PharmaSUG 2018 - Paper BB-14

Tips and Fixes for Cross-Environment Batch Transfer of SAS® Data

Yun (Julie) Zhuo, Axio Research LLC

ABSTRACT

Contract research organizations (CROs) frequently receive data transfers from pharmaceutical companies or other clinical organizations. A batch process is commonly used as we need to transfer a number of files multiple times. A smooth transfer is the first key step to a successful project. This paper cautions against some common errors during the batch process, and it presents tips on resolving these issues. The tips include selecting the correct conversion method, avoiding data truncation or data loss, converting data set encodings, and building a quality process. Code samples are provided. The scope of this paper is limited to the conversion of SAS data sets and/or transport files at the receiving end of the transfer process. The primary audience includes programmers, statisticians, and data managers responsible for receiving and converting data.

INTRODUCTION

TERMINOLOGY

For the purpose of the following discussion, a *data transfer* is defined as moving SAS entities between host operating environments. Examples of SAS entities we frequently receive at CROs are SAS data sets and SAS transport files. Different host environment may use different operating system and data representation.

Data representation is the method for structuring data in computers. In our discussion, we are concerned with numeric data representation, as well as character data encoding such as the commonly used UTF-8 and WLATIN1 encodings.

Encoding in computers is the process of putting a sequence of characters into a specialized format for efficient transmission or storage. *Transcoding* during data transfer is the process of converting SAS entities from one computer encoding to another. Transcoding is often necessary when the encodings are different between source and target computing environments.

Cross environment data transfer moves SAS entities while performing the necessary conversion of the entities from one operating environment's representation to another's. For example, the SAS transport file format and the SAS Cross Environment Data Access (CEDA) are two SAS strategies to tackle the issue of cross environment incompatibilities.

SCOPE OF THE DISCUSSION

This paper discusses the transfer of SAS data sets and/or SAS transport files in a batch process. Although SAS/CONNECT could be an elegant method for data transfers, it is not discussed, as it is not realistic or efficient in many situations. It requires a separate license. It cannot be used if source and target computers are not on the same network. A batch process is more efficient for the purpose of receiving a number of SAS entities multiple times.

Figure 1 illustrates the entire data transfer process. The scope of this paper is limited to the tasks at the receiving end of the transfer process. Although precautious steps could be taken at the delivering end of the transfer, as data recipients, we usually have no or little control over the procedures at the other end of the process. This paper also assumes there is no data corruption during data transmission. SAS version 9.4 is used, and the issue of backward compatibility will not be discussed.

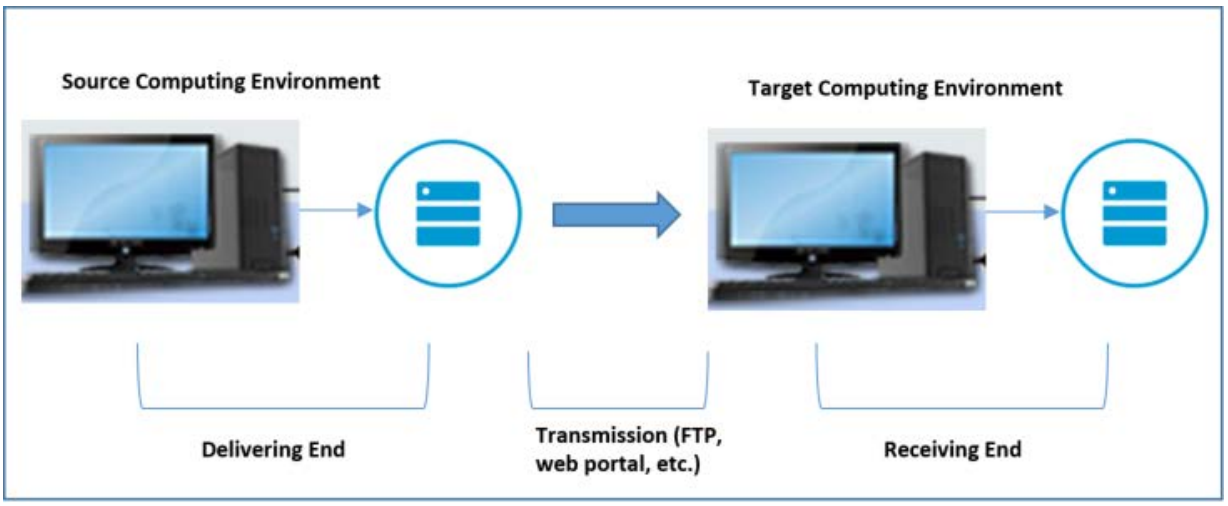


Figure 1. Data Transfer Process.

TIPS AND FIXES

In this section, we briefly review the techniques that convert source data to its target environment. We caution against some potential pitfalls, and we provide tips and fixes to avoid common mistakes.

SELECTING A COMPATIBLE CONVERSION TECHNIQUE

A number of conversion techniques are available. Using a technique that is incompatible with the source data could trigger errors.

1. The CIMPORT procedure imports a transport file that was created by the CPORT procedure. It automatically converts the transport file to its target environment as it imports it. The following example converts multiple data sets from a transport file:

```
filename source 'transport-file';
libname target 'SAS-data-library';
proc cimport infile=source library=target memtype=data;
run;
```

   If PROC CIMPORT is used to import a transport file created by an incompatible technique such as the XPORT engine, an error message will show up in the log:

```
ERROR: CIMPORT is not able to read XPORT engine data sets.
```

   Additionally, starting in SAS 9.4, the CIMPORT method requires that the target session use compatible encodings with the source data set, unless the encoding value of the data set is ASCIIANY. Starting in SAS 9.4M3, the procedure supports the ability to import non-UTF8 source data sets into UTF-8 SAS sessions.

2. The XPORT engine with the COPY procedure imports and converts a transport file that was created by the XPORT engine at the source computer:

```
libname source xport 'transport-file';
libname target 'SAS-data-library';
proc copy in=source out=target;
run;
```

   Using the XPORT engine on a transport file created by PROC CPORT will trigger the following error message:

```
ERROR: File is probably a cport file. XPORT engine unable to read
file created by proc cport.
```

3. The XML engine with PROC COPY imports and converts Extensible Markup Language (XML) documents only. XML documents can be opened with text editors, and they contain markup symbols similar to Hypertext Markup Language (HTML). Below are SAS codes to import XML documents:

```
libname source xml 'XML-document';
libname target 'SAS-data-library';
proc copy in=source out=target;
run;
```

4. If the incoming data is SAS data sets, PROC COPY is the preferred simple strategy. SAS CEDA will be automatically invoked to convert the 'foreign' formats of the source computer to the 'native' formats of the target computer. If the data sets are in the 'foreign' formats from a different computing environment, be sure to specify the NOCLONE option in PROC COPY:

```
libname source 'Source-data-library';
libname target 'Target-data-library';
proc copy in=source out=target NOCLONE memtype=data;
run;
```

Due to potential pitfalls from incompatible techniques, it is best practice that we look before we leap when it comes to selecting a correct conversion method. It is worthwhile that you spend a few minutes upfront to understand what you get. There are some tips that can assist you in the process:

1. You can identify which strategy was used to create the transport files at the source computer by examining the file headers using a text editor.

   If the XPORT engine was used, the first 40 characters of the file contains the following text:

```
HEADER RECORD*******LIBRARY HEADER RECORD!!!!!!!00
```

   If PROC CPORT without the NOCOMPRESS option was used, the first 40 characters contains the following text:

```
**COMPRESSED** **COMPRESSED** **COMPRESSED** **COM
```

   The method works for any operating environment such as UNIX and Windows that uses ASCII to store character data.

2. An alternative solution to identify whether the XPORT engine or PROC CIMPORT was used to create the transport file is to examine the result in the log from running the following codes:

```
filename tranfile 'transport-file';
data _null_;
    infile tranfile obs=5;
    input theline $ASCII80.;
    put theline;
run;
```

   If the XPORT engine was used, the following text will appear in the log:

```
HEADER RECORD*******LIBRARY HEADER RECORD!!!!!!!00
```

   If PROC CPORT was used, the following text will appear:

```
**COMPRESSED** **COMPRESSED** **COMPRESSED** **COMPRESSED**
```

   This method will also work for the z/OS operating environment that does not use the ASCII encoding.

3. Identifying the source format of the SAS data can be accomplished by examining the output from the CONTENTS procedure. For example, the following text in the output shows that the SAS data uses the UNIX format and the UTF-8 encoding:

```
Data Representation HP_UX_64, RS_6000_AIX_64, SOLARIS_64, HP_IA64
```

```
Encoding utf-8 Unicode (UTF-8)
```

By contrast, the following text in the output shows that the SAS data uses the Windows format and the WLATIN1 encoding:

```
Data Representation WINDOWS_64

Encoding wlatin1 Western (Windows)
```

If the target computer uses a different operating environment, SAS will automatically invoke CEDA, and writes the following message to the log:

```
NOTE: Data file 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.
```

To conclude, in order to avoid potential problems, it is important that we take the following steps to check the source data format:

1. Look for file formatting information in the source documentation, if there is any.
2. Identify if the source data is transport file, XML document, or SAS data sets.
3. Identify the creation technique for the transport files.
4. Identify the source data representation of the SAS data.
5. Only use conversion techniques that are compatible with the source data formats.

AVOIDING DATA TRUNCATION WITH THE CVP ENGINE

Identifying and implementing a correct conversion strategy may not resolve all the problems, especially when it involves transcoding. You might encounter the following error message when you try to convert a source data set that has a different encoding:

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

What happened? In order to understand the problem, we need to take a step back and make sure we understand the concept of character set encoding. In computers, character data is stored as a series of bytes, and a coded character set associates each character with a number. Computer encoding maps the bits and bytes of stored data to the correct characters via the coded character set. Table 1 gives examples of some common encodings. The last column of the table shows that the storage size of each encoding differs. While the single byte character set (SBCS) only takes up one single byte, the multiple byte character set (MBCS) needs up to four bytes to store characters.

Name of Encoding	Character Set	Size
WLATIN1	ASCII and Extended ASCII character set	All 256 characters are stored in a single byte Single Byte Character Set (SBCS)
SHIFT-JIS	ASCII, Katakana, and other Japanese characters	Characters are stored in either 1 or 2 bytes. Double Byte Character Set (DBCS)
UTF-8	Unicode character set including ASCII, foreign languages, special symbols, and more	Over 120,000 Characters are stored in 1 to 4 bytes. Multi-Byte Character Set (MBCS)

Table 1. Encoding Examples.

Now that we understand different encoding may require different numbers of bytes to store character data, it becomes clear that if the target data set uses a MBCS such as the UTF-8, the length of the character variables defined in the source data set may not be sufficient to hold the values for storage in the target encoding environment.

To find out the encoding of your session, you can submit the following code and look for the results in the log:

```
%put encoding=%sysfunc(getoption(encoding));
```

And the example code below gives you the encoding of the source data set:

```
%let dsn=source.datasetname;
%let dsid=%sysfunc(open(&dsn,i));
%put &dsn ENCODING is: %sysfunc(attrc(&dsid,encoding));
%let rc=%sysfunc(close(&dsid));
```

Alternatively, as mentioned in the previous section, you can also look for the encoding information in the PROC CONTENTS output. The technique works for both source and target SAS data sets. For example, the following message in the output indicates the data set uses the WLATIN1 encoding:

```
Encoding wlatin1 Western (Windows)
```

If you find out that the encoding of your session uses the MBCS but the encoding of the source data set uses the SBCS or the DBCS, the error message you encountered above usually means that there is not enough space in one or more character columns to convert the data to the target encoding. If this is the case, you should consider using the Character Variable Padding (CVP) engine to avoid truncation.

The CVP engine expands the length of the character variables at your request. It is a read-only engine. It serves as an intermediate engine that is used to prepare the data for transcoding. After the lengths are increased, the primary engine is used to do the actual processing.

By default, the character variable lengths are multiplied by 1.5. Here is the example code that uses the CVP engine with the default multiplier of 1.5:

```
libname source cvp 'Source-data-library';
```

In order to multiply the character variable lengths by a different amount, you need to use the CVPMULTIPLIER= option to specify a number. You can specify a value from 1 to 5, or you can specify a value of 0 to let the CVP engine determine the amount. Here is the example code that uses a user-specified multiplier:

```
libname source cvp 'Source-data-library' cvpmultiplier=2.0;
```

It may take some experimenting to determine the correct multiplier in order to make sure you have adequate space for all situations. During the experimenting process, be sure to review the log carefully and watch for truncation error messages.

Since the CVP engine is read-only, an additional LIBNAME statement is required in order to save a permanent copy of the converted data set. Here is the full example code:

```
libname source cvp 'Source-data-library';
libname target 'Target-data-library';
proc copy noclone in=source out=target;
run;
```

In the above example, the NOCLONE option in PROC COPY is required. The option makes sure SAS adopt the target operating system data representation, the target session encoding, and other relevant attributes.

To conclude, we run the risk of truncating character data when transcoding into a character set that stores data in multiple bytes. By expanding the length of the character variables, the CVP read-only LIBNAME engine is an effective and elegant solution to avoid data truncation. For more information

regarding the CVP engine, please refer to the documentation in the SAS 9.4 National Language Support (NLS): Reference Guide.

AVOIDING CHARACTER DATA LOSS

Although the CVP engine resolves the issue of truncation, it is not helpful if the following error message was triggered by the characters that are not representable in the target encoding:

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

What happened this time? To better understand the issue, we should re-visit the encoding examples in Table 1. Some encodings such as the UTF-8 encoding store more than 120,000 characters covering 129 scripts and multiple symbol sets. ASCII characters, Asian languages and non-ASCII special characters are all represented in the UTF-8 encoding. By contrast, the WLATIN1 encoding, a commonly used encoding in the Western world, only contains 256 ASCII and extended ASCII characters. Any character that is not included in the WLATIN1 encoding will be lost if we try to transcode from UTF-8 to WLATIN1.

Therefore, for example, if the source data encoding is UTF-8 but the target session encoding is WLATIN1 (or any encoding other than the comprehensive UTF encoding), the error message stating there are characters that are not representable in the target encoding may be triggered. To find out the encoding of the source data set and the session encoding in the target computer, refer to the tips from the previous section.

Then what do we do? This section will discuss both a simple solution to bypass the error message, and a more technically correct solution to recover the lost data.

The first solution is to suppress the CEDA transcoding and then use the KPROPDATA() function to transcode a character string from and to any encoding. In some cases, it can be appropriate to bypass transcoding errors so that you can finish your transfers. If you do not mind seeing a weird symbol in your data for any character that is not represented in the target encoding, you could simply bypass the error message with the ENCODING=ASCIIANY option:

```
data target.ae;
  set source.ae (encoding=asciiany);
run;
```

You can also use the KPROPDATA() function to convert characters that are not represented in the target encoding to a character of your choice, including Unicode escape (\Uxxxx) or Numeric Character Reference (NCR) format. SAS provides the following macro so that you can accomplish the task with ease:

```
%macro sas_iconv_ds(in,out,from=UNDEFINED,to=UNDEFINED,sub='?',file_opt=);
   data &out;
     set &in(encoding=asciiany);
     array cc (*) _character_;
     do _N_=1 to dim(cc);
         cc(_N_)=kpropdata(cc(_N_),&sub,"&from","&to");
     end;
     run;
     %let lib=%scan(&out,1,%str(.));
     %let mem=%scan(&out,2,%str(.));
     %if %length(&mem) = 0 %then %do;
         %let mem=&lib; %let lib=work;
     %end;
     proc datasets lib=&lib nolist; modify &mem / correctencoding=&to;
     run; quit;
%mend;
%sas_iconv_ds(WORK.countries,WORK.countrieslt1,from=utf-8,to=wlatin1);
```

The required parameters for the above macro are the names of the source and target data sets. The optional parameters are source and target encoding values. By default, characters that are not representable in the target encoding are converted to question marks. A third optional parameter allows you to instruct KPROPDATA to change the default question marks to other character of your choice. For more information about the macro, refer to the SAS technical paper on Multilingual Computing with SAS 9.4.

The second solution will allow you to recover the lost characters. The key is to invoke your target SAS session with the UTF encoding or the encoding of the source data so that all characters in the source data can be represented in the target environment.

To embark on the task, you need to firstly find out the encoding in the source data sets. The previous section has the example code to help you accomplish this. For the purpose of the following discussion, let's take the UTF-8 as the example for the source data set encoding.

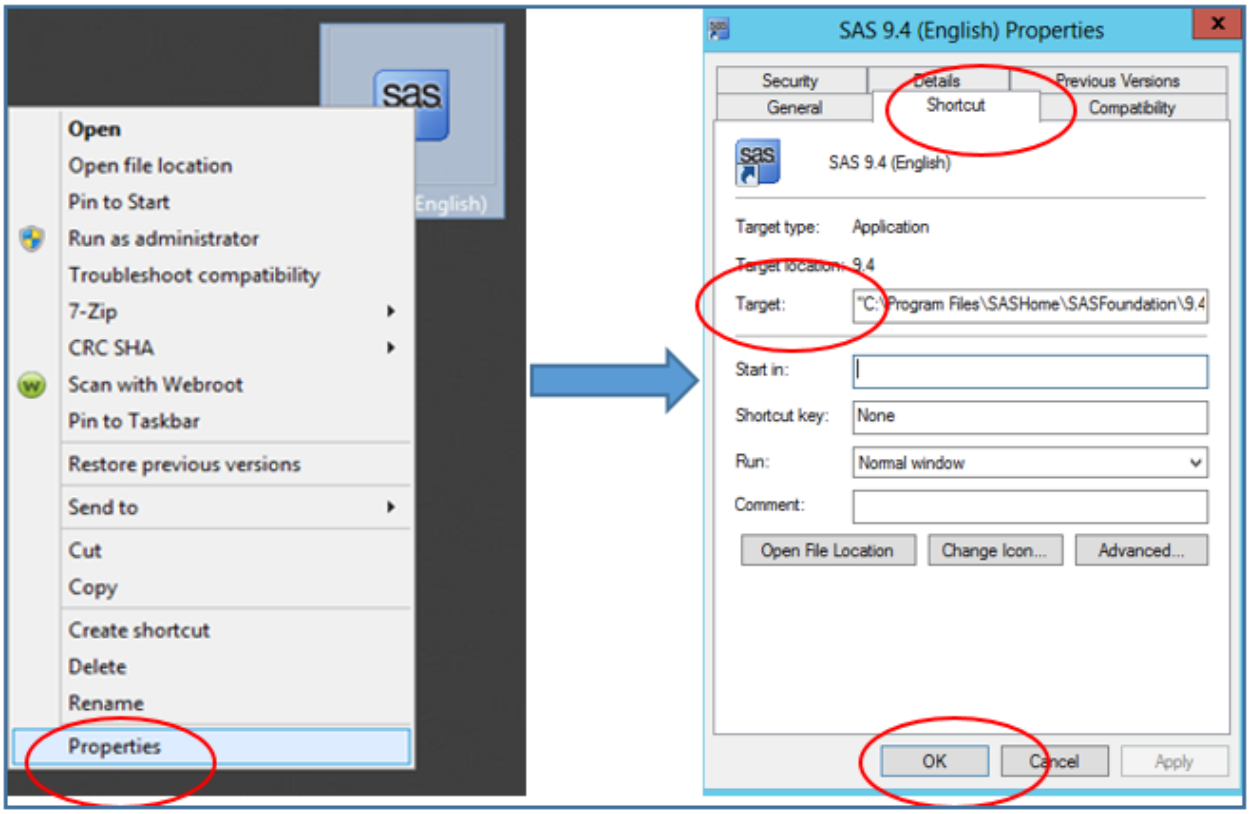
After you make the decision on the encoding, you can take advantage of the National Language Support (NLS) to update your SAS session encoding. NLS is included in SAS 9.2, 9.3, and 9.4. By default, the Unicode Support is already installed during SAS deployment.

If you use the Windows platform and a SAS shortcut icon is created in your computer, you can customize your shortcut icon to instruct SAS to invoke UTF-8 by following the steps below, as illustrated in Display 1.

1. Locate your SAS shortcut icon.
2. Right-click the SAS shortcut icon and select Properties.
3. On the Shortcut tab, in the Target line, update to instruct SAS to use the UTF-8 version of the NLS configuration file (sasv9.cfg):

```
-config "C:\Program Files\SASHome\SASFoundation\9.4\nls\u8\sasv9.cfg"
```

4. Click OK.



Display 1. Update session encoding on Windows.

Alternatively, you can also click through the following in your Windows machine to invoke SAS with the UTF-8 encoding:

Start Menu ➡ All Programs ➡ SAS ➡ Additional Languages ➡ SAS9.x (Unicode Support)

If you runs SAS on a server versus a local machine, you have the option of creating a SAS shortcut icon before customizing it following the steps illustrated in Display 1.

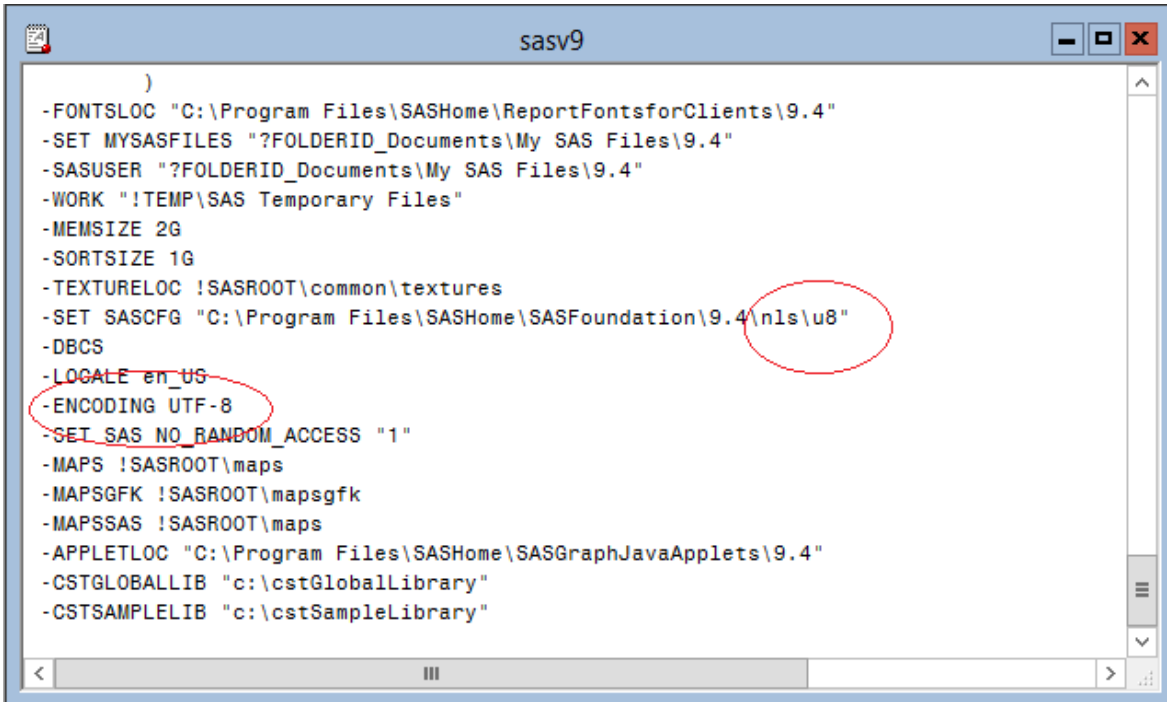
If you invoke SAS through a command line or a batch file, you have the option to specify the configuration file location through the CONFIG system option. For example, the code below invokes SAS using the Unicode version of the configuration file:

```
C:\Progra~1\SASHome\SASFou~1\9.4\sas.exe
            -sysin ProgramName.sas
            -config C:\Progra~1\SASHome\SASFou~1\9.4\nls\u8\SASV9.CFG
```

If you would like to review, update, or verify the settings in the configuration file, you can locate the sasv9.cfg files by navigating to the following directory:

```
C:\Program Files\SASHome\SASFoundation\9.4\nls\u8
```

You can then open and edit the sasv9.cfg file with the Windows Notepad or the SAS viewer. Display 2 below demonstrates some of the settings included in the configuration file for Unicode Support.



Display 2. A Partial Snapshot of the SAS9 NLS Configuration File for Unicode Support

For detailed instruction on invoking SAS with different language and encoding sessions on the Windows platform, refer to the SAS Technical Paper: Setting up SAS9 National Language Support in Microsoft Windows Operating Environments.

If you use the UNIX platform, SAS is invoked by Bourne Shell scripts. The invocation scripts are named using the language codes of the installed language. For example, sas_en invokes the English version, and sas_u8 invokes the Unicode version. Follow the steps below to update your session encoding:

1. Locate the invocation script under the following directory:

   ```
   !SASROOT/bin
   ```

2. Update the script to change the SAS session locale or encoding.

For detailed instruction on invoking SAS with different encoding sessions on the UNIX platform, refer to SAS Configuration Guide for SAS 9.4 Foundation for UNIX Environments.

To conclude, the encoding of the target session may not be able to accommodate special characters in the source data set. Sometimes it may be appropriate to suppress the transcoding error to finish the transfer. However, if the goal is to correctly display all characters in the source data, you should follow the instructions covered in this section to invoke your SAS session with an appropriate encoding value.

BUILDING A QUALITY DATA TRANSFER PROCESS

A quality process during data transfer takes a systematic approach to make sure quality standards are met. A complete quality process is composed of careful documentation and thorough validation of each step in program development. Self-validation during development makes sure the task was accomplished correctly in the first place. Independent validation can benefit the process with a fresh, second look. Automated validation with macros and other error-detecting utilities can search for systematic errors and inconsistencies efficiently. The extent and method of validation to be performed depends on a number of factors such as risk level, available resources, the format of the source data, and whether or not a source documentation is provided.

Key considerations for building a quality transfer process at the receiving end include the following:

1. An automatic log search utility checks SAS log files for warning and error messages. The SAS logs provide valuable information that helps you identify problems or potential pitfalls. The amount of information on the logs, however, can also be overwhelming. An automatic search utility not only improves efficiency. It also reduces the potential for human errors.
2. At contract research organizations, we often receive multiple rounds of data transfers over time. Data matures and becomes more complete at each new transfer. Performing quality checks only at the first round of transfer is not sufficient. As data accumulates, the programming that used to work for the first few transfers may not be able to accommodate future data transfers. For example, special characters may be entered at a later data transfer, causing transcoding errors and potential data loss. Therefore, programmers need to approach each round of transfer with the same level of care and attention to details.
3. If the delivering end provides documentation on the source data sets, we at the receiving end should take good advantage of the specifications provided in the documentation. A good source to check data structure and formats against data transfer specifications is in the PROC CONTENTS output. You can also take advantage of an automatic utility to compare the number of datasets delivered with the number of datasets successfully converted.
4. If the source data is in the format of SAS data sets, we will have the convenience of generating PROC CONTENTS output on the source data even though the source data uses a different representation. CEDA will be automatically invoked to access the source data that have a different source data representation.
5. If source documentation is lacking and source data are in not in the SAS data set format, we can consider comparing new transfers with previous transfers. For example, with a SAS utility, we can automatically compare the number of data sets, number of observations, data structure, and data formats to spot potential problems that could have prevented data from being converted completely and correctly.
6. Critical data points need to be checked as thoroughly as possible. Examples of critical data points are serious adverse events, adverse events of special interest, deaths, and disease progressions. Variables with very long comments or with many missing values are prone to error, and therefore may need a thorough second look.

To summarize, this section suggests that quality process is as important as quality coding. Although processes vary on a number of factors, there are helpful considerations we can take into account when building a quality data transfer process that fits our purposes.

CONCLUSION

Data are critical groundwork. Data transfers build the foundation of the work at the CROs. Incomplete transfers and incorrect data will not lead to right conclusions. Data transfer is not a simple process. The incompatibilities among computing environments complicate the matter. With little or no documentation from the delivering end, we as data recipients are often clouded by confusions about data format and representations. In this paper, we suggest that source data be thoughtfully examined before jumping into transfer procedures. We recognize that data truncation or loss due to transcoding is a most serious issue which must be addressed when moving data cross environments. We provide tips and example code for examining data and troubleshooting issues. We hope this discussion will inspire you into building a quality process for successful cross-environment SAS data transfers.

REFERENCES

Carlton, J. 2017. “SAS Blogs: Demystifying and Resolving Common Transcoding Problems.” Accessed March 6, 2018. https://blogs.sas.com/content/sgf/2017/05/19/demystifying-and-resolving-common-transcoding-problems/

Dutton, D. 2015. “Data Encoding: All Characters for All Countries.” *Proceedings of the PharmaSUG 2015 Conference*, Cary, NC: SAS Institute Inc. Available at https://www.lexjansen.com/phuse/2015/dh/DH03.pdf

SAS Institute Inc. 2014. *Usage Note 52716.* Cary, NC: SAS Institute Inc. Available at http://support.sas.com/kb/52/716.html

SAS Institute Inc. 2015. *Setting up SAS® 9 National Language Support in Microsoft Windows Operating Environments.* Cary, NC: SAS Institute Inc. Available at http://support.sas.com/techsup/technote/ts801.pdf

SAS Institute Inc. 2016. *Moving and Accessing SAS® 9.4 Files, Third Edition.* Cary, NC: SAS Institute Inc. Available at http://documentation.sas.com/?docsetId=movefile&docsetTarget=titlepage.htm&docsetVersion=9.4&locale=en

SAS Institute Inc. 2017. *Configuration Guide for SAS® 9.4 Foundation for UNIX Environments.* Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/installcenter/en/ikfdtnunxcg/66380/PDF/default/config.pdf#page=33

SAS Institute Inc. 2017. *Migrating Data to UTF-8 for SAS® Viya™ 3.3.* Cary, NC: SAS Institute Inc. Available at http://documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.1&docsetId=viyadatamig&docsetTarget=titlepage.htm&locale=en

SAS Institute Inc. *SAS Technical Paper: Multilingual Computing with SAS® 9.4.* Cary, NC: SAS Institute Inc. Available at https://support.sas.com/resources/papers/Multilingual_Computing_with_SAS_94.pdf

CONTACT INFORMATION

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

Yun (Julie) Zhuo
Axio Research, LLC
yunz@axioresearch.com