SAS4195-2020

Your Data Will Go On: Practice for Character Data Migration

Edwin (You) Xie, SAS Institute Inc.

ABSTRACT

With the rapid advancement of technology, you inevitably face continual upgrades and changes in your working platforms. No matter how greatly the platform evolves, your work can keep in high continuity, and reliability if your data is successfully ported without being destroyed. Character data can be at risk during data migration, which is determined by its internationalization features such as encoding sensitivity and semantics dependence. This paper discusses the potential issues when moving your character data across environments, such as a migration to SAS® with UTF-8 or other encoding environments. It also demonstrates the use of common tools during the migration, such as the character variable padding (CVP) engine. By using these tools flexibly, no matter how the environment changes, your data will go on.

INTRODUCTION

Smith is working for a global commercial bank, which is turning to analytical insights powered by the SAS® Platform and its SAS® Viya® products. When porting the company data, he gets the message in Output 1. Although he is not clear about how "Cross Environment Data Access" (CEDA) works, he does not think it is a problem. It is acceptable for him to spend a little more time during initial migration if no data is lost.

```
NOTE: Data file BANKLIB.CLASS.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 1. Note Message during Data Access

However, Smith gets into trouble when facing the warning below. He is confused because he does not know the reason why it happens, and what the solutions are.

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

Output 2. Warning Message during Data Access

This is a common issue that frequently occurs in environment migration. This article analyzes its essential nature and discusses the principle of the solutions.

UNDERSTAND CHARACTER DATA

Character data is varied because it may have different encodings and lengths in different environments. However, once you have mastered its characteristics, you will never lose the direction when thinking and solving problems.

CHARACTER SET AND ENCODING

Different languages might be used in different regions in the world. The collection of characters in each language forms the "character set". Encoding is the way these characters live as a unique identifier or code point in the computer world. Character data migration is just like a traveler who needs to cross different regions. For the traveler, he needs to translate a foreign language to the native language so that local people can understand him. For data migration, a character string may also need to be "translated" to the SAS session encoding so that it will be recognized. When data is translated like this, it is called "transcoding".

To migrate character data smoothly, encoding awareness is the most primary prerequisite. The following example shows the attributes of a data set by the CONTENTS procedure. The encoding attribute indicates the way that characters live in the data file.

```
proc contents data=sashelp.class;
run;
```

```
                              The CONTENTS Procedure

  Data Set Name         SASHELP.CLASS              Observations          19
  Member Type           DATA                       Variables             5
  Engine                V9                         Indexes               0
  Created               11/03/2019 19:58:29        Observation Length    40
  Last Modified         11/03/2019 19:58:29        Deleted Observations  0
  Protection                                       Compressed            NO
  Data Set Type                                    Sorted                NO
  Label                 Student Data
  Data Representation   WINDOWS_64
  Encoding              us-ascii  ASCII (ANSI)
```

Output 3. Output from a CONTENTS procedure

In Output 3, "US-ASCII" is the encoding of this data set. The US-ASCII is a 7-bit encoding method that was developed in the US and is widely used to encode English data. There are other typical examples of encoding, such as Latin1 or Latin2 used in the Americas and Europe, Shift-JIS and EUC-CN in East Asia, and Unicode encoding schemes like UTF-8.

Encoding compatibility is another important concept to determine if transcoding is needed. The following program may help you check if the specified data set encoding is compatible with SAS® session encoding.

```
/* Get the encoding of the data set */
libname mylib 'path to library';
%let dsid=%SYSFUNC(open(mylib.class));
%let dsenc=%KSCAN(%SYSFUNC(attrc(&dsid,ENCODING)), 1, " ");
%let rc=% SYSFUNC(close(&dsid));

/* Get the session encoding */
%let sessenc=%SYSFUNC(getOption(ENCODING));

/*Check the compatibility */
%let isCompat=%SYSFUNC(encodCompat(&dsenc, &sessenc));
%put &isCompat; /* 1: compatible; 0: incompatible */
```

Once the data encoding is not compatible with the current session, CEDA can automatically transcode the source data into session encoding. That is the reason why Smith gets the message that CEDA is enabled.

Different encodings define different characters. When transcoding from one encoding to another, there may be characters that are not supported by the target encoding. Such as the Latin1 encoding does not contain Ideographs in Chinese. When getting the error message of "the data contains characters that are not representable in the new encoding", you need to realize the source contains characters that are invalid in the target environment. Successful transcoding requires the target encoding to contain all the characters in the source data.

SINGLE-BYTE, DOUBLE-BYTE, AND MULTIBYTE CHARACTER SET

A single-byte character set (SBCS) always uses exactly one byte for each graphic character. For example, the ISO 8859 and Windows single-byte encodings use only one byte to represent each character. When a character representation needs more than one byte, it is a multibyte character set (MBCS). A double-byte character set (DBCS) is one of MBCS that encodes characters in one or two bytes, such as Shift-JIS and EUC-CN. UTF-8 uses one to four bytes to represent a character in SAS products, so it is also an MBCS.

The preceding section mentioned transcoding might happen during data migration. However, when a character is transcoded among SBCS, DBCS, and MBCS, another issue arises: its byte length may also change. For example, the Euro sign, "€", is included in the encoding WLATIN1(Windows cp1252) where it is represented as a single-byte and has a code point represented as the hexadecimal value 0x80. When that character is transcoded to UTF-8, it requires 3 bytes. The hexadecimal value of the "€" in UTF-8 is 0xE282AC. This characteristic is the root reason for those outstanding truncation issues happening in character data migration. When transcoding from SBCS to MBCS, or DBCS to MBCS, the data byte length often needs to be increased. If SAS variable length is not long enough for the target string, you will be in trouble with data loss because of unexpected string truncation. Even worse, if the truncation happens in the middle of an MBCS character, the character will be corrupted to garbage, which would be risky in the subsequent string manipulation.

Another thing to keep in mind is that a character variable is often bound with a character format. To avoid string truncation during output the format length may need to be expanded along with the variable length.

ACCOMMODATE DATA EXPANSION

The previous discussion helps to clarify the reasons and potential risks during character data migration. To guarantee a successful migration, you need to know the way that source data is encoded. Then, figure out the encoding of the target environment. If the transcoding is from SBCS or DBCS to MBCS, you also need to consider changing variables length when needed. This session discusses the possible solutions in this adjustment.

EXPAND CHARACTER VARIABLES BY CVP

Character Variable Padding (CVP) engine is a read-only SAS® I/O engine. It is a recommended data pre-process platform for character data migration because of its efficiency, flexibility, extensibility, and centralized processing ability.

The data set below is created in Latin1 and contains student information of an international school class. For each student, the data set records name, gender, and age. The code below generates a subset of the data set:

```
libname mylib 'path to library';
data mylib.class;
    length Name $ 8 Sex $ 1;
    input  Name $ Sex $ Age;
    format Name $8. Age best2.;
    datalines;
André M 14
Valérie F 13
```

```
Béatrice F 13
;
run;
```

If you print this Latin1 data set in UTF-8, you will get the warning as Output 2, and the string "Béatrice" is truncated to "Béatric":

```
proc print data=mylib.class; run;
```

```
                         Obs     Name         Sex     Age

                          1      André         M      14
                          2      Valérie       F      13
                          3      Béatric       F      13
```

Output 4. Output from a PRINT Procedure.

The truncation happens because the string "Béatrice" needs 9 bytes in UTF-8, but the variable "Name" only has 8 bytes. To avoid the truncation, a solution is to specify "CVP" in the LIBNAME statement:

```
LIBNAME mylib CVP 'path to library';
proc print data=mylib.class; run;
```

```
                     Obs     Name                 Sex     Age

                      1      André                 M      14
                      2      Valérie               F      13
                      3      Béatrice              F      13
```

Output 5. Output from a PRINT Procedure with CVP.

In Output 5, All the names are printed successfully without any warnings, and the truncation issue is also solved.

By checking variables attributes with the CONTENTS procedure, you will see the length of all the character variables is expanded, and the numeric variable has no change. To your surprise, the width of the character format is also adjusted along with the size of character variables:

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

```
                          Variables in Creation Order

                     #    Variable     Type    Len    Format

                     1    Name         Char     16    $16.
                     2    Sex          Char      2
                     3    Age          Num       8    BEST2.
```

Output 6. Output from a CONTENTS Procedure with CVP.

Besides using the default setting above, the CVP engine also provides several options to control the adjustment of character variables and formats. The CVPMULTIPLIER= option specifies a multiplier value that expands the length of character variables in the processed SAS data file. In the case above, all character variables have been doubled by default in SAS Viya (In SAS 9, the length expands to 1.5 times by default). The valid number for CVPMULTIPLIER= is from 1 to 5.

Alternatively, you can specify a value of 0 to enable CVP to estimate the expanding rate to avoid truncation in transcoding. The auto-value of CVPMULTIPLIER is detected depending on the SAS session encoding of the target environment and self-adjusted based on each file encoding in the library. For example, you have a library that contains 3 data sets with 3 different encodings: US-ASCII, Latin1, and EUC-CN. When using "CVPMULTIPLIER=0" in a UTF-8 session, the multipliers will be 1, 2 and 1.5 accordingly.

The CVPBYTES= option supplies another way of expansion. By explicitly specifying the exact number of bytes, the length of each character variable can be incremented by the fixed size.

The CVPFORMATWIDTH= option is the mechanism to ensure the width of format that can keep synchronization with character data length expansion. The default value is YES meaning that character formats will be expanded together with character variables.

The CVP engine offers great possible flexibility in selecting operations. CVPINCLUDE= or CVPEXCLUDE= can be used to filter data expansion by specifying variable names. The CVPINCLUDE= specifies which variables to process and the CVPEXCLUDE= specifies which variables not to process. The two options cannot be used together. If both options are omitted, then all character variables are processed. Both options support case-insensitive Perl regular expressions. For example, when specifying CVPINCLUDE="name", all variable names that contain "name" will be selected, such as "First_Name" and "Last_Name". You can also use CVPINCLUDE="^name$" to fully qualify the variable name to "name". Both "^" and "$" are Perl regular expressions metacharacters. The "^" matches the beginning of a string and the "$" matches the end of a string. You can refer to *Tables of Perl Regular Expression (PRX) Metacharacters* for the complete list for the metacharacters.

If at least one CVP option is specified, the CVP engine is automatically called, and the keyword "CVP" can be omitted in the LIBNAME statement.

The following example demonstrates how the CVP engine integrates CVPBYTES= and CVPEXCLUDE= together for the precise adjustment. Only the character variable "Name" will be increased 1 byte in size, as well as the associated character format:

```
libname mylib 'path to library' cvpbytes=1 cvpinclude="name";
proc contents data=mylib.class; run;
```

```
                    Variables in Creation Order

              #     Variable     Type     Len     Format

              1     Name         Char       9     $9.
              2     Sex          Char       1
              3     Age          Num        8     BEST2.
```

Output 7. Output from a CONTENTS Procedure with CVP.

The CVP engine does not directly access data files. Instead, it always cooperates with other engines for special adjustments in data sets. The CVPENGINE= option can specify the engine to process SAS data files (the default is BASE engine). For example, if the "class" data set above is stored by the SAS® Scalable Performance Data Engine (SPD Engine), the following statement may help to expand its character variables with CVP engine:

```
libname mylib 'path to library' cvpengine=SPDE;
proc contents data=mylib.class; run;
```

```
                     Engine/Host Dependent Information

                  Underlying Engine Accessing Data  SPDE
                  Blocking Factor (obs/block)       43690
                  Data Partsize                     134217216


                         Variables in Creation Order

                     #    Variable     Type    Len    Format

                     1    Name         Char     16    $16.
                     2    Sex          Char      2
                     3    Age          Num       8    BEST2.
```

Output 8. Output from a CONTENTS Procedure with SPD Engine.

CVP is a read-only engine. It does not save modifications back to data files. You may use the COPY statement in the DATASETS procedure to make permanent storage. For example:

```
libname oldlib "path to the original library" cvpmultiplier=0;
libname newlib "path to save the converted data sets";
proc datasets nolist;
   copy in=oldlib out=newlib override=(encoding=session outrep=session)
        memtype=data;
run; quit;
```

In the case above, all the data sets in the original library are copied to a new library. By specifying "override=(encoding=session outrep=session)", the data sets encoding is transcoded to the SAS session encoding, and the data representation is converted to the representation of the operating environment where your SAS session is running. After this operation, all the new data sets are native to the current environment. You do not need any help from CEDA to access them, and no truncation occurs.

EXPAND CHARACTER VARIABLES IN CAS SERVER

SAS Viya often needs to load data sets to SAS® Cloud Analytic Services (CAS) whose session encoding is UTF-8. As described above, you may realize that character data might also have truncation risk if the client encoding is different from CAS server. There are 3 options that are ready to serve you here:

- System options CASNCHARMULTIPLIER=
- LIBNAME option NCHARMULTIPLIER=
- Data Set option NCHARMULTIPLIER=

These 3 options are used to define a multiplier whose value is greater than 1 and less than or equal to 4. The CASNCHARMULTIPLIER= system option affects all CAS engines. The NCHARMULTIPLIER= LIBNAME option works for only one CAS engine, and the NCHARMULTIPLIER= data set option just acts for single data set. If multiple options are specified, the latter ones overwrite the previous ones. The default value of these options is based on the encoding of the clients: 1 for SBCS clients and 1.5 for DBCS clients.

Assuming that a SAS client is working under a Latin1 session and has already connected to a CAS server (whose session encoding is UTF-8), the code below attempts to load a data set to CAS tables with these options:

```
data class;
   name="Béatrice";
```

```
run;

libname sys cas;
libname lib cas NCHARMULTIPLIER=1.5;

data sys.class_sys;
   set class;
run;

data lib.class_lib;
   set class;
run;

data lib.class_ds(NCHARMULTIPLIER=2);
   set class;
run;
```

The data set "class" is in the client session encoding Latin1. It has a character variable "name" that has a length of 8. The sample code creates two CAS librefs. The "sys" libref uses the default system option CASNCHARMULTIPLIER=. Since the client encoding Latin1 is SBCS, the default multiplier is 1, meaning no variables will be expanded. The LIBNAME statement for the "lib" libref specifies LIBNAME option NCHARMULTIPLIER= to 1.5. All character variables read by this engine will have 1.5 times size increment, regardless of whether CASNCHARMULTIPLIER= is specified.

The code above should create three CAS tables. Even though the SAS session encoding is Latin1, all the CAS tables are encoded in UTF-8 because CAS server session encoding is UTF-8. The "class_sys" table should be generated under the effect of the default system option CASNCHARMULTIPLIER=. The "class_lib" table is generated by the specifying NCHARMULTIPLIER=1.5 in LIBNAME. And the "class_ds" table is the result of the specifying NCHARMULTIPLIER=2 in the data statement.

However, the creation of "class_sys" will fail with the error message in Output 9 because the string "Béatrice" needs 9 bytes in UTF-8, but the variable has only 8-bytes length.

```
ERROR: Some character data was truncated during transcoding in the dataset
       CAS1.CLASS1. Use of the NCHARMULTIPLIER option is recommended.
ERROR: An error has occurred.
```

Output 9. Error Messages in SET Statement

To prevent the error and force CAS to store the truncated string into the "class_sys" table, specify the LIBNAME option TRANSCODE_FAIL=WARN. The string will turn to "Béatric" in this case.

The variable "name" in "class_lib" has a length 12 because the LIBNAME option overrides the system option and expands the variable to it 1.5 times. The same variable in "class_ds" has a length 16 because the data set option overrides the LIBNAME option.

EXPAND CHARACTER VARIABLES IN TRANSPORT FILE

Character data migration also happens in SAS transport files when data is ported from one environment to another. The CPORT procedure exports data to a transport file in the source environment. In the target environment, the CIMPORT procedure reads the transport file and converts it to a native format.

Using a similar policy as the CVP engine, PROC CIMPORT provides options to expand character variables and formats. The "EXTENDVAR=multiplier | AUTO" option specifies the multiplier. It supports a value between 1 to 5, or "AUTO" to let the procedure automatically choose the multiplier based on the encodings of the current session and the transport file.

The "EXTENDFORMAT=YES | NO" controls the extension of character formats. In the example below, MYLIB.CLASS data set is exported from a Latin1 environment and then imported into a UTF-8 environment:

```
/* Export the data set to a transport file in the source environment */
libname mylib 'SAS-data-library';
filename cportout 'transport-file';
proc cport data=mylib.class file=cportout; run;

/* Import the transport file in the target environment */
/* Extend all the character variables and their format */
/* 1.5 times to avoid truncation.                      */
filename infile 'transport-file';
libname target 'SAS-data-library';
proc cimport infile=infile library=target extendvar=1.5; run;
```

CHARACTER SEMANTICS AND DATA MIGRATION

It would be interesting to look at character migration from semantics perspective – byte and character semantics. Byte semantics is usually machine-oriented because the inside of computer processing and storage are based on byte. However, character semantics is human-oriented with linguistic meaning. The regions using SBCS might be less sensitive to the difference between byte and character where a character always occupies one byte, and the number of bytes is always equal to the number of characters. But this behavior has an obvious difference in the regions that are using MBCS and in UTF-8, where many characters need more than 1 byte. Treating a character as a byte there is incorrect. When people step into the Cloud, data must have liquidity for free exchange. Character-based processing is a feasible strategy for migration and is more consistent and intuitive than byte semantics.

CHARACTER SEMANTICS CONVERSION WITH CVP

Besides expanding character variables, the CVP engine can also convert byte semantics variables to character semantics variables. The following example demonstrates how the CVP engine converts CHAR variable (byte semantics) to VARCHAR data (character semantics). After the conversion, no matter where the string goes, truncation will not happen because character semantics supported environments already guarantee the integrity of the character data:

```
libname original 'path to library';
proc contents data=original.class; /* The original data set */
run;

libname mylib 'path to library' cvpvarchar=yes;
proc contents data=mylib.class; /* The data set is a read-only copy */
run;
```

```
                Variables in Creation Order

         #    Variable     Type    Len    Format

         1    Name         Char      8    $8.
         2    Sex          Char      1
         3    Age          Num       8    BEST2.
```

Output 10. The Variables in the Original Data Set.

```
                     Variables in Creation Order

          #    Variable    Type         Bytes    Chars    Format

          1    Name        Varchar         32        8    $32.
          2    Sex         Varchar          4        1
          3    Age         Num              8             BEST2.
```

Output 11. The Variables Converted by CVP Engine.

The CVPVARCHAR= option helps to enable the CVP engine to convert variables from CHAR to VARCHAR. In Output 11, the character variables have 2 length properties: "Bytes" and "Chars". The "Chars" indicates the length in characters, which values are equal to the length of the original variables in Output 10. The "Bytes" is the storage length in bytes, which values depend on the session encoding. For example, the current session encoding is UTF-8, and one UTF-8 character can be up to 4 bytes in SAS. Thus, for an 8-char length variable, its storage length in bytes is 8 x 4=32 bytes.

Since VARCHAR columns are only supported in CAS tables, to keep the converted data set, you may need to save it into a CAS table:

```
/* Specify the connection information. */
option casport=5570 cashost="cloud.example.com";

/* Create a CAS session. */
cas sascas1 user=&SYSUSERID;

/* The libref mycas is associated with the active caslib. */
libname mycas cas;
libname mylib 'path to the client library' cvpvarchar=yes;

/* Load the data set to CAS using CAS engine. */
proc copy in=mylib out=mycas;
   select class;
run;
```

CHARACTER SEMANTICS CONVERSION WITH DATA CONNECTOR

CAS Data Connector is another useful tool that can convert character variables. For example, you may want to do the following tasks at the same time:

- Load the data set "MYLIB.CLASS" to CAS.
- Convert the variable "Name" to VARCHAR.
- Keep the variable "Sex" as CHAR but double its length.

Before loading the data file, the CASLIB statement may help to list caslibs and their settings:

```
caslib _all_ list;
```

```
NOTE: Session = SASCAS1 Name = CASUSER(userid)
         Type = PATH
         Description = Personal File System Caslib
         Path = /u/userid/
         Definition =
         Subdirs = Yes
         Local = No
         Active = No
         Personal = Yes
NOTE: Session = SASCAS1 Name = CASUSERHDFS(userid)
         Type = HDFS
         Description = Personal HDFS Caslib
         Path = /user/userid/
         Definition =
         Subdirs = Yes
         Local = No
         Active = Yes
         Personal = Yes
```

Output 12. Output from a CASLIB Statement.

In Output 12, "CASUSER(userid)" points to your personal file system and "CASUSERHDFS(userid)" is the active caslib to access HDFS in CAS. The data file at the specified path "/u/userid/data/class.sas7bdat" needs to be load into CAS:

```
proc casutil;
  load casdata="data/class.sas7bdat" /* relative path to the data file   */
       incaslib="CASUSER(userid)"    /* input caslib that points to      */
                                     /* /u/userid                        */
       casout="class_cas"            /* CAS table name, using the active */
                                     /* caslib CASUSERHDFS(userid)       */
       importOptions={ filetype="basesas", /* Data Connector options     */
                       VarcharConversion=2,
                       CharMultiplier=2 };
  altertable casdata="class_cas" columns={{name="Name", format="$9."}};
quit;
```

Data Connector option "VARCHARCONVERSION=2" defines a threshold. Only the variables longer than 2 will be converted to VARCHAR. Option "CHARMULTIPLIER=2" doubles the size of the rest of CHAR variables. Since Data Connector does not change the format width, an ALTERTABLE statement is used to update the character format width after the loading. The output from the CONTENTS procedure below shows the final table attributes:

```
libname mycas cas;
proc contents data=mycas.class_cas; run;
```

```
                         Variables in Creation Order

                                                            Max
                                                          Bytes
   #    Variable    Type          Bytes    Chars           Used    Format

   1    Name        Varchar          32        8              9    $9.
   2    Sex         Char              2
   3    Age         Num               8                            BEST2.
```

Output 13. Output from a CONTENTS Procedure.

HANDLE UNEXPECTED CHARACTERS

Unexpected characters may exist in data sets and cause troubles before and after data migration. Dealing with them properly can smooth the data migration process.

CONVERT UNEXPECTED PUNCTUATION MARKS

When a United States customer creates data using an SBCS encoding (such as WLATIN1), all characters are expected to be US-ASCII (basic English in 7 bits, non-extended ASCII) that only contains English alphabets and plain symbols. However, some punctuation characters might be automatically converted to extended ASCII characters (8-bit) by another software product that supports AutoFormat. For example, when pressing the apostrophe on the keyboard, the AutoFormat feature may place left or right curly double quotation (0x93 or 0x94 in WLATIN1) in the string instead of the straight double quotation (0x22). Since extended ASCIIs are not represented directly in UTF-8, you may see garbage when the quotation marks are present (treated as ASCIIANY under Unicode session). Data size expansion also needs to be considered because the extended ASCIIs turn to multiple bytes after transcoded to UTF-8.

The KPROPDATA function provides an option "PUNC" to convert commonly used 8-bit punctuation marks to 7-bit ones. It can be used to look through data and normalize unexpected punctuation for a better migration experience.

Suppose a school class keeps all its notification messages in a WLATIN1 data set. Below is an example of one of the messages:

```
libname mylib 'path to library';
data mylib.notice;
   message="We will visit the "National Museum" next Friday";
run;
```

The message contains curly quotation marks. If the data set is opened in a SAS session with UTF-8, the string will be truncated because these quotation marks are extended ASCII and need extra bytes in UTF-8:

```
proc print data=mylib.notice; run; /* execute it in UTF-8 session */
```

```
          Obs                        message

           1     We will visit the "National Museum" next Fr
```

Output 14. Output from a PRINT Procedure in UTF-8.

Since all characters in "MYLIB.NOTICE" are in 7-bit except for some unexpected 8-bit punctuation, KPROPDATA with option "PUNC" may help to access the data successfully without any loss under the UTF-8 session. Specifying "inencoding=asciiany" in the LIBNAME statement will disable the transcoding by CEDA. Then KPROPDATA will explicitly convert the punctuations to ensure that all data is real US-ASCII, which size will not change across environments:

```
libname mylib 'path to library' inencoding=asciiany;

data test;
   set mylib.notice;
   message = kpropdata(message, 'PUNC', 'wlatin1');
run;

proc print data=test; run;
```

```
               Obs                            message

                1      We will visit the "National Museum" next Friday
```

Output 15. Output from a PRINT Procedure in UTF-8.

ELIMINATE UNPRINTABLE CHARACTERS

Unsuccessful data migration may leave some binary garbage in strings. Those binaries are usually generated by mismatched encoding or corrupted multi-byte characters and are not valid. They are the so-called unprintable characters here. These characters not only cause garbage to display but are also risky in character manipulation because they may lead to unexpected results. To be safe, it is better to remove unprintable characters from the data.

Suppose there is a UTF-8 data set that contains an unprintable character:

```
data corrupted;
    length text $ 30;
    text = 'The SAS® System in UTF-8';
    output;
    text = 'The SAS' || 'ae'x || ' System in Latin1';
    output;
run;
```

The 0xAE in the second observation represents the register trademark symbol in Latin1 but is invalid in UTF-8. KPROPDATA with the "TRIM" option removes that unprintable character and makes the data clear:

```
data removed;
    set corrupted;
    text = kpropdata(text, 'TRIM');
run;
proc print data=removed; run;
```

```
                          Obs                text

                           1      The SAS® System in UTF-8
                           2      The SAS System in Latin1
```

Output 16. Output from a PRINT Procedure.

If the string is suspected to contain Latin1 characters, KPROPDATA will help to detect unexpected 8-bit Latin1 characters and revise the binaries into the valid UTF-8 characters:

```
data fixed;
    set corrupted;
    keep new;
    new = kpropdata(text, 'REMOVE'); /* Remove the data string if any    */
                                     /* unprintable characters are found */
    if new = ' ' then
       new = kpropdata(text, 'REMOVE', 'latin1'); /* Transcode as Latin1 */
run;
proc print data=fixed; run;
```

```
                         Obs                   new

                          1      The SAS® System in UTF-8
                          2      The SAS® System in Latin1
```

Output 17. Output from a PRINT Procedure.

In the code above, the "REMOVE" option tells the first call to KPROPDATA to remove the whole string if any invalid characters are found. If a blank string is returned, that triggers the second call to KPROPDATA. The string is explicitly transcoded from Latin1 to session encoding UTF-8 because the invalid characters binaries are already known as Latin1 characters. Finally, the string is repaired as expected.

CONCLUSION

In today's big data era, data is a valuable asset for many companies. Mastering data migration methods and techniques, helps you minimize the possibility of data corruption when switching platforms. According to the characteristics of character data, this paper analyzes the common issues during data migration and gives the corresponding solutions.

This paper does not cover everything that happens in data migration. Each product may supply special tools or solutions for its data migration support. You may also encounter other issues not discussed in this article. However, as long as you understand the principles of character data migration, you can identify potential problems and solve them gracefully. In this way, no matter how the platform changes in the future, you can transfer your data to the new platform without any obstacles. Your data will be your most solid asset, along with the latest analytics technology, to ensure the success of your business.

REFERENCES

Elizabeth Bales and Wei Zheng. 2017. "SAS® and UTF-8: Ultimately the Finest. Your Data and Applications Will Thank You!" *Proceedings of the SAS Global 2017 Conference*, Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings17/SAS0296-2017.pdf.

ACKNOWLEDGMENTS

I would like to express my gratitude to the people who helped make this paper possible. Elizabeth Bales, Lane (Li) Li supported my efforts in researching and writing this paper.

RECOMMENDED READING

- *Moving and Accessing SAS® 9.4 Files, Third Edition*
- *Migrating Data to UTF-8 for SAS®Viya® 3.5*
- *SAS® Cloud Analytic Services 3.5: User's Guide*

CONTACT INFORMATION

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

Edwin (You) Xie
SAS Research and Development Co., Ltd., Beijing
+86 10 83193674
you.xie@sas.com