DBCS and NLS Support in the SAS System
Randal K. Whitehead, SAS Institute Inc., Cary, NC

ABSTRACT

This paper describes new and ongoing efforts to provide DBCS and NLS support within the SAS System. Areas covered include expanded translate table support allowing users to provide tables for upper and lower casing of data, local-to-transport and transport-to-local processing, interpretation of local national characters as SAS special characters (u-umlaut for right-curly-brace, etc.), and others.

INTRODUCTION

The term “National Language Support” or NLS refers to features provided to accommodate the variety of languages, formats for monetary, date, and numeric data, and sorting concerns encountered when software is developed for an international market.

In addition to the problems caused by differences in language and social convention, the greatest difficulties are caused by conflicts in the sets of characters used by different languages and cultures. A standard character set, usually a variation on the ASCII or EBCDIC character sets, provides for the use of up to 256 characters. Unfortunately, there are more than 256 alphabetic and special characters in use in the world. This leads to the replacement of one character with another for use in a particular country. This practice is often referred to as “mapping” and will be a focus of many of the examples in this paper.

An even more challenging character set problem is encountered when dealing with languages that use a fundamentally different system of writing. Of greatest interest are the Asian languages that use ideographic characters, in which each character represents a thing or idea, rather than a phonetic alphabet as in western languages. Japanese and Korean both use Chinese characters, called Kanji in Japan and Hanja in Korea. Because these ideographic languages make use of a far greater number of characters—Japanese uses at least 2000 and traditional Chinese up to 14,000—it is necessary to use two bytes for each character. Such character sets are referred to as Double Byte Character Sets or DBCS. Figure 1 illustrates the single and double byte codes.

<table>
<thead>
<tr>
<th>Character</th>
<th>Single Byte</th>
<th>Double Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>(EBCDIC)</td>
<td>X'42c1'</td>
<td>X'455b'</td>
</tr>
<tr>
<td>(IBM Mainframe)</td>
<td>X'c1'</td>
<td>X'c2e7'</td>
</tr>
<tr>
<td>(IBM Mainframe)</td>
<td>X'91e5'</td>
<td>X'4267'</td>
</tr>
</tbody>
</table>

Figure 1

DBCS processing presents problems far more difficult than simply mapping one character to another. The way character strings are entered, stored and processed must be fundamentally changed.

Before Chinese, Japanese, or Korean can be used with SAS software, the necessary hardware and system software support must be in place. You cannot use these languages on an English or other European language system. All major hardware vendors offer computer systems to support DBCS data, but availability may be limited outside of Asia.

The methods of entering DBCS data on such systems will vary, but the most common are phonetically based. In such a system the user enters a pronunciation of a Chinese character, often using English letters. The system then provides a list of similar-sounding characters, and the user chooses one.

Chinese, Japanese, and Korean are traditionally written vertically and right to left, but most software, including the SAS System, implements DBCS horizontally and left to right like English or other European languages. In general, only desktop publishing software supports the traditional writing style.

Additional complications come from the fact that no standard exists for DBCS; each hardware vendor is free to develop a new encoding for each new machine. Even for a particular machine there will be different encoding schemes for Japanese, traditional or complex Chinese simplified Chinese, and Korean.

DBCS data must also be intertwined with single-byte data. On mainframe systems that use the EBCDIC character set this requires the use of special escape codes, called Shift Out/Shift In or SO/SI. These codes originated in the need to shift from the standard EBCDIC print train to a DBCS print train on old style printers. SO/Sl, which are usually one byte each, surround the DBCS string, padding it on each side. Minicomputers, workstations, and PCs, which do not use the EBCDIC character set, do not use SO/Sl. The addition of SO/Sl can cause problems when DBCS data are transferred between systems.

All of these issues fall under the heading of “National Language Support” (with a special subheading for DBCS), and it is these issues that must be addressed to meet the needs of software users in an international market. Through features supporting cross-system porting of data and applications, DBCS data entry and presentation, and alternative formats and informats, SAS Institute has made a commitment to meeting those needs.

AREAS OF SAS SYSTEM SUPPORT

SAS SYSTEM OPTIONS

To control the NLS and DBCS features of SAS software, a number of system options are provided. These include options specifying the use of translate tables (which are discussed in detail later), and the selection of the DBCS encoding and language to use within the SAS session. The DBCS support options, which must be set at SAS System invocation, are DBCS/NODBCS, DBCSTYPE=, and DBCSFLANG=.

DBCSES/NODBCS DBCS enables general DBCS support throughout the SAS System.

DBCSFLANG= DBCSTYPE= specifies the DBCS encoding to be used. Because no standards exist for DBCS encoding, the same character will have different hexadecimal codes under different operating systems, as shown in Figure 2.

<table>
<thead>
<tr>
<th>Character</th>
<th>Single Byte</th>
<th>Double Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>(IBM Mainframe)</td>
<td>X'455b'</td>
<td>X'c1'</td>
</tr>
<tr>
<td>(IBM Mainframe)</td>
<td>X'91e5'</td>
<td>X'c2e7'</td>
</tr>
</tbody>
</table>

Figure 2
The value chosen for DBCSTYPE = will depend upon the environment in which the SAS System is executing. Some of the possible values are shown in the following table:

Table 1

<table>
<thead>
<tr>
<th>Platform</th>
<th>DBCSTYPE = Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mainframe:</td>
<td></td>
</tr>
<tr>
<td>IBM</td>
<td></td>
</tr>
<tr>
<td>FUJITSU (or FACOM)</td>
<td></td>
</tr>
<tr>
<td>HITACHI (or HITAC)</td>
<td></td>
</tr>
<tr>
<td>minicomputers:</td>
<td></td>
</tr>
<tr>
<td>DEC and DG</td>
<td></td>
</tr>
<tr>
<td>UNIX systems:</td>
<td></td>
</tr>
<tr>
<td>EUC (for Sun, MIPS, NeXT, and SR10)</td>
<td></td>
</tr>
<tr>
<td>DEC (for ULTRIX)</td>
<td></td>
</tr>
<tr>
<td>DG (for DG/UX)</td>
<td></td>
</tr>
<tr>
<td>HP15 (for HP-UX)</td>
<td></td>
</tr>
<tr>
<td>PoIBM or SJIS (for AIX/6000)</td>
<td></td>
</tr>
<tr>
<td>Windows, OS/2</td>
<td></td>
</tr>
<tr>
<td>PCIBM (for IBM PCs)</td>
<td></td>
</tr>
<tr>
<td>PCFCM (for Fujitsu PCs)</td>
<td></td>
</tr>
<tr>
<td>PCHTC (for Hitachi PCs)</td>
<td></td>
</tr>
<tr>
<td>POMS (for PCs using Microsoft encoding)</td>
<td></td>
</tr>
</tbody>
</table>

Refer to the SAS companion for each environment to determine the appropriate value for DBCSTYPE =.

DBCSLANG = DBCSLANG = specifies which DBCS language will be used. Possible values are:

- **JAPANESE** Japanese DBCS and single-byte English lowercase
- **KATAKANA** Japanese DBCS and single-byte Katakana (a Japanese alphabet)
- **KOREAN** Korean Hanja and English lowercase
- **HANGUL** Korean Hanja and Hangul (the Korean syllabary)
- **CHINESE** Simplified Chinese
- **TAIWANESE** Traditional or complex Chinese

The primary differences between DBCS languages are the starting and ending hexadecimal code points for DBCS character sets. For example, on IBM mainframe systems, DBCS Japanese begins at '4140'x and extends to '88FF'x, and DBCS Chinese begins at '4150'x, stops at '46FF'x, resumes at '4840'x, and ends at '6FFF'x.

There is no standard encoding scheme for Chinese characters even among the DBCS languages supported by a particular hardware vendor. Thus, the same character for "big" on IBM mainframe systems will have a different hexadecimal value depending on which DBCS language is in use, as shown in Figure 3.

**EXAMPLES** Here are examples of these options used in various environments:

```
options dcb
   / dbctype=IBM /* turn on DBCS */
   / dcbtype=JAPANESE; /* using Japanese Kanji */
```

```
options dcb
   / dbctype=HP15 /* HP UNIX environment */
   / dcbtype=KOREAN; /* using Korean Hanja */
```

```
options dcb
   / dbctype=DEC /* DEC VMS/ULTRIX environment */
   / dcbtype=TAIWANESE; /* using traditional Chinese */
```

**CONNECTIVITY AND TRANSLATION**

The ability to move data and applications between systems is an important feature of the SAS System. This support extends to the transfer of national language and DBCS data through the use of translate tables and DBCS support in products such as SAS/CONNECT* and SAS/ACCESS· software.

**SPECIFYING TRANSLATE TABLES**

Many of the features used for transferring data between hosts using different character sets make use of tables to control the translation from one character to another. If both hosts are using a standard English character set such as ASCII or EBCDIC there will be no problem. If a language other than English is in use, however, the standard translation may fail. In such a situation, custom translate tables must be specified. There are also standard translate tables that allow SAS software to correctly use characters in character sets other than standard ASCII or EBCDIC.

Beginning with Release 6.07 of the SAS System, you can use the TRANTAB= option to specify replacements for the standard translate tables. Tables are specified in a parenthetical list with ten positions, each corresponding to different standard table. The positions, default table names, and types of translation are as follows:

Table 2

<table>
<thead>
<tr>
<th>Position</th>
<th>Default</th>
<th>Type of Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SASXPT</td>
<td>local-to-transport-format</td>
</tr>
<tr>
<td>2</td>
<td>SASLCL</td>
<td>transport-to-local-format</td>
</tr>
<tr>
<td>3</td>
<td>SASUCS</td>
<td>lowercase-to-uppercase</td>
</tr>
<tr>
<td>4</td>
<td>SASLCS</td>
<td>uppercase-to-lowercase</td>
</tr>
<tr>
<td>5</td>
<td>SASCCL</td>
<td>character classification</td>
</tr>
<tr>
<td>6</td>
<td>*</td>
<td>scanner translation</td>
</tr>
<tr>
<td>7</td>
<td>*</td>
<td>delta characters</td>
</tr>
<tr>
<td>8</td>
<td>*</td>
<td>scanner character classification</td>
</tr>
<tr>
<td>9</td>
<td>**</td>
<td>DBCS system table</td>
</tr>
<tr>
<td>10</td>
<td>**</td>
<td>DBCS user table</td>
</tr>
</tbody>
</table>

* indicates that the default table is not generally accessible.
** indicates that the default table is specified by the DBCSTYPE. system option.

If a null entry is specified then the table corresponding to that position will not be changed. For example the following statement would specify a custom lowercase-to-uppercase table:

```
options trantab=lowercase-table;
```
The other tables remain unchanged. Submitting another 
`options trantab(,upercase-table);`
`options trantab(,lowercase-table);`
is equivalent to
`options trantab(,lowercase-table,upercase-table);
although the OPTIONS procedure will only reflect the last 
TRANTAB option submitted.

**CREATING TRANSLATE TABLES**

There are two SAS procedures provided for the creation of translate tables. One, PROC TRANTAB, for single-byte character sets and, the other, PROC DBCSTAB, for DBCS data. Both allow you to 
create a new table from an existing table.

**PROC TRANTAB** To create a translate table with PROC TRANTAB you first specify which table you want to modify.

`proc trantab tables=sasucs nls;`
The NLS option tells PROC TRANTAB that the table is one of the 
standard tables, in this example the standard lowercase-to­
upercase table. You then list the modifications to the values in that 
table with the REPLACE command.

`replace 'DO'x 'a'; replace 'DO'x 'A';`
other translations

Finally, you use the SAVE command to save the modified table.

`save table=sasucse;`

**PROC DBCSTAB** If you are using a DBCS encoding that is not 
supported by the DBCSTYPE option, or that uses different standard 
translate tables, you can use PROC DBCSTAB to modify existing 
DBCSTABLE translate tables. For example 

`proc dbcstab;`
`/* name of new translate table*/
`name=custom;`
`/* based on KP encoding*/
`base=Kp15;`
`/* data to create new table*/
`data-trantab;`
`/* Korean language*/
`dbcs=ko;`
`/* catalog description*/
`desc='Modified Korean Trantab';`
`/* where the table is stored*/
`catalog='trantab.dcr';`
`/* checks for invalid DBCS in the new data*/
`verify;`
The TRANTAB data set is created by the DATA step.

`data trantab;`
`hp='6189'; dec='9803'; pcim='AG2';`
`run;`
with one observation for each value to be modified.

The custom translate table is then specified by the TRANTAB option.

`options trantab(,custom);`
The translate table is used for DBCS conversion with 
SAS/CONNECT software, PROC CPOR1 and PROC CIMPORT, 
and the DATA step function KCVT.

**USING TRANSLATE TABLES**

An important problem in porting applications across systems is that 
some of the national characters added to character sets are mapped 
to the same hexadecimal values as characters in the standard char­
acter set, but this mapping is not consistent between character sets. 

For example, on the PC the German umlaut (ü) is given a hex value 
of 81 which does not map to any standard character, but in the Ger­
man EBCDIC character set it maps to the same hex value, D0, as 
the right curly brace (}) which is used in ARRAY statements in the 
DATA step. This leads to a two-to-one mapping of characters in the 
PC character set, 'ü' at '81'x and '}' at 'D0'x, to one character in 
the German EBCDIC character set, 'ü' at 'D0'x. Porting SAS state­
ments containing special characters and national characters can 
result in syntax errors or the loss of national character data.

On the PC, an array statement might look like this:

`array a[1:1]='Johannes';`
and the hexadecimal representations of the characters in question 
('J', 'ü') are 7F, 7D, and 81 respectively. When this statement is 
ported to the MVS operating system (using German translate tables) 
it looks like this:

`array a[1:1]='Johannes';`
and the hexadecimal representations are E0, 7D, and D0. While this may 
look like a syntax error, it is not because the SAS System is created 
using that standard EBCDIC character set in which 'E0'x and 'D0'x 
correspond to ('J') and ('ü'). It may look strange, but it works.

The serious problems begin when this same statement is ported 
back to the PC. The question is whether the hex value D0 should 
be translated to '7D'x ('}') or to '81'x ('ü').

If the national characters are translated to national characters, the 
quoted string is preserved, but the statement will generate syntax 
errors because the special characters are lost:

`array a[1:1]='Johannes';`
If the national characters are translated to special characters, the 
statement will run correctly, but the national language data will 
be lost.

Because it is important to preserve national language data, solu­
tions to this problem will focus on changing how special characters 
are handled.

The first solution is to use a table that specifies translations to be 
made when a character is scanned by the SAS syntax scanner, but not when it is displayed. The sixth parameter to the TRANTAB= 
option specifies a scanner translate table that fills this need. In the 
ARRAY statement example on the PC, the national characters 'ü' 
and 'ü' should be scanned as 'ü' and 'ü' respectively, but displayed 
as 'ü' and 'ü'. To do this the hexadecimal values '81'x and '81'x 
should be translated for syntax scanning purposes only to '7D'x and 
'7D'x. A translate table to accomplish this can be created with 
PROC TRANTAB. Because the default table is not accessible, the 
table must be initialized with each character translated to itself. 
The characters to be translated are then replaced. For example 

`proc trantab table=trantab;`
rep '000000203204050670590A0B0C0D0E0';
'151115331545161718191A1C1D1F';
'202233333435363738393A3B3D3E3F';
'404142434445464748494A4B4C4D4E4F';
'505152535455565758595A5B5C5D5E5F';
'606162636465666768696A6B6C6D6E6F';
'707172737475767778797A7B7C7D7E7F';
'898A8B8C8D8E8F909192939495969798999A9B9C9D9E9F';

574
DBCS CROSS-SYSTEM SUPPORT

In addition to the translation support for using alternative character sets, the SAS System provides capabilities for transporting and converting DBCS data. These capabilities are provided within products such as SAS/CONNECT and SAS/ACCESS software and additional services such as the KCVT function in the DATA step which will be described later.

CPORT AND CIMPORT PROCEDURES

In addition to the KCVT function, the SAS System provides cross-system DBCS conversion through the CPORT and CIMPORT procedures. For example, a SAS/AF® application may be converted from DEC DBCS to IBM DBCS:
```
proc cport catalgamy .lib. afsys .catalog
  intypeadec
cuttype .. ibm;
run;
```

One potential problem to be aware of when porting windowing applications from a system that does not use SO/SI to one that does is truncation of the display. The reason for this is that each DBCS string may be expanded by at least two bytes: one for SO and one for SI. By default, available blank spaces will be used for the SO/SI, but if none are available the right side of the display will be truncated. It is important to allow room for the addition of SO/SI when designing DBCS applications on non-SO/SI systems if the applications will be ported to a mainframe system.

CULTURAL FORMATS

There are cultural differences not only in the way data are stored and processed but in how they are presented as well. In particular, there are differences in the way data are sorted.

SAS/CONNECT SOFTWARE

SAS/CONNECT software can automatically convert data sets and catalogs containing DBCS when uploading and downloading data. It also correctly handles DBCS characters that are remotely submitted to a host and returned to the local level for display. Translate tables based on the DBCS TYPE= option are used, and proper SO/SI are added if needed.

The same truncation problem described above can also occur with SAS/CONNECT software.

SAS/ACCESS SOFTWARE

SAS/ACCESS software provides support for DBCS data. With the SAS/ACCESS interfaces to DB2® and SQL/DS®, DBCS is supported through the GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC data types. On other databases, DBCS is supported as standard character data.

The SQL procedure supports DBCS on some systems, those names must be changed to valid SAS names.

The SOL procedure supports DBCS queries directly and through SQL pass-through:
```
proc sol;
  select * from connection to db2
    select * from db2. customer
    where address contains 'ABBA';
run;
```

The solution within the SAS System is to use PROC SORT with the SORTSEQ= option, which specifies an alternative collating sequence to be used for sorting. For example, the following will sort your character variables according to the Swedish convention. The SORTSEQ= option is also supported by PROC SQL for use with the ORDER BY clause. Consult the SAS Procedures Guide, Version 6, Third Edition for a list of Institute-supplied sorting tables.

If none of the Institute-supplied tables meets your needs, you can create your own using PROC TRANTAB.
DATE FORMATS

There are almost as many ways to display a date as there are countries in the world. These formats differ not only in the order of day, month, and year, but also in the form of the day and month and the delimiters between each element of the date. Here are a few examples:

- Bulgaria: 1993-X-3
- Canada: 93-10-03
- Germany: 3.10.1993
- Italy: 03/10/93
- USA: 10/03/93

The SAS System provides a number of standard date formats to present date data in a number of ways, including verbose forms such as the WEEKDATEw. format which prints a date as day-of-week, month-name dd, and yyyy or yyyy:

sunday, October 3, 1992

The WEEKDATXw. format is similar, but prints the day before the month name:

Sunday, 3 October, 1992

The standard date formats, including the NENGOW, format which prints Japanese dates, cover many of the common date forms.

The European headquarters of SAS Institute has developed additional date formats using SAS/TOOLKIT software for use with languages other than English. Currently there are additional European formats which are equivalent to the DATEw, DDMMYyw, WEEKDATXw, and WORDDATXw formats: EURDFDEw, EURDFDDw, EURDFXw, and EURDFWXw.

The language used by the European formats may be changed by applying a zap. Depending on what zap you apply, you get the equivalents of “Saturday, October 3, 1992” in Finnish, French, German, Italian, Spanish, and Swedish as shown below:

- Lauantasia, 3 lokakuuta 1992
- Samedi, 3 octobre 1992
- Samstag, 3 Oktober 1992
- Sabato, 3 Ottobre 1992
- Sabado, 3 de Octubre de 1992
- Sábado, 3 oktobre 1992

Please contact your local SAS Institute office for more information about these formats.

NUMERIC FORMATS

Numeric formats can also vary, commonly between English and other European communities, in the use of a period or comma to indicate decimals. The SAS System offers the COMMAw.d format, which prints numeric values with commas separating every three digits and a period separating the decimal fraction, and the COMMAxw.d format, which does it vice-versa as in this example:

```sas
proc print data=nums label; var commas commas; format commas commas15.2 commas commas15.2; title 'Print different numeric formats'; run;
```

### Output 1

<table>
<thead>
<tr>
<th>Print different numeric formats</th>
<th>COMMA.d</th>
<th>COMMAx.d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12,345.00</td>
<td>12,345.00</td>
</tr>
<tr>
<td></td>
<td>9.23</td>
<td>9.23</td>
</tr>
<tr>
<td></td>
<td>-12,345.00</td>
<td>-12,345.00</td>
</tr>
</tbody>
</table>

MONETARY FORMATS

To format monetary values, the SAS System offers the DOLLARw.d and DOLLARXw.d formats that print numeric data with a leading dollar sign. In the UK a leading pound sign (£) is seen due to character set differences.

If you need other monetary formats, you can create your own using the FORMAT procedure as seen in the following example:

```sas
proc format; picture dm low-c='0.000,000,000,000'; 0-high='0.000,000,000,000'; picture dx low-c='0.000,000,000,000' high='0.000,000,000,000'; run;
```

```sas
proc print data=curr noobs; var dollars dm francs dx escudos; format dollars dollar15.2 dm. francs fr. escudos es.; title 'Print Different Currency Formats'; run;
```

### Output 2

<table>
<thead>
<tr>
<th>Print Different Currency Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DOLLARS</strong></td>
</tr>
<tr>
<td>DOLLAR.DM</td>
</tr>
<tr>
<td><strong>DM</strong></td>
</tr>
<tr>
<td><strong>FRANCS</strong></td>
</tr>
<tr>
<td><strong>ESCUDOS</strong></td>
</tr>
<tr>
<td>$100.000.00 DM250.000.00 540.000.00 12.500.000.00 $00 Ese.</td>
</tr>
<tr>
<td>$150.000.00 DM270.000.00 810.000.00 F 18.750.000.00 $00 Ese.</td>
</tr>
</tbody>
</table>

OTHER THAN THE CHARACTER SET MAPPING PROBLEMS DESCRIBED ABOVE, THERE ARE FEW PROBLEMS IN THE DATA AND PROC STEPS WITH WESTERN LANGUAGES. DBCS, ON THE OTHER HAND, DOES REQUIRE SOME SPECIAL PROCESSING.

On the most basic level—text within quoted strings—you can use DBCS anywhere you can use single-byte text. This includes variable values, variable labels, data set labels, titles and footnotes. DBCS can also be used as input data, as range and label specifications in PROC FORMAT, and with the WHERE clause wherever quoted strings are allowed. When processing WHERE clauses the SOISI characters are ignored, making searches for embedded DBCS text possible.

One important restriction in Version 6 is that variable names cannot be in DBCS; they must conform to standard SAS naming conventions.
Here is an example of using DBCS in a SAS program:

```sas
* DBCS を用いたプログラムの例 *
data sales(label='売り上げデータ');
informat dept sales.;
label region='地区';
dep=dept; sales=売り上げ;
cards;
1 1 1200
1 2 340
2 1 830
2 2 1000
;
proc format;
value region 1='支店'
2='大阪'
3='名古屋';
run;
proc print label;
title '売り上げデータ';
format region region.;
run;
```

On mainframe systems that use SO/SP, the SO/SP are stored as part of the string when input with the $CHARn. informat, so the length of the character variable should be at least two bytes longer than the expected length of the character string.

### DATA STEP FUNCTIONS

Although DBCS data can be input or specified in quoted strings, the standard DATA step character functions will not handle DBCS data. To support DBCS data, a number of DBCS-enabled DATA step functions, all prefixed with the letter K, have been developed. KSUBSTR, KSTRCAT, and KTRUNCATE perform substringing, concatenation (the equivalent of the || concatenation operator), and truncation of DBCS strings correctly, as shown in this example:

```sas
data DUL;
  city='Tokyo' in DBCS /* get first chu of DBCS string */
  firstchar_k$substr( city, 1, 1);
polt firstchar;
run;
```

Using the standard SUBSTR function would have allowed the splitting of a DBCS character, which is undesirable.

The following DBCS-enabled functions are available and correspond to the standard DATA step functions:

- KCOMPRESS
- KSCAN
- KCOUNT
- KTRANSLATE
- KINDEX
- KSUBSTR
- KLEFT
- KUPDATe
- KLENGTH
- KTRIM
- KREVERS
- KVERIFY
- KRIGHt
- KUPCASE

Also provided is the KCVT function to perform cross-platform DBCS conversion, for example converting from IBM mainframe encoding to DEC encoding:

```sas
data _null_;
  file _null_;
  input string $80.;
  put destr(string='IBM', 'DEC');
r
run;
```

### DATA STEP FORMATS AND INFORMATS

To better handle DBCS data on SO/SP systems, several SAS formats and informats have been developed. The most useful of these are:

- **formats**
  - $KANJIX: adds SO/SP to DBCS
  - $KANJI: removes SO/SP from DBCS
  - $KANJIX: adds SO/SP to DBCS
  - $KANJI: removes SO/SP from DBCS

Since DBCS strings must have SO/SP to display correctly, these formats and informats are useful for storing large amounts of DBCS data without SO/SP.

### PROCEDURES

Just as the standard DATA step character functions can split a DBCS character, procedures that split character strings across lines can split DBCS characters in the process. This problem is most often seen with the FREQ, PRINT, REPORT, or TABULATE procedures.

To avoid undesirable character splitting, you should add spaces to labels or strings to move the split to a better location. You may also use the SPLIT= option with PROC REPORT or PROC TABULATE to force splitting in a specific location.

PROC REPORT is a particularly good procedure to use with DBCS data because it offers tremendous customization capabilities. The following example uses PROC REPORT and DBCS to produce the results shown in Output 3:

```sas
proc report data=sql union nowindow;
title '1989-90年度台湾、中南大陵、及香港地区';
title2 '進口貿易狀況分析報告';
col'country item year (import export net)'
import
export
net
z;
define country / group format=$char10. width=10 ' 地区';
define item / group format=$char10. width=10 '商品';
define year / group format=$char10. width=10 '年度';
define net / computed format=dollar16. width=16 '總出超全額';
define import / computed format=dollar16. width=16 '進口全額';
define export / computed format=dollar16. width=16 '出口全額';
define z / computed format=$1. width=1 special=0 ';
run;
```

577
When using PROC REPORT with DBCS data, you should be aware that the default splitting character, \'/\' or \'\'\', is used as part of many DBCS characters on mainframe systems. To avoid confusion you should choose a character other than \'/\'. The blank character, \'\'\'\'\'\', usually works without problems.

**WINOWING**

The DBCS features of the DATA step and procedures described so far are batch oriented. The SAS System does, however, provide DBCS support for windowing and interactive capabilities such as the SAS Display Manager System (DMS), SAS/FSP, SAS/Ap, SAS/CALC, SAS/INSIGHT, and SAS/EIS software.

The DBCSLANG= and DBCTYPE= options must be specified at SAS System invocation to use DBCS in interactive mode.

Figure 4 is an example of a windowing application developed using SAS/AF software. It was created just as an English-language application would be developed, only using DBCS input capabilities.

**GRAPHICS**

In graphics, there are a number of ways to produce output for DBCS languages. The first method is to use hardware fonts. If DBCS hardware fonts are available they may be accessed by specifying F=NONE.

Also, some mainframe operating systems such as MVS and MSP (by Fujitsu, Inc.) provide vendor-supplied software fonts for products such as GDDM and GSP. You must specify the appropriate device driver and chartype=2 in the GOPTIONS statement to access the graphics library’s software fonts.

Finally, SAS Institute provides a number of software fonts in various Asian languages: for Japanese, KANJI and MINCHO; for Korean, HANGUL; and for traditional Chinese, KAI and MING. Users specify the appropriate font, as seen in Figure 5.
BIBLIOGRAPHY

Richardson, L., and Kayano S. "SAS System Support for Asian Languages." Observations, 2,2 (First Quarter 1993), 4-11.


SAS, SAS/ACCESS, SAS/AF, SAS/CALC, SAS/CONNECT, SAS/EIS, SAS/FSP, SAS/INSIGHT, and SAS/TOOLKIT software are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. IBM, DB2, CICS, and SQL/DS are registered trademarks or trademarks of International Business Machines Corporation. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.