Paper 281-28

Multi-Lingual Computing with the SAS® 9.1 Unicode Server

Stephen Beatrous, SAS Institute Cary, NC

ABSTRACT

Business information today comes in many languages. Your customers and employees may come from all over the globe. Your mission-critical data is in more than one language. SAS offers several features that allow you to store and process multilingual data. This paper introduces Unicode support that is built into the 9.1 SAS system.

A LITTLE BACKGROUND

The SAS system since Version 5 is delivered in 2 flavors - the SBCS system and the DBCS system. The SBCS system handles character data in the ASCII and EBCDIC encodings. ASCII and EBCDIC store characters in a single byte. There are multiple versions of ASCII and EBCDIC to handle national characters in different regions. For example the Windows Latin1 encoding handles the characters necessary for the languages of Western Europe and the US. The Windows Latin 2 encoding handles the characters in the languages of Central and Eastern Europe.

The DBCS SAS system handles character encodings where individual characters take multiple bytes to represent. In the DBCS system an individual character could take one or more bytes. The code cannot assume that a byte contains a character. The DBCS system has been used by Japan, China, Korea, and Taiwan.

The SBCS system has been used to handle the languages where characters are represented as a single byte. The DBCS system has been used to handle languages where characters take up a varying number of bytes.
Both the DBCS and the SBCS SAS systems were designed so that an individual SAS session could represent and process characters within a region or country. That is, a SAS session can process Western European characters, or Eastern European characters, or Japanese Characters, or Chinese characters etc. Until 9.1, however, a SAS session could not represent characters from all of the above languages.

With SAS 9.1 SAS now supports a flavor of Unicode. Unicode provides a universal character set supporting the characters necessary for commerce in most of the languages spoken in the world today. With 9.1 it is now possible to write a SAS application which processes Japanese data, German data, and Polish data all in the same session. A single server can deliver multi-lingual data to users around the world.

The 9.1 SAS system uses the DBCS system to support a UTF-8 encoding. UTF-8 is a flavor of Unicode where characters are represented in 1 to 4 bytes. I will refer to the DBCS system running with a session encoding of UTF-8 as **the SAS Unicode server.**

This paper will discuss four use scenarios for the SAS Unicode server:

1. Universal Data Server
2. Universal Compute Server
3. Batch generation of Unicode HTML
4. Best Practices and Pitfalls of the SAS Unicode Server

STARTING A SAS UNICODE SERVER

There are two things you need to do to start a SAS Unicode Server:

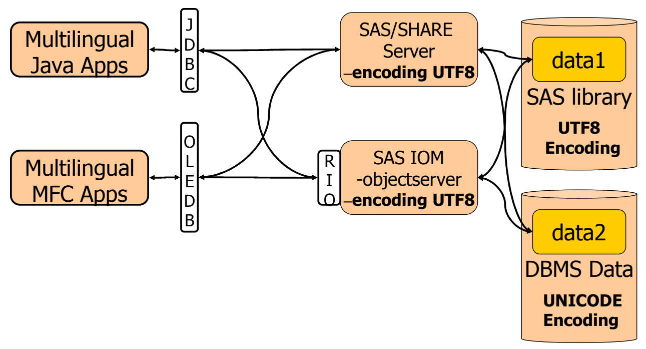
1. Install the DBCS/Unicode extensions of SAS on your system
2. Use the session option –ENCODING UTF8

It is that simple. The picture gets complicated when you start thinking about how to convert your systems to use the Unicode server efficiently.

When you start a SAS Unicode server the files being written will by default store characters in UTF8 encoding. If files in other encodings are read, they will automatically be converted to UTF8 format on input using SAS' CEDA feature. In the section of this paper titled "Best Practices" I will discuss how to efficiently use CEDA to bring legacy files into a Unicode format.

UNIVERSAL DATA SERVER

One way to exploit the SAS Unicode server is as a data server. The following diagram shows how Unicode clients can get multi-lingual data from a SAS server. Note that Java and MFC are already Unicode enabled environments. It is a natural fit to use a Unicode Java or MFC client to get to Unicode data from a SAS server.

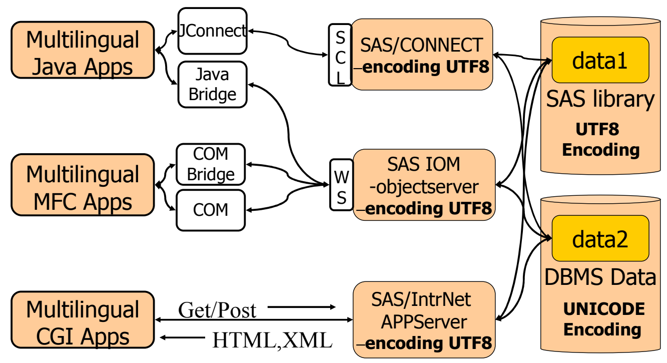


Note that the data sources can be SAS data files or DBMS data files. If the data on disk is in UNICODE format then there will be no loss of information when the data flows through the SAS Servers.

The SAS ODBC driver does not surface Unicode data from a SAS server. Also the SAS SPDS and V6 compatibility engines do not support storing Unicode data. This means that applications relying on ODBC or on SPDS or V6 storage will not be able to exploit a SAS Unicode server.

UNIVERSAL COMPUTE SERVER

The SAS system is often used as a compute server from a non-SAS client. This is another natural fit for the SAS Unicode Server. The following diagram shows the three ways that a Unicode Server may be used as a compute server from a Unicode enabled Java, MFC, or CGI.



BATCH GENERATION OF UNICODE HTML

The final way that a SAS Unicode server may be used is in a batch program producing ODS output. At the time of this writing the following ODS destinations support –encoding UTF8:

- HTML
- XML

The SAS Unicode server was used to produce the following report (with a simple proc print). Note that without the 9.1 Unicode server it would not have been possible to produce output with this rich set of National characters.

locale	language	langname
Arabic (Lebanon)	Arabic	العربية
Czech	Czech	čeština
English (United States)	English	English
German (Germany)	German	Deutsch
Japanese	Japanese	日本語

ACCESSING AND CREATING DATA

Data comes in 3 forms for the SAS system. Each must be looked at independently.

1. External files
2. SAS Data Libraries
3. DBMS Tables

EXTERNAL FILES

When an external file contains all character data that is in an encoding different from the session encoding then you must use the ENCODING= option (introduced in SAS 8.2) to force SAS to transcode the data on input. ENCODING= may be specified on the FILENAME, INFILE, and ODS statements. Please see the documentation on these statements for details on the ENCODING= option.

When an external file contains a mix of character and binary data then you must use the function to convert individual fields from the file encoding to the session encoding.

The KVCT function is of the form:

outstring = KVCT (instring, enc_in, enc_out);

Where,

instring - Input character string
enc_in - Encoding of instring
enc_out – Encoding of out string
outstring – Results of transcoding instring from enc_in to enc_out.

For example, if you have a wlatin1 string that needs to be conveted to UTF-8:

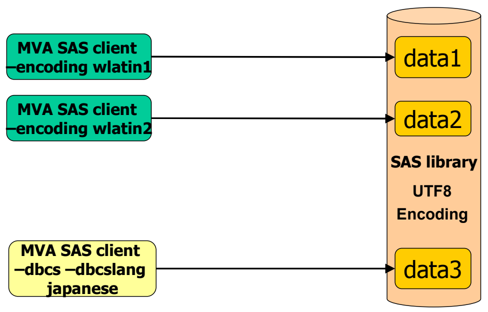
```
out = KVCT ( in,
                 “WLATIN1”,
                  “UTF-8”);
```

SAS DATA LIBRARIES

SAS DATA files have an ENCODING attribute in V9. The CEDA facility (see Base SAS documentation) will automatically transcode character data on input and on output when the file encoding is different from the session.

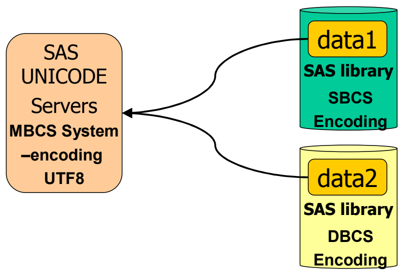
On output, SAS will by default create new files in the session encoding. The SBCS or DBCS session can, however, explicitly create a UTF8 data file. The ENCODING=UTF8 option and the OUTENCODING=UTF8 libname option are used to force any 9.1 SAS session to create a UTF8 encoded file.

CEDA (Transcoding on Output)


MVA SAS client –encoding wlatin1
MVA SAS client –encoding wlatin2
MVA SAS client –dbcs –dbcslang japanese
data1
data2
SAS library UTF8 Encoding
data3

CEDA can also be used to convert SBCS and traditional DBCS files into UTF8 data as the files are read.

CEDA (Transcoding on Input)


SAS UNICODE Servers MBCS System –encoding UTF8
data1
SAS library SBCS Encoding
data2
SAS library DBCS Encoding

BEST PRACTICES AND PITFALLS OF THE SAS UNICODE SERVER

WHAT FORMAT SHOULD MY DATA BE IN?
To make the most efficient use of a SAS Unicode compute or data server the data should be in Unicode format. By default files will be in an encoding which matches the encoding of the session which creates the file. Your legacy files will contain character data that is not in Unicode format. One of your first steps in converting an application to run in Unicode should be to convert the data files. As noted above files can be read by a Unicode server even if they are not in Unicode format. However, there is a performance cost (as character data is converted on input) and there are restrictions (files that are not in session encoding cannot be updated and cannot do index optimization).

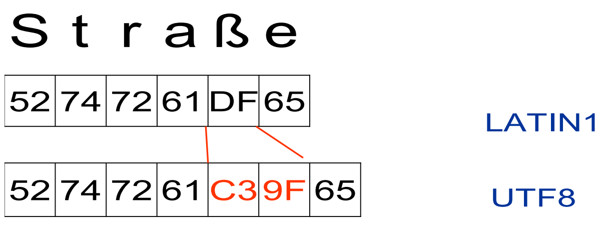
You should use the CVP engine described above to convert your files without truncation.

AVOIDING CHARACTER TRUNCATION DURING TRANSCODING
If you are using CEDA to transcode on input or to transcode on output you will encounter problems of character variable truncation. National characters will take a varying number of characters to represent in memory.

Bytes in UTF8	Character Sets
1	7bit, US_ASCII Characters
2	East, Central and Western European, Baltic, Greek, Turkish, Cyrillic, Hebrew, and Arabic

3	Japanese, Chinese, Korean, Thai, Indic and certain control characters
4	Some ancient Chinese, special Math symbols (surrogate pairs in UTF16)

Assume that you have a 6 byte character field with the value “Straße”. In memory the field will look like this:



If CEDA is used to read this field from a Latin1 encoding into a UTF8 Session encoding then the value will truncate to “Straß” (all that can be represented in a 6 byte UTF8 field). A new engine (CVP) has been added to V9.1 to automatically pad character lengths. Using the CVP engine allows you to transcode data to UTF8 without truncation. By default the CVP engine will double the length of the character fields on input. The program below will copy all of the input files from X to Y, doubling the length of all character fields and transcoding them to UTF8 along the way.

```
Libname x CVP ‘path1’ CVPENGINE=V9;
Libname u ‘path2’ outencoding=UTF8;
PROC COPY NOCLONE in=x out=u;
   SELECT datasetname;
  RUN;
```

SUPPRESS TRANSCODING OF BINARY DATA

Sometimes a data set will contain character fields that are really binary in nature. SAS would corrupt these fields if it transcoded them from file encoding to session encoding. In V9 you can flag fields as binary with the TRANSCODE=NO option.

For example, the MXG data set PDB.XTY70D contains many binary fields – e.g. CPUSER0. These fields will be incorrectly transcoded as character data if the file is processed with CEDA. The ATTRIB statement below will suppress any transcoding of the CPUSER0 field while allowing all other character fields to be transcoded.

```
 DATA PDB.XTY70D;
   ATTRIB CPUSER0 TRANSCODE=NO;
   SET PDB.XTY70D;
```

CODING ISSUES: USING THE K FUNCTIONS

If you do not currently use the DBCS SAS system then your SAS programs will assume that every character is a single byte in length. You will have to convert your SAS programs if you want them run with UTF8 data. The SAS character functions (for example SUBSTR, INDEX, LENGTH) have DBCS character equivalents (for example KSUBSTR, KINDEX, and KLENGTH).

Here is a simple example using two K functions. This example loops over the characters in a string it assumes that a character can be as much as 4 bytes in length:

```
DATA _NULL_ ;
     SET merged;
     LENGTH ch $ 4 ;
     DO I = 1 TO KLENGTH(maktx) ;
      CH = KSUBSTR(maktx, I, 1) ;
      PUT CH=;
     END ;
   RUN;
```

CONCLUSIONS

The Unicode SAS Server introduces a SAS system that can handle data from around the world in a single application. The 9.1 SAS Unicode Server allows you to meet your business need to capture and present national characters from around the world.

ACKNOWLEDGMENTS

Shin Kayano SAS Japan
Joji Koyabashi SAS Japan
Pranav Patel SAS Cary
Jack Wallace SAS Cary
Manfred Kiefer SAS EMEA

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
steve.beatrous@sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.