

# Ensuring Compatibility of Encoding across Different Versions of SAS

## Introduction

Beginning with SAS 8.2, Base SAS has provided separately encoded versions for various locales. It has also provided the NLSCOMPATMODE option to deal with legacy code. This option, however, might become obsolete in future releases. This document shows how to keep applications from being affected by these possible changes. It also describes ways to migrate legacy code.

## The problem with EBCDIC

Extended Binary Coded Decimal Interchange Code (EBCDIC) is a family of related encodings used by IBM. Unlike most 8-bit encodings, it is not compatible with ASCII. Rather, characters are coded according to the historical needs of punched card machines. Each EBCDIC variation, known as a code page, is identified by a Coded Character Set Identifier, or CCSID. The characters in the basic set (a-z, A-Z, 0-9, etc.) are mapped to the same code points on all EBCDIC code pages (these are called invariant characters). The rest of the code points can be used for different national and special characters (these are called variant characters), depending on which country and language a code page was developed for.

### *What are variant characters?*

The following characters are considered variant because they can have different code positions in various EBCDIC variations:

! # \$ @ \ [ ] ^ ` { } | ~

These characters exist in every encoding, but their hex values may change from one encoding to another, as shown in the following table:

Character	1047	838	870	1025	1141	1142	1143	1144	1145	1146	1147	1148
!	5A	5A	4F	4F	4F	4F	4F	4F	BB	5A	4F	4F
#	7B	7B	7B	7B	7B	4A	63	B1	69	7B	B1	7B
\$	5B	5B	5B	5B	5B	67	67	5B	5B	4A	5B	5B
@	7C	7C	7C	7C	B5	80	EC	B5	7C	7C	44	7C
\	E0	E0	E0	E0	EC	E0	71	48	E0	E0	48	E0
[	AD	49	4A	4A	63	9E	B5	90	4A	B1	90	4A
]	BD	59	5A	5A	FC	9F	9F	51	5A	BB	B5	5A

Character	1047	838	870	1025	1141	1142	1143	1144	1145	1146	1147	1148
^	5F	69	5F	5F	5F	5F	5F	5F	BA	BA	5F	5F
`	79	79	79	79	79	79	51	DD	79	79	A0	79
{	C0	C0	C0	C0	43	9C	43	44	C0	C0	51	C0
}	D0	D0	D0	D0	DC	47	47	54	D0	D0	54	D0
	4F	4F	6A	6A	BB	BB	BB	BB	4F	4F	BB	BB
~	A1	A1	A1	A1	DC	DC	DC	58	BD	BC	BD	A1

**Table 1: Variant EBCDIC characters**

The problem of EBCDIC encoding for SAS is that variant characters are part of the SAS language syntax itself. For example, the dollar sign (\$) is used for character formats, and the curly braces ({} ) or the square brackets ([]) are used for array statements.

In addition, the newline (\n) character is treated as a variant character, even though it is at the same code point in all EBCDIC encodings. This treatment results from the possible variations in transcoding from the newline character to the line feed character. For more details, see “What is the difference between Open Edition EBCDIC and traditional EBCDIC?”

### ***What is compiler encoding?***

SAS uses compiler encoding, which is the encoding used to compile the SAS system on your platform, for processing external data and SAS syntax. This compiler encoding is the same encoding used for U.S. English and is used if you do not change the locale or encoding of the SAS session. Before SAS 8.2, Base SAS used compiler encoding for all data processing.

## **The SAS 6 approach to encoding**

Variant characters appear in every encoding that SAS supports, but usually they occupy different code points on different EBCDIC code pages. For example, in Danish EBCDIC an “Å” occupies the same code point as the compiler encoding of a “\$” (0x5B). Hence in some contexts 0x5B must be treated as a “\$”, while in others it must be treated as an “Å”.

The SAS 6 solution was to try to do context-sensitive transcoding and to allow users to use alternate characters from their keyboards when, for example, they wanted to use an “Å” in place of a “\$” in a SAS program (see the following example). The SAS 6 transcoding often involved proprietary (or nonstandard) translate tables. This usage forced some (but not all) of the variants to transcode in some contexts.

For example, a Danish mainframe user wanted to enter the following SAS code fragment:

```
array entry{8} $ 23 _temporary_;
```

However, the user actually had to enter the following:

```
array entryæ8å Å 23 _temporary_;
```

This code fragment looks strange on the Danish terminal (or with a Danish terminal emulator setting), but the compiler processed these characters correctly because it recognized them as 0xC0 and 0xD0 for the array statement, and as 0x5B for the character format.

The SAS 6 solutions were not perfect. For example, porting an application to a different platform resulted in mapping problems. If 0xC0 is mapped to “{”, 0xD0 is mapped to “}”, and 0x5B is mapped to “\$”, then how could a Danish user use “æ”, “å” and “Å” in textual data?

The two-to-one mappings of variant characters created headaches throughout the life of SAS 6. For example, the PROC CPORT user needed to remember the source platform in order to determine the proper translation table to use with PROC CIMPORT.

The SAS 6 solution worked well enough for the SAS 6 product. The most important problems with context-sensitive transcodings have been fixed, and customers have learned to live with the others. Most of the context-sensitive transcodings were done in SAS software to SAS communications (for example, in SAS/SHARE or SAS/CONNECT client/server settings or in CPORT files). The fact that SAS software was on both ends of the communication clarified the context of many transcodings.

## The SAS 8.2 approach to encoding

SAS 8.2 introduced radical innovations, but also maintained continuity with the past.

SAS continued the implementation of session encoding (begun in SAS 8.1) and addressed the problems posed by variant characters.

The first requirement of the new solution was that it must run by default in a manner that was compatible with SAS 6. That is, if a user's SAS 6 program reads:

```
input x Åchar10;
```

then that program must continue to run in SAS 8. The SAS 8.2 approach enabled the customer to run in NLS compatibility mode so that existing applications could run unchanged. In other words, when the NLSCOMPATMODE (NLS compatibility mode) option is used, customers can run SAS production programs that they created prior to SAS 8.2 without needing to make any changes. In NLS compatibility mode, SAS uses compiler encoding, which is the encoding used to compile the SAS system on your platform, for processing external data and SAS syntax.

The SAS 6 solution was not adequate for SAS 8 and beyond, because the SAS 8 system was a more open system than its SAS 6 counterpart. In SAS 8, SAS contends with open clients such as Java, IOM, and SAS/IntrNet software. It is no longer reasonable for SAS to know the context of every transcoding or to rely on nonstandard translate tables.

A client has the choice of running the system with an explicit session encoding. In this case, the system uses an internal character representation that matches the client's native encodings. Variant characters that are part of the SAS language syntax itself are transcoded to the values of session encoding. For example, when a customer using the Finnish EBCDIC encoding (code page 1143) specifies the character “\$”, the code point 0x67 is transmitted to the SAS software.

Consequently, z/OS media was made available in the following encoded versions, which support multiple locales and regions. (See the *Configuration Guide—SAS 9.1.3 Foundation for z/OS* for the locale and region values):

- 838 (Thailand)
- 870 (Central Europe)
- 1025 (Russia)
- 1047 (United States)
- 1141 (Austria and Germany)
- 1142 (Denmark and Norway)
- 1143/1122 (Finland and Sweden)
- 1144 (Italy)
- 1145 (Spain)
- 1146 (United Kingdom)
- 1147 (France)
- 1148/1130 (International)

SAS is distributed in the preceding encoding support groupings, which usually support a single encoding and a group of related locales. SAS system installation media VOLSERS and certain installed SAS system filenames (such as SASHELP) contain a two-character code that identifies the encoding and locale(s) supported—for example, W3 for Austria and Germany (CP1141) or WB for International (CP1148). See the following table and “Languages, Encodings and Installation Codes” in the Appendix of the *Installation Instructions—SAS 9.1.3 Foundation for z/OS* for further detail. User data files should match these encodings.

Encode value	Country	Main encoding	Default locale	Alternate locale(s)
C0	Poland	OPEN_ED-870	Polish_Poland	Croatian_Croatia Czech_CzechRepublic Hungarian_Hungary Romanian_Romania Slovak_Slovakia Slovenian_Slovenia
F0	Thailand	OPEN_ED-838	Thai_Thailand	
R0	Russia	OPEN_ED-	Russian_Russia	Bulgarian_Bulgaria

Encode value	Country	Main encoding	Default locale	Alternate locale(s)
		1025		Ukrainian_Ukraine
W0	United States	OPEN_ED-1047	English_UnitedStates	English_Australia English_Canada English_Jamaica English_NewZealand English_SouthAfrica Dutch_Netherlands French_Canada Icelandic_Iceland Portuguese_Portugal Spanish_Mexico Spanish_UnitedStates
W3	Germany	OPEN_ED-1141	German_Germany	German_Austria
W5	Denmark	OPEN_ED-1142	Danish_Denmark	Norwegian_Norway
W6	Finland	OPEN_ED-1143	Finnish_Finland	Swedish_Sweden
W7	Italy	OPEN_ED-1144	Italian_Italy	
W8	Spain	OPEN_ED-1145	Spanish_Spain	
W9	United Kingdom	OPEN_ED-1146	English_UnitedKingdom	English_Hongkong English_Ireland
WA	France	OPEN_ED-1147	French_France	French_Luxembourg
WB	Belgium	OPEN_ED-1148	French_Belgium	Dutch_Belgium French_Switzerland German_Switzerland Italian_Switzerland

**Table 2: Encodings and installation code**

SAS software had to use standard encodings to effectively exchange data with other software. Ambiguity of code points (such as that between “Å” and “\$”) had to be removed in the long term. The SAS 8.2 approach was a first step to achieving this goal. SAS expected most SAS 8 customers to run in NLS compatibility mode. Over time, however, SAS expected the norm to be standard session encoding that matches the client's OS encoding. The system options NLSCOMPATMODE and NONLSCOMPATMODE were supposed to help to make the transition as smooth as possible.

The NONLSCOMPATMODE option turns off NLS compatibility mode. When the NONLSCOMPATMODE option is used, all character data is processed by using the ENCODING option encoding, including external data, SAS syntax, and user data. In this mode, the encoding that SAS uses to process character data is the encoding set by the ENCODING or LOCALE option. It has become the default in SAS®9.

## Problems with mixed encodings

Imagine you're generating HTML output on the mainframe to be displayed by a Web browser on your Windows PC.

To generate the HTML output, you are running Base SAS using compiler encoding, but session encoding is used for your data. Let's further imagine that you want to create a short table of the 12 host cities of the Football World Cup 2006, their number of inhabitants, their states ("Bundesland"), and the corresponding UN Location codes. Your code might look something like the following:

```
data cities (label="Die zwölf WM-Städte");
input CNAME $1-20 POP 21-27 STATE $28-49 CODE $50-54;
label CNAME='Stadt';
label POP='Einwohner';
label STATE='Bundesland';
label CODE='UN/LOCODE';
datalines;
Berlin          3387828Berlin          DEBER
Dortmund        588680 Nordrhein-Westfalen  DEDTM
Frankfurt (Main) 646889 Hessen             DEFRA
Gelsenkirchen   270107 Nordrhein-Westfalen  DEGEK
Hamburg         1734830Hamburg            DEHAM
Hannover        515841 Niedersachsen       DEHAJ
Kaiserslautern  99182 Rheinland-Pfalz      DEKLT
Köln            969709 Nordrhein-Westfalen  DECGN
Leipzig         498491 Sachsen             DELEJ
München         1249176Bayern             DEMUC
Nürnberg       495302 Bayern             DENUE
Stuttgart       590657 Baden-Württemberg     DESTR
run;
```

The HTML output is created with ODS, as follows:

```
filename odsout '<your output file>';
ods listing close;
ods html file=odsout;
title 'Die zwölf WM-Städte';
proc print data=cities label noobs; run;

ods _all_ close;
ods listing;
title;
```

The resulting HTML output file on the mainframe (displayed with your browser setting as "Austrian ECECP (1141)", will look like this:

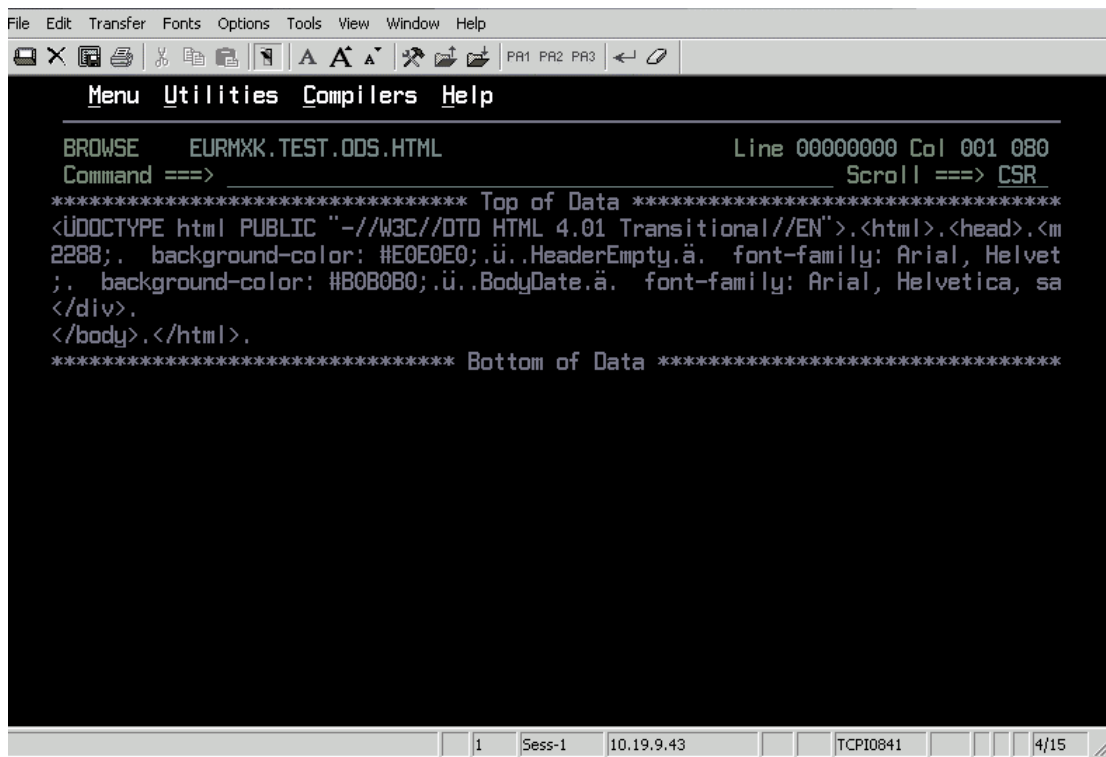


Figure 1: ODS output file on z/OS

Let's have a closer look at the document type declaration:

```
<ÜDOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

Where did the “Ü” come from? With compiler encoding (the only encoding available before SAS 8.2), the “!” is always stored as 0x5a. How the national characters such as “ä”, “ö”, “ü” and the like are stored depends on your native encoding. For instance, with CP1141 (Austrian/German) the “ä” is 0xC0, the “ö” is 0x6a, and the “ü” is 0xD0. Hence part of the table that you created looks like this:

```
<td class="l Data">Köln</td>.<td class="r,Data">,9
```

For the Web browser, this needs to be transcoded from EBCDIC to Windows Latin1 (windows-1252).

When your LOCALE= option is set to German, the transcoding would proceed as follows: 0x5a → 0xdc, 0xc0 → 0xe4. Because 0xdc displays as “Ü”, the HTML would still look like this:

```
<ÜDOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

Thus the HTML would still fail. Setting the LOCALE= option to German is necessary, however, to have the national characters such as “ä”, “ö” and “ü” in the data transcoded correctly. This means there is a dilemma: you can generate the HTML code correctly, or you can have the data transcoded correctly, but not both. This is a classic example of the problems with two-to-one mappings mentioned previously.

If the code was generated using a German-encoded SAS version, the “!” would be stored as 0x4f and the “ä” as 0xC0. Thus the transcoding would be correct. Using an encoded z/OS version for CP1141 (Austrian/German) results in generating correct HTML code for German. The same is true when the appropriately encoded version is used for other locales.

## Issues when migrating legacy code

Let’s have a look at another example. Generations of mainframe programmers in the UK have been using “£” as a replacement for “\$”. This worked fine in programs like this:

```
data mylib.employee;
input name fchar8. +2 income comma6.;
datalines;
Peterson 21,000
Morgan 17,132
;
run;
```

It worked because compiler encoding was used for *all* data processing, and the hex value for the characters “\$” and “£” are identical (0x5b), whereas the display depends on the terminal emulator settings.

To make the transition transparent from SAS 6.09E to SAS 8.2 with different encodings, the NLSCOMPATMODE (NLS compatibility mode) option was introduced. With this option, you can run SAS production programs that you created prior to SAS 8.2 without needing to make any changes. In NLS compatibility mode, SAS uses compiler encoding for SAS syntax. The NLSCOMPATMODE option was the default in SAS 8.2.

SAS<sup>®</sup>9 is designed to move away from NLS compatibility mode, so that customers use encodings that are appropriate for them. From the UK perspective, this means recognizing “£” as a character in its own right (£80.00), but not when used to identify a character format.

So in SAS<sup>®</sup>9, the NONLSCOMPATMODE option is the default, with the TRANTAB= and ENCODING= options set appropriately.

The NONLSCOMPATMODE option does *not* use compiler encoding, in which “£” is recognized as a character format, but uses the scanner-character-classification table instead. This means that legacy code (such as the code in the preceding example written in SAS 6.09E) will *not* run any more. To enable the code to run, you have to replace the “£” character (0x5b) with a “\$” character from session encoding (0x4a in this case, from EBCDIC 1146), because “\$” is the only recognized format identifier (see the following “Migration Aids” section).

There is one important exception: variant characters that appear in *physical OS data set names* (such as data set names, volume serial number names, or SMS data/storage/management class names) must be specified in *compiler* encoding. This

is because the interfaces to MVS data set services are defined in terms of code points, not characters; the interfaces perform no transcoding. For example, for the 8th character in the data set name `userid.$myfile`, a British customer would specify the character “£” (which is at code point 0x5B in the English-UK 1146 code page) to reference the data set `userid.$myfile`.

## What is the difference between Open Edition EBCDIC and traditional EBCDIC?

The characters that are used to explicitly indicate line boundaries are represented differently on different platforms, but they also show ambiguous behavior even on the same platform.

Software that runs under ASCII operating environments requires the end of the line to be specified by the line feed character. When data is transferred from z/OS to a machine that supports ASCII encodings, formatting problems can occur, particularly in HTML output, because the EBCDIC newline character is not recognized. SAS supports two sets of EBCDIC-based encodings for z/OS.

The encodings that have EBCDIC in their names (for example, EBCDIC1141 or EBCDIC1144) use the traditional mapping of the EBCDIC line feed character to the ASCII line feed character. This mapping can cause data to appear as one stream. The `ENCODING` option is set to the traditional EBCDIC encoding for the locale for NLS compatibility mode (`NLSCOMPATMODE`).

The Open Edition EBCDIC encodings have `OPEN_ED` in their names (for example, `OPEN_ED-1141` or `OPEN_ED-1144`). These encodings use the line feed character as the end-of-line character. When the data is transferred to an operating environment that uses ASCII, the EBCDIC newline character maps to an ASCII line feed character. This mapping enables ASCII applications to interpret the end-of-line correctly, resulting in better formatting. The `ENCODING` option is set to an Open Edition version of the EBCDIC encoding when you are running SAS with NLS compatibility mode turned off (`NONLSCOMPATMODE`).

## Hints and Tips

- SCL Code compiled by using the `NLSCOMPATMODE` option still runs when the `NONLSCOMPATMODE` option is used. If you need to make changes, however, you need to transcode the source code and recompile it using the `NONLSCOMPATMODE` option.
- Customized menus associated with an `FSEDIT` window need to be recompiled with variant characters in session encoding and when the `NONLSCOMPATMODE` option is used.

- The NLSCOMPATMODE option might affect the format of outputs that are produced using ODS. If you are using ODS, set the option value to NONLSCOMPATMODE or run with an Open Edition session encoding (even if the NLSCOMPATMODE option is used). Open Edition encodings are identical to the traditional EBCDIC encodings, except that they use the 0x15 code position for the newline character.

## Migration Aids

### *Catalog and data sets*

Of course, the easiest way to migrate a catalog or a data set is to use PROC MIGRATE. When remote library services are not used, migrated data files take on the data representation and encoding attributes of the library, and data sets take on the attributes of the host on which the target library is located.

Use PROC CPORT, create a transport file, and then use PROC CIMPORT on the catalog or data set. Imported data files take on the data representation and encoding attributes of the SAS session they are imported to. For external files you can use the same method, or you can use the ENCODING= option with the FILENAME statement.

Unfortunately, this method does not allow doing context-sensitive transcoding. So if you have mixed data and code, you might have to change national characters in textual data manually or programmatically (see the following section).

### *Raw source code*

Read in code with the ENCODING= option to convert to session encoding. Of course, this method cannot do context-sensitive transcoding. This means that if you have code with embedded text and comments (such as sequential input in a DATALINES statement as in the following example), all characters will be transcoded from one EBCDIC code page to another.

Imagine you have stored source code to create an employee table with Swedish names in a PDS member called “mydata.source(employee)”:

```
data employee;
input name Åchar9. Ö10 income commax6.;
datalines;
Bergman 21.000
Holmberg 17.132
Lindgren 22.500
Sandström18.000
Åkerlund 22.000
Öst      19.800
;
```

```
run;
```

Now you are running a SAS session with your LOCALE= option set to Swedish. This means your session encoding is EBCDIC1143.

After you specify `inc 'mydata.source(employee)' encoding='open_ed-1047'`, your code will be transcoded from EBCDIC1047 to EBCDIC1143 and will look like this:

```
data employee;
input name $char9. @10 income commax6.;
datalines;
Bergman 21.000
Holmberg 17.132
Lindgren 22.500
Sandstr|ml8.000
$kerlund 22.000
@st      19.800
;
run;
```

This means that you have to change the Swedish characters manually or do this programmatically. SAS Technical Support can provide you with a REXX procedure that can make this programmatic change. For more information, please contact SAS Institute Technical Support.

## Conclusion

In order to achieve a consistent behavior of applications across platforms, you should run SAS on z/OS using the NONLSCOMPATMODE option with a version that matches your OS encoding. You should migrate legacy code by using one of the previously described methods.

## References

IBM (International Business Machines). 1993. 3174 Establishment Controller: *Character Set Reference. Configuration Support C, Release 3*. Manual no. GA27-3831-05. Sixth Edition. N.p.: IBM.

SAS Institute Inc. 2006. *Configuration Guide for SAS 9.1.3 Foundation for z/OS*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2006. *Installation Instructions for SAS 9.1.3 Foundation for z/OS*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2004. *SAS 9.1 National Language Support (NLS): User's Guide*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. TS-653 - *Globalization and National Language Support for Your Version 8.2 SAS Environment*. Cary, NC: SAS Institute Inc.

The ASCII/EBCDIC Character Set Task Force. 1989. SHARE Report SSD No.366: *ASCII and EBCDIC Character Set and Code Issues in Systems Application Architecture*. Edited by Edwin Hart. The Johns Hopkins University, Applied Physics Laboratory, Laurel, Maryland, USA. Chicago: Share Inc.

The Unicode Consortium. c2003. *The Unicode Standard, Version 4.0*. Edited by Joan Aliprand, et al. Boston: Addison-Wesley.