€ sl=1
s=80202020
l=4
```

Output 1. String Function Results in a WLATIN1 SAS Session

The SUBSTR function successfully copies the Euro character from the variable STR to the variable S. The LENGTH function correctly reports the number of bytes in STR is 4 and number of bytes in S is 1. Since each character in WLATIN1 is 1 byte, the lengths are accurate. The output of the $HEX. format for STR shows that the hexadecimal value represents the string "€123" in the WLATIN1 encoding. The $HEX. format also displays that the WLATIN1 value of the character '€' was correctly stored in S.

If you submit the same program to SAS with a UTF-8 session encoding, the results will look very different. Output 2 shows the results you will see in the SAS log.

```
str=E282AC313233
s=□ sl=1
s=E22020202020
l=6
```

Output 2. String Function Results in a UTF-8 SAS Session

If you look closely at the value of S, you can see that it is not correct. The SUBSTR function is instructed to start at column 1 and copy 1 byte into S. But the Euro symbol requires 3 bytes in UTF-8 rather than 1 byte as it did in the WLATIN1 encoding. The SUBSTR function only copied the first byte of the Euro symbol, the 'E2'x, to the variable S. The byte value 'E2'x does not represent a valid UTF-8 character.

The results of the LENGTH function are also very different. When LENGTH is called on the variable STR, the value assigned to L is 6 even though there are only 4 characters in STR. That's because the Euro symbol requires 3 bytes in UTF-8 and the other 3 characters in the string each require 1 byte. The length of S is still 1 byte because that was the only data SUBSTR was instructed to copy.

In order to get the expected results from these string manipulations in the UTF-8 session, the standard SAS string functions, SUBSTR and LENGTH, need to be replaced with KSUBSTR and KLENGTH. Output 3Output 1 shows the results you will see in the log of a SAS UTF-8 session after this change is made.

```
58          data test ;
59             str= "€123" ;
60             s=ksubstr(str,1,1) ;
61             sl=klength(s);
62             l=klength(str) ;
63             put str= $hex16. /s= sl= / s= $hex. /l=;
64          run ;

str=E282AC313233
s=€ sl=1
s=E282AC202020
l=4
```

Output 3. Results Using K Functions

In the example above, the KSUBSTR function is instructed to start at the first *character* position in STR and copy the entire *character* to the variable S. Therefore, KSUBSTR copies all of the bytes for the Euro symbol into S. KLENGTH correctly reports that S has 1 character and STR has 4 characters. Finally, the PUT function shows that the hexadecimal value for the string in STR correctly represents the string "€123" in UTF-8 and that the 3 bytes that represent the '€' character have been copied to S.

WHEN TO USE K FUNCTIONS

In some cases, you do not need to replace the traditional SAS string function even when a K function is available. The UPCASE function works correctly for all of your character data, even multibyte characters.

Note that you are not required to use the equivalent K function if you are *positive* that your data only contains ASCII characters. For example, the values in a column that contain product IDs might simply contain a combination of letters and numbers typically used in the English language. In that case, K functions are not required in order to successfully manipulate the data.

The *SAS® 9.4 National Language Support (NLS): Reference Guide* can help you determine when a K function is needed. The section "Internationalization Compatibility for SAS String Functions" in the "Functions for NLS" chapter of the guide contains a list of all string functions along with a compatibility setting for each function. Functions that use byte semantics only support single-byte characters. Those functions and call routines are assigned an I18N compatibility level of 0 and should not be used. Functions that are safe to use with multibyte data have an I18N compatibility of 2.

As a best practice, you should know your data when you are making decisions about which SAS functions to use.

USING SAS FORMATS

Like the traditional string functions, the length specified to SAS formats is also byte based. That is, when you assign a SAS format to a character variable or string, the length is not equal to the number of characters you want to see. Instead, SAS interprets the format length as the number of bytes that the format will use when it displays the string.

In order to display all of the characters in a UTF-8 string, the length of the format must be big enough to display all of the data you want to see. As a best practice, you need to know what type of characters might be present in the data.

If your UTF-8 data only contains ASCII characters, you can assume that the size of the characters in that data is one byte. In that case, the format length for the string is the same as the number of characters in the data. However, a multibyte character in UTF-8 must be represented by the correct number of bytes in the format length. Therefore, if you want to see all of the characters in the string, the format length must be the same as the number of bytes required to display the whole string. Otherwise, some of the data will be truncated when the string is displayed.

In the code below, the FORMAT statement specified in the PROC PRINT step specifies a format of $4. for the variable STR. The length of the format is 4 bytes.

```
proc print data=test;
   format str $4.;
run;
```

If you run this code in SAS with a WLATIN1 session encoding, you will see the result shown in Output 4:

Obs	str
1	€123

Output 4. Result in WLATIN1 Using the $4. Format

Since all of the characters WLATIN1 are 1 byte, the length of the format applied to STR is enough to display all of the characters. However, Output 5 shows the result of using the $4. format to display the value of STR in UTF-8:

Obs	str
1	€1

Output 5. Result in UTF-8 Using the $4. Format

Notice that only 4 bytes of the value of STR are displayed. The rest of the data is truncated because the value “€123” is 6 bytes long. In order to display all of the characters in STR, a format length of 6 must be used, as below:

```
format str $6.;
```

Output 6 shows the result from PROC PRINT after the format width is changed to 6.

Obs	str
1	€123

Output 6. Result in UTF-8 Using the $6. Format

If you are migrating your SAS libraries from another SAS encoding to UTF-8, you might need to increase the size of the format length in order to display all of the characters you want to see. If the original encoding supports Western European characters, it is often safe to assume that the size of your data will increase by an additional 10%. But Thai characters that are represented by 1 byte in the ISO 8859-11 or THAI encoding in SAS require 3 bytes in UTF-8. You must usually multiply by a factor of 3 to display all of the data.

The “APPENDIX” section of this document provides data tables to help you determine the additional storage requirements when saving your data as UTF-8.

WORKING WITH SAS DATA SETS

Data sets created in SAS®9 have an encoding attribute stored in the data set header. By default, that encoding attribute and the encoding of the character data stored in the data set match the SAS session encoding that is in effect when the data set is created. As we will see later, it is possible to override the encoding of the data set.

Note: The data set encoding must match the encoding of the character data in the file. Otherwise, data could be lost during transcoding or unnecessary transcoding could take place. See the “TROUBLESHOOTING” section of this guide for help with specific problems.

READING SAS DATA SETS

When SAS reads a data set that was created in another SAS®9 session, SAS compares the current session encoding with the data set encoding to see if there is a match. If the two encodings do not match, SAS invokes Cross-Environment Data Access (CEDA) to transcode the characters in the data from the original encoding to the current session encoding. It is possible to disable CEDA transcoding. The section “Disabling CEDA Transcoding” provides hints for when and how to do this.

When a SAS UTF-8 session reads a data set that was created in SAS with a different encoding, you might see a message from CEDA telling you that the data was lost during transcoding. Since the UTF-8 encoding supports all of the characters that are available in other SAS data set encodings, SAS with a UTF-8 session encoding should be able to successfully transcode all characters from a SAS data set that is not encoded using UTF-8. Therefore, when you see this type of message in your SAS UTF-8 session, it usually indicates that data in one or more character columns was truncated when CEDA attempted to transcode the characters to UTF-8. Truncation occurs in this situation because the characters require more bytes when they are represented as UTF-8 than they required in the original encoding. See the section **DEFINING UTF-8** for an example that explains the space requirements in UTF-8.

For example, when the DATA step code below is run in SAS with a WLATIN1 session encoding, the SAS data set will have WLATIN1 in the data set header:

```
libname mylib 'mylib';
data mylib.test;
   str= "€123" ;
run ;
```

The PROC CONTENTS code below displays the attributes of the data set including the encoding:

```
proc contents data=mylib.test;
run;
```

Output 7 shows a portion of the output from PROC CONTENTS that displays the encoding of MYLIB.TEST:

The CONTENTS Procedure

Data Set Name	MYLIB.TEST	**Observations**	1
Member Type	DATA	**Variables**	1
Engine	V9	**Indexes**	0
Created	01/04/2017 14:31:40	**Observation Length**	4
Last Modified	01/04/2017 14:31:40	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Output 7. Output from PROC CONTENTS Showing the ENCODING Attribute

The variable STR in MYLIB.TEST contains a Euro character, which is represented by 1 byte in the WLATIN1 encoding, so SAS assigns a length of 4 to STR. Output 8 shows a different section of the output from PROC CONTENTS listing the attributes of the variables in MYLIB.TEST.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	str	Char	4

Output 8. Output from PROC CONTENTS Showing Variable Attributes

When SAS reads MYLIB.TEST in a SAS UTF-8 session, you will see a message in the SAS log that some data was lost during transcoding. If SAS simply opens the data set to display the data, the message you see will be a warning. However, if SAS attempts to save the data to a new file, you will see an error. For example, the DATA step code below is run in a SAS UTF-8 session, and it should create a new data set, WORK.MYTEST:

```
libname mylib 'path to mylib';
data mytest;
   set mylib.test;
run;
```

But when the code is run in a SAS UTF-8 session, the DATA step will fail during CEDA transcoding. Output 9 shows the error that CEDA writes to the SAS log when truncation occurs.

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

Output 9. Transcoding Error Displayed by CEDA

Here's what is going on behind the scenes. When SAS tries to save the data, the variable STR and its attributes are copied from MYLIB.TEST. The length of STR in MYLIB.TEST is 4 bytes. Table 2 in the section "HANDLING UTF-8 STRINGS IN SAS" shows that the Euro character is represented by 3 bytes in UTF-8. When the string in STR is transcoded to UTF-8, it requires 6 bytes. Therefore, some of the data from STR is truncated when CEDA attempts to transcode.

Using CVP to Prevent Truncation

The CVP engine can be used to prevent truncation errors when CEDA reads SAS data sets. CVP, or character variable padding, is a read-only engine for SAS files that does just what the name implies: pads character variables with more bytes to make them bigger.

Note: If the TRANSCODE attribute for a variable is set to NO, CVP will not change the size of that variable.

To use CVP, you can specify the CVP keyword in the LIBNAME statement as follows:

```
libname mylib cvp 'path to SAS files';
```

By default, CVP uses an expansion of 1.5 times the variable length. This is usually sufficient to support the data for Western European languages because the text for those languages usually has a high ratio of ASCII characters to national characters. 1.5 might also be sufficient as a multiplier for most Asian language data. Even though many Asian characters require 3 bytes in UTF-8, the double-byte (DBCS) encodings that support those languages already require 2 bytes for each Asian character.

However, many languages that are represented by single-byte (SBCS) encodings use a high ratio of national characters in the text, which require 2 or 3 bytes in UTF-8. For example, Russian text primarily uses characters from the Cyrillic alphabet, which require 1 byte in the SBCS encoding Windows cp1250, or WCYRILLIC. Since each Cyrillic character requires 2 bytes in UTF-8, an expansion rate of 1.5 might not provide enough space to avoid truncation, depending on the length of the variable.

If you need to change the expansion rate, you can specify either CVPMULTIPLIER or CVPBYTES in the LIBNAME statement. CVPMULTIPLIER specifies a multiplier value that is applied to the length of the variable to expand the size. CVPBYTES adds a specific number of bytes to the length of character columns.

```
libname mylib 'path to SAS files' cvpmultiplier=2.5;
libname mylib 'path to SAS files' cvpbytes=5;
```

The “APPENDIX” section of this document provides data tables to help you determine the additional storage requirements when saving your data as UTF-8. See “Table 3. Saving Data as UTF-8: Possible Storage Size Increase” and “Table 4. UTF-8 Character Length by Language” for hints on expanding the size of your character columns based on the language or encoding of your data.

Disabling CEDA Transcoding

Transcoding can be costly, depending on the size of your data set. If the data set encoding is not compatible with the SAS session encoding, CEDA will attempt to transcode the characters to the session encoding. Transcoding can be disabled if it is not necessary. This section provides some guidelines, tips, and cautions so that you can make a wise decision about whether transcoding to UTF-8 can be disabled.

Follow these steps with caution. When used correctly, you could see significant performance improvements if you prevent CEDA from transcoding your data if your data sets are very large. But misuse of these suggestions could result in data loss or corruption.

ASCIIANY

UTF-8 is 100% compatible with the 128 encoded characters of the 7-bit ASCII encoding. When SAS reads a data set that ***only*** contains ASCII characters, it is not necessary for SAS to transcode this data. You can prevent CEDA from transcoding a data set by specifying the special encoding, ASCIIANY, even if the data set was created with a different encoding.

To override the encoding of a single data set, specify ENCODING=ASCIIANY. Or, if all of the data sets in the SAS library are limited to ASCII characters, specify ASCIIANY as the value of the INENCODING option in the LIBNAME statement. This SAS code demonstrates how to use the LIBNAME statement with the INENCODING=ASCIIANY option:

```
libname mylib ‘path to SAS files’ inencoding=asciiany;
```

TRANSCODE Variable Attribute

Your data sets might have a mix of character variables, with data in some variables that should transcode to UTF-8, while others in the same data set should be ignored by CEDA transcoding. If this describes your situation, you can run SAS using the original session encoding and specify the option TRANSCODE=NO in the ATTRIB statement for the appropriate variables.

Saving a Permanent Copy of the Data Set

While the CVP engine does expand the size of character variables, it is a read-only engine. The expanded character columns and transcoded UTF-8 data are only available in memory and only when the data is referenced using the libref associated with CVP. The physical copy of the data set is not updated. In that scenario CVP must be used each time a SAS UTF-8 session reads the data set that is not encoded as UTF-8. Also keep in mind that CEDA must also be invoked in order to transcode that character data to UTF-8.

If SAS must read those non-UTF-8 data sets often, consider creating a permanent copy of the data set encoded as UTF-8. It is possible to accomplish this using PROC COPY, PROC DATASETS, or a DATA step. *As a best practice, consider storing SAS files with different character encodings in separate SAS libraries.* The SAS statements below show how to use the CVP engine and PROC DATASETS with a COPY statement to copy data sets in another to a new SAS library:

```
libname oldlib cvp "path_to_SAS_files";
libname utflib "path_to_UTF-8_library";
proc datasets nolist;
   copy in=oldlib out=utflib override=(encoding=session outrep=session) memtype=data;
run; quit;
```

The OVERRIDE option in the example specifies the ENCODING and OUTREP options. These options prevent SAS from copying the original data set encoding and host representation when creating the new copy of the data set. Otherwise, the attributes from the old data set, including the original data set encoding, will be written to the new data set. The characters that are written to the new data set will also be encoded using the original encoding and CEDA will be required by the SAS UTF-8 session to read the new data set and transcode the data.

If you want to save your new data set as UTF-8 and prefer to replace all of the data set attributes, you can specify the NOCLONE option in the PROC COPY statement. The SAS statements below demonstrate how to run PROC COPY with the NOCLONE option:

```
proc copy in=oldlib out=utflib noclone index=no constraint=no;
run;
```

You can also use PROC MIGRATE to make a UTF-8 copy of your data. Note that PROC MIGRATE does not work with the CVP engine. If additional padding is required for the character variables, you must create the UTF-8 copy of your data set using one of the other methods described above.

More information is available about PROC COPY and PROC DATASETS in the *Base SAS® 9.4 Procedures Guide*.

Expanding the Format Length

While the CVP engine will expand the length of your character variables, it does not increase the length of the formats that are applied to those variables. Be aware that the format length represents the number of bytes that that format will display, not the number of characters.

If the character variables require more bytes in UTF-8 but the format length is not increased for those variables, the formatted strings might be truncated in your output. If you are not satisfied with the results that the format displays, you should increase the format length to the number of bytes needed to display all of the data you want to see.

A SAS macro called *utf8_estimate* is included in the paper “Processing Multilingual Data with the SAS® 9.2 Unicode Server”. *Utf8_estimate* estimates the length needed for character variables and formats when you migrate data sets to UTF-8.

SAS Viya 3.2 includes a new option for the CVP engine that can help. If you specify CVPFORMATWIDTH=YES on your LIBNAME statement, the CVP engine will increase the length of the formats that are applied to character variables. The length is increased according to the default multiplier or the value specified by CVPMULTIPLIER or CVPBYTES.

Note: The CVPFORMATWIDTH does not impact the length of user-defined formats.

WRITING SAS DATA SETS

When SAS 9 creates a new data set, the characters written to that data set are encoded using the SAS session encoding. The session encoding is also saved as the data set encoding. For SAS with a UTF-8 session encoding, that data set encoding is UTF-8. If you want to create your data sets using a different encoding, you must specify the encoding using the ENCODING data set option or the OUTENCODING option in the LIBNAME statement.

Caution: If the UTF-8 data contains characters that are not supported in the specified encoding, you will see a transcoding error in the SAS log. Use the KPROPDATA function to remove or convert any problematic characters. See the section “HOW TO USE KPROPDATA” for more information about using this function.

The ENCODING data set option specifies the encoding for a single data set. The code sample below shows how to specify an alternate encoding using the ENCODING option in the DATA statement:

```
data mydata (encoding=wlatin2);
   .
   .
run;
```

If you want to set an alternate encoding for SAS data sets created in a SAS library, use the OUTENCODING option in the LIBNAME statement to specify the output encoding to use when creating files in that SAS library. The code below specifies the LATIN1 encoding using the OUTENCODING option in the LIBNAME statement:

```
libname mylib 'path to SAS files' outencoding=latin1;
```

```
data mylib.mydata;
      .
      .
run;
```

When you specify an encoding using either of these methods, SAS also transcodes the characters to the new encoding as it writes them to the data set. In addition, if you replace an existing data set in the SAS library using a libref with OUTENCODING specified, the data set encoding will also be changed to the new encoding. As a best practice, consider storing SAS files with different character encodings in separate SAS libraries.

If both the OUTENCODING LIBNAME statement option and the ENCODING data set option are specified, SAS uses the encoding from the data set option.

WORKING WITH OTHER TYPES OF SAS DATA

The section “READING SAS DATA SETS” provides hints about how to use the CVP engine to avoid truncation when CEDA transcodes a SAS data set. However, CEDA does not support transcoding for some SAS file types, such as SAS format catalogs. CEDA also does not support indexes if character variable padding is required and does not process integrity constraints. The “Cross-Environment Data Access (CEDA)” chapter in *Moving and Accessing SAS® 9.4 Files* contains a section that documents all of the CEDA limitations.

This section describes some techniques you can use to migrate user-defined formats and re-create indexes and integrity constraints in your SAS UTF-8 environment.

MIGRATING SAS FORMAT CATALOGS

Unlike SAS data sets, SAS catalogs, including format catalogs, do not store the encoding of the data. Also, CEDA does not support catalogs. Therefore, when a SAS session reads a catalog that was created using a different encoding the data in the catalogs is not transcoded.

If you created user-defined formats in another encoding, you will need to transfer those to the SAS UTF-8 session using an output control data set. An output control data set is used to transfer the format data from the original SAS session encoding to UTF-8. If you see the CEDA transcoding error, you can use the CVP engine to pad the size of the character columns. See the section “Using CVP to Prevent ” for additional information.

The code below demonstrates a two-step process for moving format data from the original SAS encoding to SAS with a UTF-8 session encoding. The first program searches a SAS library for all of the format catalogs then runs PROC FORMAT to create an output control data set for each format catalog it finds. The output control data sets are saved in a temporary library that will be read later by a SAS UTF-8 session. This program must be run using the SAS environment with the session encoding that was used to create the formats:

```
/*************************************************************************/
/* OLDLIB points to the existing SAS library that was created using the  */
/* original session encoding.                                            */
/* INTERIM points to a library used to store the control data sets and   */
/* other files. INTERIM should be empty before running this code.        */
/*************************************************************************/
libname oldlib "path to SAS library" access="readonly";
libname interim "path to interim library";

/*************************************************************************/
/* This PROC DATASETS step will remove all of the SAS files from the     */
/* INTERIM library to prevent problems with creating formats in the UTF-8*/
/* environment.                                                          */
/*************************************************************************/
proc datasets library=interim kill nolist;
run; quit;


/*************************************************************************/
/* Search through the OLDLIB library for format catalogs. Save the formats*/
/* to output control data sets and saves them in the INTERIM library.    */
/*************************************************************************/
```

```
%macro getCatalogs();
   proc sql;
      create table catalogs as
      select distinct memname
      from dictionary.catalogs
      where libname='OLDLIB'
      ;
      %let tot = &sqlobs;
   quit;

   %if &tot %then %do;
      data _null_;
          set catalogs end=last;
          retain cnt 1;
          call symput('cat'||strip(cnt), strip(memname));
          if last then do;
             call symput('cnt', strip(cnt));
          end;
          cnt = cnt + 1;
      run;
      %do i=1 %to &cnt;
          proc format library=oldlib.&&cat&i cntlout=interim.&&cat&i;
          run;
      %end;
   %end;
%mend getCatalogs;

%getCatalogs();
```

The second program reads the output control data sets that were created in the previous program and runs PROC FORMAT to create the format catalogs in UTF-8. The UTF-8 format catalogs are stored in a different library than the original catalogs. The CVP engine is used to ensure that the control data sets have enough space when CEDA transcodes the strings to UTF-8. The code assumes that a multiplier of 1.5 will provide enough additional space. This code is run in your SAS UTF-8 session:

```
/*************************************************************************/
/* INTERIM is the SAS library where the control data sets are located.   */
/* U8LIB is the SAS library where the UTF-8 version of the format catalogs  */
/* is created. The CVP engine will pad the character variables in the    */
/* output control data set so they are big enough to hold all of the data.  */
/* If a transcoding error occurs, specify CVPMULTIPLIER and increase the    */
/* sizes of the multiplier.                                              */
/*************************************************************************/
libname interim cvp "path to interim library";
libname u8lib "path to SAS library";


/*************************************************************************/
/* Read the output control data sets from the INTERIM library formats.   */
/* Search through the OLDLIB library for format catalogs. Save the formats  */
/* to output control data sets and saves them in the INTERIM library.    */
/*************************************************************************/
%macro getFormats();
      proc datasets library=interim nolist;
         contents data=_ALL_ out=work.datasets noprint;
      run; quit;
      proc sql;
         create table formats as
         select distinct memname
         from work.datasets
      ;
```

```
      %let tot = &sqlobs;
      quit;
      %if &tot %then %do;
         data _null_;
            set formats;
            retain cnt 1;
            call symput('dat'||strip(cnt), strip(memname));
            cnt = cnt + 1;
         run;
         %do i=1 %to &tot;
            proc format library=u8lib.&&dat&i cntlin=interim.&&dat&i;
            run;
         %end;
      %end;
%mend getFormats;

%getFormats();
```

MIGRATING INDEXES AND INTEGRITY CONSTRAINTS

CEDA does not support integrity constraints and will not recreate them in your SAS UTF-8 environment. CEDA will support recreation of indexes as long as you use PROC MIGRATE when you create the UTF-8 copy of the data. However, PROC MIGRATE does not work with CVP. If you need to use the CVP engine to expand the size of your character columns, you must use a different method to recreate your indexes.

If you want to preserve indexes and integrity constraints defined for your data sets, you can save them to an output data set with PROC DATASETS and apply them to a UTF-8 copy of the data sets in the SAS UTF-8 session. Then you need to run your SAS UTF-8 session to re-create the indexes and integrity constraints for the UTF-8 version of your SAS data sets.

The code below demonstrates a two-step process, similar to the process used by the code samples in "**MIGRATING SAS FORMAT CATALOGS**." The first program runs PROC DATASETS to create an output data set. The OUT2 option is used in the CONTENTS statement to save information about the indexes and integrity constraints for all data sets in the specified library. This program must be run using the SAS environment with the session encoding that was used to create the data set associated with the indexes and constraints:

```
/*************************************************************************/
/* OLDLIB points to the existing SAS library that was created using the     */
/* original session encoding.                                               */
/* INTERIM points to a library used to store the data for indexes and       */
/* integrity constraints. INTERIM should be empty before running this code. */
/*************************************************************************/
libname oldlib "path to SAS library" access="readonly";
libname interim "path to interim library";

/*************************************************************************/
/* WARNING!!!!                                                              */
/* This PROC DATASETS step will remove all of the SAS files from the        */
/* INTERIM library to prevent problems with creating the new data in the    */
/* SAS UTF-8 environment.                                                   */
/*************************************************************************/
proc datasets library=interim kill nolist;
run; quit;

/*************************************************************************/
/* Create the constraints dataset in the INTERIM library                    */
/*************************************************************************/
proc datasets lib=oldlib nolist;
   contents data=_ALL_ out2=interim.constraints noprint;
run; quit;
```

The second program runs PROC COPY to create a UTF-8 copy of each data set. The NOCLONE option is used in the PROC COPY statement so that the attributes of the SAS UTF-8 session will be used when the new data sets are created. Note that the NOCLONE option prevents PROC COPY from copying any attributes from the original data set. If you want to retain attributes such as BUFSIZE and COMPRESS, use the PROC DATASETS COPY statement with the OVERRIDE option to override ENCODING and the output data representation (OUTREP=). The sample code in **Saving a Permanent Copy of the Data Set** shows an example of how to use PROC DATASETS with the OVERRIDE option. See the documentation for PROC COPY and PROC DATASETS in the *Base SAS® 9.4 Procedures Guide* for more information on the NOCLONE and OVERRIDE options.

The CVP engine is used to pad the size of the character variables for the SAS data sets and for the output data sets created in the previous step. If a transcoding error occurs when you run the program, you will need to increase the multiplier used by the CVP engine. CVP uses a default multiplier value of 1.5. The section **Using CVP to Prevent Truncation** explains how to increase the amount of space that is added to the character columns of your data set.

Next, the %getConstraints macro is called to gather the data about indexes and constraints and create them on the UTF-8 copies of the data sets. This code is run in SAS with a UTF-8 session encoding:

```
/*************************************************************************/
/* OLDLIB points to the existing SAS library that was created using the     */
/* legacy session encoding.                                                 */
/* INTERIM is the SAS library where the data for the indexes and            */
/* is stored. The CVP engine will pad the character variables in the        */
/* data sets so they are big enough to hold all of the data in UTF-8.       */
/* If a transcoding error occurs, specify CVPMULTIPLIER and increase the    */
/* sizes of the multiplier.                                                 */
/* U8LIB is the SAS library where the UTF-8 copies of the data sets are     */
/* created.                                                                 */
/*************************************************************************/
libname oldlib cvp  "path to original SAS library";
libname interim cvp "path to interim library";
libname u8lib "path to SAS library";

/*************************************************************************/
/* Run PROC COPY to create permanent copies of the data sets in UTF-8.      */
/* The NOCLONE option is specified so SAS will apply the system option      */
/* settings of the UTF-8 session.                                           */
/*************************************************************************/
proc copy in=oldlib out=u8lib noclone index=no constraint=no;
run;

/*************************************************************************/
/* Create the indexes and integrity constraints on the new UTF-8 data sets. */
/*************************************************************************/
%macro getConstraints();
      proc datasets library=interim nolist;
         contents data=_ALL_ out=work.datasets noprint;
      run; quit;
      proc sql;
         create table work.constraints as
         select distinct memname
         from work.datasets
         where upcase(memname) eq 'CONSTRAINTS'
      ;
      %let tot = &sqlobs;
      quit;
      %if &tot %then %do;
         data _null_;
            set interim.constraints end=last;
            retain cnt 0;
            if (member ne "" and recreate ne "") then do;
               cnt = cnt+1;
               call symput('dat'||strip(cnt), strip(Member));
               call symput('con'||strip(cnt), strip(Recreate));
```

```
            end;
            if last then call symput('cnt', strip(cnt));
         run;
         %do i=1 %to &cnt;
            proc datasets library=u8lib nolist;
               modify &&dat&i;
                      &&con&i;
            run; quit;
         %end;
      %end;
%mend getConstraints;
%getConstraints();
```

WORKING WITH RAW DATA

A single SAS session can read raw data from many sources. If no encoding is specified, SAS assumes that the encoding of the character data matches the SAS session encoding. SAS can also read data from a file with a different encoding as long as the data encoding is identified and the data encoding is compatible with the session encoding. SAS with a UTF-8 session encoding supports all of the characters that other SAS encodings support. Therefore, your SAS UTF-8 session should be able to transcode all of the characters from an external file to UTF-8 as long as the data is valid.

Use the ENCODING statement option to specify the file encoding. ENCODING is used on the SAS statement that references the external file. The code below shows an example of using the ENCODING option in the FILENAME statement to specify the LATIN1 encoding:

```
filename extfile 'external-file' encoding='latin1';
```

LATIN1 is the SAS defined encoding name that refers to the ISO 8859-1 encoding. When a SAS UTF-8 session reads the data referred to by the EXTFILE fileref, the character data is transcoded from LATIN1 to UTF-8.

In addition to the FILENAME statement, the ENCODING statement option is supported as an option on the DATA, INFILE, and FILE statements as well as several ODS statements, and %INCLUDE. You can also specify the ENCODING statement option when you want to use a different encoding to write data to an external data source. Documentation for the ENCODING= statement option is available in the *SAS® 9.4 National Language Support (NLS): Reference Guide* in the chapter titled "Options for Commands, Statements, and Procedures for NLS."

SAS can also write data to a file in a different encoding. The same ENCODING statement option is used to identify the encoding of the file.

Caution: If your UTF-8 data contains characters that are not supported by the specified encoding, SAS will transcode the unsupported characters to a substitution character. *No error will appear in the SAS log to warn you of the data loss.* The KPROPDATA function can be used to remove or convert any problematic characters from the data. See the section "HOW TO USE KPROPDATA" for more information.

BYTE ORDER MARK (BOM)

Many applications add a byte-order mark (BOM) when writing data as UTF-8 or any other format of Unicode to an output file. If a BOM is present at the beginning of a file, SAS uses the BOM to determine the encoding of the data and the ENCODING statement is not needed.

If the BOMFILE system option is enabled and ENCODING is not specified, the SAS UTF-8 session will write the BOM to any external files that it creates.

The BOM for UTF-8 is represented by the hexadecimal bytes "EFBBBF".

WORKING WITH RELATIONAL DBMS TABLES

SAS/ACCESS® software enables SAS to read and write data to a relational database management systems (DBMS) in the same way you would read and write to a SAS library. If the encodings of the client and server are different, transcoding is required so that the characters are represented correctly.

Transcoding between the database and client encodings is usually determined by the operating system locale where the client is installed. However, some of the DBMS clients do not depend on the operating system and can be configured differently. Therefore, it is critical for the DBMS server to know the client's encoding.

In a SAS UTF-8 session you need to be sure the DBMS client is configured using the UTF-8 encoding. This paper explains the steps necessary for a SAS UTF-8 session to communicate with Oracle and Hadoop. You can find additional information for accessing data in these and other DBMS clients in the technical papers "Multilingual Computing in SAS® 9.4" and "Processing Multilingual Data with the SAS® 9.2 Unicode Server".

SAS/ACCESS® INTERFACE TO ORACLE

The NLS_LANG parameter setting lets the Oracle server know what locale you are using on the client. The format of the NLS_LANG value includes the language, region, and encoding that match your SAS client. If the Oracle server encoding is UTF-8, then there is no transcoding between Oracle and SAS UTF-8 session.

On Windows, NLS_LANG is defined as a Windows registry key. The following command shows how to set NLS_LANG for use with your SAS UTF-8 session:

```
NLS_LANG=American_America.AL32UTF8
```

On UNIX hosts, NLS_LANG is defined as a UNIX environment variable. The command below can be added to the sasenv_locale file located in the !SASROOT/bin directory of your SAS® Foundation deployment:

```
export NLS_LANG=American_America.AL32UTF8
```

SAS/ACCESS® INTERFACE TO HADOOP

Accessing data from Hadoop has become very popular. Fortunately, communication between your SAS UTF-8 session and Hadoop is simple. By default, the JRE option, file.encoding, should be set to UTF8 when your SAS session encoding is UTF-8. If the value of file.encoding has been changed, it should be set to UTF8. To verify that file.encoding is set correctly, run PROC JAVAINFO in SAS:

```
proc javainfo all;
run;
```

CONVERTING AN EXISTING DEPLOYMENT TO UTF-8

When SAS is installed on a Windows or UNIX operating system, at least three NLS configuration files are created during the installation process. One of the NLS configurations runs the SAS UTF-8 session. If your default SAS environment uses a different session encoding, you can use the UTF-8 configuration file to invoke SAS with UTF-8.

The UTF-8 configuration file is located under the !SASROOT/nls directory in your SAS Foundation installation. On Windows, you can set the -config option on the SAS command line in the command prompt or create a shortcut for the SAS command and set the option there.

On UNIX, the sas_u8 invocation script invokes the SAS UTF-8 session. The sas_u8 script is located in the !SASROOT/bin directory.

REQUIREMENTS FOR CHANGING THE SAS® INTELLIGENCE PLATFORM

If you have already installed the SAS Intelligence Platform, the SAS servers for that application server context are configured using the SAS session encoding that was in effect at configuration time. In order for your SAS applications to communicate successfully with SAS servers that are running UTF-8, you must create a new SAS Application Server context that is configured for UTF-8. Otherwise, you might see transcoding errors when you attempt to connect your SAS applications with the UTF-8 servers. See the *SAS® 9.4 Intelligence Platform: Installation and Configuration Guide* for instructions on how to install and configure additional server contexts for your SAS deployment.

TROUBLESHOOTING

The UTF-8 encoding supports all characters that are available in any encoding that SAS supports. Therefore, if you have followed the guidelines above, SAS with a UTF-8 session encoding should be capable of reading characters from any data source. Occasionally you might still see bad data or errors in your SAS UTF-8 environment. This section describes some common problems that can occur and provides suggestions for potential solutions.

BAD DATA AFTER READING A DATA SET

After SAS reads your data set into the UTF-8 session, you should be able to see the characters that you are expecting as long as no transcoding problems occurred. If all of the characters in a column appear to be correct while the last character is unreadable, the problem might be that the format length is not big enough. The section “Expanding the Format ” provides details about why this happens and how you can fix the problem.

If the ASCII characters look correct but all of the other characters for a variable are garbled, check the TRANSCODE attribute for that variable. If TRANSCODE is set to NO, CEDA ignores that variable when it transcodes the rest of the data set.

You can use the VTRANSCODE, VTRANSCODEX, or VARTRANSCODE to get the setting of the TRANSCODE attribute for variables in your data. Each function returns 0 if the variable does not transcode and 1 if it does.

The *SAS® 9.4 National Language Support (NLS): Reference Guide* documents each function and shows sample SAS code that uses the variable.

BAD DATA AFTER READING A FILE

You might also notice bad data after reading a file that was not created by SAS. If the encoding of that file is not UTF-8, be sure you set the ENCODING option on the FILENAME or other statement that references the file so that SAS will transcode the data when reading the file. If ENCODING was set, be sure the specified the SAS encoding name that represents the actual encoding of the data. You might need to follow up with the person who created the file to get that information.

The *SAS® 9.4 National Language Support (NLS): Reference Guide* lists the SAS encoding names that are valid to use on SAS statements.

HOW TO USE KPROPDATA

If you are writing UTF-8 data to a data source in a different encoding, SAS will attempt to transcode the characters to the destination encoding. Most characters require less space when they are represented in regional encodings. If the data is written to a SAS data set in another encoding, you should not need to worry about the size of character variables. If your UTF-8 data contains characters that are not supported in the destination encoding, those characters will be lost or transcoding will fail.

KPROPDATA can help you preserve or remove problematic characters that the destination encoding does not support. For example, the option BLANK replaces them with a blank space while the UESC option converts characters to a Unicode escape string. *Each Unicode escape string requires 6 characters in the resulting string.*

When KPROPDATA is run in the UTF-8 session, the input encoding is the same as the session encoding. You must specify the output encoding in order for KPROPDATA to transcode the characters correctly.

Note: When you specify an output encoding other than the session encoding, the encoding of the string that is returned by KPROPDATA is also the output encoding.

KPROPDATA is documented in the *SAS® 9.4 National Language Support (NLS): Reference Guide*.

TRANSPORT FILES IN ANOTHER ENCODING

When PROC CPORT creates a transport file, all data sets are added to the file using current SAS session encoding. However, starting in SAS 9.4, PROC CPORT preserves the data set encoding instead of using the session encoding if the following two conditions are met: the data set must be US-ASCII and SAS must be running on an ASCII host. In that case, the encoding of the data set that is written to the transport file remains US-ASCII.

When you use PROC CIMPORT to import data sets that are in a transport file, the data set encoding must be compatible with the SAS session encoding. So data sets imported into a SAS UTF-8 session must be encoded as UTF-8 or US-ASCII. Note that this restriction was removed in the third maintenance release of SAS 9.4.

Beginning in the third maintenance release of SAS 9.4, you can run PROC CIMPORT in your SAS UTF-8 session to import data sets that were created using any encoding. If the transport file does contain a data set that is not encoded as UTF-8 or US-ASCII, the SAS UTF-8 session will transcode the data to UTF-8. If the columns in the data set are not long enough to hold the data that was transcoded to UTF-8, a warning will be written to your SAS log stating that the

destination buffer size is not sufficient for the transcoded data. SAS will continue to read the data set. However, the character strings will be truncated to fit within the number of bytes available in the character column.

If you are unable to read data sets correctly from a transport file using PROC CIMPORT in your SAS UTF-8 session, you will need to find another way to access the data. You can accomplish this by running PROC CIMPORT in SAS with a session encoding that is compatible with the data sets in the transport file. Once the data sets are saved to a temporary SAS library, follow the advice provided in the section "READING SAS DATA SETS" to read the data sets in your SAS UTF-8 session. You may need to use the CVP engine to prevent truncation when the data is transcoded. If you then wish to store the data in a transport format, run PROC CPORT in a SAS UTF-8 session to save the UTF-8 data sets to a transport file.

PATHNAME ON UNIX HOSTS

As you move from other SAS session encodings to UTF-8 on your UNIX system, you might find that SAS does not recognize some of the pathnames that you used before. If the filename or pathname contains a non-ASCII character, SAS will send a UTF-8 encoded version of that name to the operating system when searching for the file or path. But if the filename or pathname was created by the operating system using the original encoding, SAS will not be able to locate the file.

To avoid the problem, define an environment variable called PATHENCODING in your UNIX shell. The value you assign to PATHENCODING should be the SAS encoding name that matches the UNIX file system encoding that was in place when you created the files.

For example, if a SAS UTF-8 session is running on the Linux operating system and pathnames were previously created using an encoding of "eucjp", you can create the PATHENCODING variable like this:

```
export PATHENCODING=EUC-JP
```

The *SAS® 9.4 Companion for UNIX Environments, Sixth Edition* documents the PATHENCODING environment variables and directs you to a list of the encoding values that are valid to use.

BEST PRACTICES

This paper has introduced many concepts that might be new to you. This section lists some of the best practices to keep in mind when you are working with UTF-8 as your SAS session encoding.

1. SAS files with different character encodings should be stored in separate SAS libraries.
2. Know your data. Transcoding problems often occur when SAS programmers are not familiar with the data in their data sets. You can avoid many problems by knowing what types of characters your SAS data sets and other files can contain.
3. Use the CVP engine to expand the size of character columns in your data sets.

RESTRICTIONS

The following restrictions apply to SAS with UTF-8 support:

1. You cannot run SAS with a UTF-8 session encoding on z/OS.
2. SAS windowing environment does not support multibyte UTF-8 characters correctly. You can use SAS® Studio or SAS® Enterprise Guide® as an interface to write and run your SAS programs. You can also run your SAS programs in batch mode.

APPENDIX

When your character data is saved as UTF-8, the memory required to represent those characters might need to increase. The tables here in the appendix offer some hints in calculating that expansion.

Table 3 shows the amount of overall increase you can expect as you move data from another encoding to UTF-8.

Source Encoding	Languages Supported	Storage Size Increase in UTF-8

ASCII	English, Malay, and other supported languages	0%
ISO-8859-1 (LATIN1) and Windows cp1252 (WLATIN1)	Western European	10%
ISO-8859-7, plain text	Greek	90%
ISO-8859-7, 50% markup	Greek	45%
TIS-620, plain text	Thai	190%
TIS-620, 50% markup	Thai	95%
EUC-KR, plain text	Korean	50%
EUC-KR, 50% markup	Korean	55%

Table 3. Saving Data as UTF-8: Possible Storage Size Increase

The character mapping for UTF-8 makes it easy to determine how many bytes you need for the characters in your language. Table 4. UTF-8 Table 4 shows how many bytes are needed in order to represent the characters for the languages in the list.

Character length	Language
1 byte	US_ASCII Characters
2 bytes	East and West European, Baltic, Greek, Turkish, Cyrillic, Hebrew, Arabic, and other supported character sets.
3 bytes	Chinese, Japanese, Korean (CJK), Thai, Indic, and certain control characters
4 bytes	Emoji characters, less common CJK characters and various historic scripts

Table 4. UTF-8 Character Length by Language

REFERENCES

Bouedo, Mickaël, and Beatrous, Stephen. 2013. “Internationalization 101: Give Some International Flavor to Your SAS® Applications.” *Proceedings of the SAS Global Forum 2013 Conference*, Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings13/025-2013.pdf.

Kiefer, Manfred. 2013. “Multilingual Computing in SAS® 9.4.” Technical Paper. Available at https://support.sas.com/resources/papers/Multilingual_Computing_with_SAS_94.pdf.

Kiefer, Manfred. 2009. “Processing Multilingual Data with the SAS® 9.2 Unicode Server”. Available at http://support.sas.com/resources/papers/92unicodesrvr.pdf.

Kiefer, Manfred. 2012. *SAS® Encoding: Understanding the Details*. Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

We would like to express our gratitude to the people who helped us create this paper. Jody Carlton and Jill Ackerman provided excellent feedback during multiple reviews and took time to test our SAS code. Diane Olson, Jack Wallace, Marie Dexter and Craig Martin provided helpful feedback during the review. And Juliane Foster wrote the code that we used for migrating format catalogs and indexes.

Finally, we want to thank two of our former colleagues. Steve Beatrous served as our fearless leader for many years. Steve, along with Shin Kayano, led the effort to design much of the internationalization support that SAS provides today. Manfred Kiefer mentored many of us to help us understand the issues that our customers in Europe faced. He wrote many papers that we rely on today for information about SAS support of Unicode. He even wrote the book on SAS® encodings and how they work (Kiefer 2012c). We miss you both and wish you much happiness in your retirements!

RECOMMENDED READING

- *SAS® 9.4 National Language Support (NLS): Reference Guide, Fifth Edition*
- *SAS® 9.4 Intelligence Platform: Installation and Configuration Guide, Second Edition*
- *SAS® 9.4 Language Reference: Concepts, Sixth Edition*

- *SAS® 9.4 Companion for UNIX Environment, Sixth Edition*
- *Base SAS® 9.4 Procedures Guide, Sixth Edition*
- *Moving and Accessing SAS® 9.4 Files, Third Edition*

CONTACT INFORMATION

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

Elizabeth Bales
SAS Institute Inc.
Elizabeth.Bales@sas.com

Wei Zheng
SAS Institute Inc.
Wei.Zheng@sas.com