CHAPTER 13

Formats

Formats in the CMS Environment

In general, formats are completely portable. Only the formats that have host-specific behavior are documented in this section.

All of these formats are described in *SAS Language Reference: Dictionary*; that information is not repeated here. Instead, each format description includes "CMS specifics" information that tells how the format behaves under CMS. Then you are referred to *SAS Language Reference: Dictionary*.

Considerations for Using Formats under CMS

EBCDIC and Character Data

The following character formats produce different results on different computing platforms, depending on which character-encoding system the platform uses. Because CMS uses the EBCDIC character-encoding system, all of the following formats convert data from EBCDIC.

These formats are not discussed in detail in this section because the EBCDIC character-encoding system is their only host-specific aspect.

$ASCIIw.
converts EBCDIC character data to ASCII character data.

$BINARYw.
converts EBCDIC character data to binary values.

$CHARw.
writes standard character data.

EBCDICw.
converts native format character data to EBCDIC representation.

$HEXw.
converts EBCDIC character data to hexadecimal data.

$OCTALw.
converts EBCDIC character data to octal data.

VARYINGw.
writes varying-length character values.

$w.
writes standard character data.

All the information that you need in order to use these formats under CMS is included in *SAS Language Reference: Dictionary*.

Floating-Point Number Format and Portability

The manner in which CMS stores floating-point numbers can affect your data. See "Representation of Floating-Point Numbers" on page 111 for details.

Writing Binary Data

If a SAS program that writes binary data is run on only one type of machine, you can use the following native-mode formats. Native mode means that these formats use the byte-ordering system and floating-point representation which are standard for the machine.

IBw.d	writes integer binary (fixed-point) values, including negative values, which are represented in twos complement notation
PDw.d	writes data that are stored in the IBM packed decimal format
PIBw.d	writes positive integer binary (fixed-point) values
RBw.d	writes real binary (floating-point) data.
ZD	writes zoned decimal data.

If you want to write SAS programs that can be run on multiple machines that use different byte-storage systems, then use the following IBM 370 formats:

S370FIBw.d
writes integer binary data in the IBM mainframe format

S370FIBU
writes unsigned integer binary data in the IBM mainframe format

S370FPDw.d
writes packed decimal data in the IBM mainframe format

S370FPDU
writes unsigned packed decimal data in the IBM mainframe format

S370FPIBw.d
: writes positive integer binary data in the IBM mainframe format

S370FRBw.d
: writes real binary data in the IBM mainframe format

S370FZD
: writes zoned decimal data in the IBM mainframe format

S370FZDL
: writes zoned decimal leading sign data in the IBM mainframe format

S370FZDS
: writes zoned decimal separate leading sign data in the IBM mainframe format

S370FZDT
: writes zoned decimal separate trailing sign data in the IBM mainframe format

S370FZDU
: writes unsigned zoned decimal data in the IBM mainframe format.

These IBM 370 formats enable you to write SAS programs that can be run in any SAS environment, regardless of the standard for storing numeric data. They also enhance your ability to port raw data between host operating environments.

For more information about the IBM 370 formats, see *SAS Language Reference: Dictionary*.

BEST

SAS System chooses best notation (default)

Numeric

Width range: 1– 32

Default width: 12

Alignment: right

CMS specifics: minimum and maximum values

Syntax

BEST*w*.

w
specifies the field width of the output value.

Details

When a format is not specified, the SAS System chooses the notation that gives the most precision and information about the value that fits into the number of columns available. Numbers are interpreted using the EBCDIC character-encoding system, with one digit per byte.

Under CMS, acceptable values can range from 5.398E-79 to 7.237E+75. Any number outside this range causes an overflow error. The following table illustrates the use of the BEST format.

Decimal Number	Format	EBCDIC Data Pattern Written	Actual NumericValue
1234	best6.	’4040F1F2F3F4’x	1234
-1234	best6.	’4060F1F2F3F4’x	-1234
12.34	best6.	’40F1F24BF3F4’x	12.34
123456789	best6.	’F14BF2F3C5F8’x	1.23E8
123456789	best8.	’F14BF2F3F4F6C5F8’x	1.2346E8

Note:

- ’40’x=blank
- ’F1’x=1, ’F2’x=2, and so on
- ’4B’x=decimal point
- ’C5’x=E
- ’4E’x=plus sign
- ’60’x=minus sign.

△

See Also

- *SAS Language Reference: Dictionary*

E

Writes numeric values stored in scientific notation

Numeric

Width range: 7– 32

Default width: 12

Alignment: right

CMS specifics: minimum and maximum values

Syntax

E*w*.

w
specifies the field width of the output value, in bytes.

Details

Numbers are interpreted using the EBCDIC character-encoding system, with one digit per byte. Under CMS, acceptable values can range from 5.398E-79 to 7.237E+75. Any number outside this range causes an overflow error. The following table illustrates the use of the E format.

Decimal Number	Format	EBCDIC Data Pattern Written	Scientific Notation
123	e10.	’40F14BF2F3F0C54EF0F2’x	1.230E+02
-123	e10.	’60F14BF2F3F0C54EF0F2’x	-1.230E+02
12.3	e10.	’40F14BF2F3F0C54EF0F1’x	1.230E+01
123456789	e10.	’40F14BF2F3F5C54EF0F8’x	1.235E+08
123456789	e8.	’40F14BF2C54EF0F8’x	1.2E+08
0.1230	e10.	’40F14BF2F3F0C560F0F1’x	1.230E-01

Note:
- ’40’x=blank
- ’F1’x=1, ’F2’x=2, and so on
- ’4B’x=decimal point
- ’C5’x=E
- ’4E’x=plus sign
- ’60’x=minus sign.

△

See Also

- Informat: “E” on page 166
- *SAS Language Reference: Dictionary*

HEX

Converts hexadecimal positive binary values to either integer (fixed-point) or real (floating-point) values.

Width range: 1– 16

Default width: 8

Alignment: left

CMS specifics: IBM floating-point format

Syntax

HEX*w*.

w

specifies the field width of the output value and determines whether the output is an integer or real binary value.

Details

The HEX format converts real binary (floating-point) numbers to hexadecimal representation. Each hexadecimal digit that is written in the EBCDIC code uses one byte per digit. For example, the floating-point number 1.0 has the hexadecimal value 'F1'x (EBCDIC 1) using the HEX1. format.

The *w* value of the HEX format determines the width of the value and whether the number is written as a floating-point number or an integer. When you specify a width value of 1 through 15, the real binary numbers are truncated to fixed-point integers before being written to hexadecimal values. However, when you specify 16 for the width, the floating-point value is written and the numbers are not truncated. For example, if the value of Y is 31.5, and you use the following PUT statement to write it to the SAS log, the following hexadecimal value is written in EBCDIC code:

```
put y hex16.;
421F800000000000
   ('F4F2F1C6F8F0F0F0F0F0F0F0F0F0F0F0'x)
```

The result shows the hexadecimal value for the CMS floating-point representation of 31.5. The value of a floating-point number is as follows: the first bit of this number is the sign bit and the next seven bits are the characteristic, or exponent. Since the sign bit is 0, the number is positive. To calculate the exponent from the characteristic, you must subtract hexadecimal 40 from the number. Subtracting '40'x from '42'x gives a difference of '02'x. Thus, the exponent is 2_{16}. The last part of the floating-point number, bits 8 through 63, represents the fraction. Since the exponent is 2_{16} , the radix point is moved two places to the right, giving a value of '1F.8'x, which is the hexadecimal equivalent of decimal 31.5.

If you set the variable Y equal to -31.5, you get the following result with a width of 16 specified:

```
C21F800000000000 ('C3F2F1C6F8F0F0F0'x)
```

The only difference between this example and the first example is the changing of the first digit from '4'x to 'C'x. This occurs because the sign bit has been changed from 0 to 1 if you set the variable Y equal to -31.5.

However, if you change the format to HEX15. in the first example, the result writes the following hexadecimal value in EBCDIC code:

```
'00000000000001F'x ('F0F0F0F0F0F0F1C6'x)
```

This example illustrates the result when a width value of less than 16 is specified. Here, the SAS System first converts 31.5 to an integer by truncating the number to 31. The result is then printed in the specified number of hexadecimal digits.

With a width of less than 16, a negative floating-point number is first truncated to an integer and then printed in twos complement form. Therefore, when the format HEX15. is specified for Y=-31.5, the result is as follows:

```
FFFFFFFFFFFFFE1 ('C6C6C6C6C6C6C5F1'x)
```

See Also

- "Representation of Floating-Point Numbers" on page 111
- Informat: "HEX" on page 167
- *SAS Language Reference: Dictionary*.

IB

Writes numbers in integer binary (fixed-point) format

Numeric

Width range: 1– 8

Default width: 4

Decimal range: 0– 10

Alignment: left

CMS specifics: twos complement notation

Syntax

IB*w*.

w
specifies the field width of the output value.

d
specifies a multiplier for the output value. If the format includes a *d* value, the output value is multiplied by 10^d.

Details

Negative values are stored in twos complement notation under CMS. The following table shows several examples of the IB format.

Decimal Number	Format	Integer Binary Data Pattern Written	Actual Numeric Value
1234	ib4.	’000004D2’x	1234
-1234	ib4.	’FFFFFB2E’x	-1234
12.34	ib4.	’0000000C’x	12
12.34	ib4.2	’000004D2’x	1234
123456789	ib4.	’075BCD15’x	123456789
1234	ib6.2	’00000001E208’x	123400
-1234	ib6.2	’FFFFFFFE1DF8’x	-123400

See Also

- Informat: “IB” on page 168
- *SAS Language Reference: Dictionary*

PD

Writes values in IBM packed decimal format

Numeric

Width range: 1– 16

Default width: 1

Decimal range: 0– 10

Alignment: left

CMS specifics: IBM packed decimal format

Syntax

PD*w.d*

w
specifies the field width of the output value, in bytes.

d
specifies a multiplier for the output value. If the format includes a *d* value, the output value is multiplied by 10^{d}.

Details

An IBM packed decimal number consists of a sign and up to 31 digits, thus giving a range from 10^{31} -1 to 10^{-31} +1. The sign is written in the rightmost nibble, with a ’C’x indicating a plus sign and a ’D’x indicating a minus sign. The rest of the nibbles to the left of the sign nibble represent decimal digits. The hexadecimal values of these digit nibbles correspond to decimal values; therefore, only values between ’0’x and ’9’x are displayed in the digit positions. The following table shows several examples of the PD format.

Decimal Number	Format	Packed Decimal Data Pattern Written
1234	pd3.	’01234C’x
-1234	pd3.	’01234D’x
1234	pd2.	’999C’x
1234	pd4.	’0001234C’x
1234	pd4.1	’0012340C’x
1234	pd4.2	’0123400C’x

See Also

- Informat: “PD” on page 169
- *SAS Language Reference: Dictionary*

RB

Writes real binary (floating-point) data

Numeric

Width range: 2– 8

Default width: 4

Decimal range: 0– 10

Alignment: left

CMS specifics: IBM floating-point format

Syntax

RB*w.d*

w
specifies the field width of the output value, in bytes.

d
specifies a multiplier for the output value. If the format includes a *d* value, the output value is multiplied by 10^d.

Details

The format of floating-point numbers is specific to CMS. (See “Representation of Floating-Point Numbers” on page 111 for a description of the format used to store floating-point numbers.) The following table shows how several decimal numbers are written as floating-point numbers using the RB format.

Decimal Number	Format	Real Binary Data Pattern Written	Actual Numeric Value
123	rb8.	’427B000000000000’x	123
123	rb8.1	’434CE00000000000’x	1230
123	rb8.2	’44300C0000000000’x	12300
-123	rb8.	’C27B000000000000’x	-123
1234	rb8.	’434D200000000000’x	1234
1234	rb2.	’434D’x	1232
12.3	rb8.	’41C4CCCCCCCCCCCC’x	12.29999999...

See Also

- “Representation of Floating-Point Numbers” on page 111
- Informat: “RB” on page 170
- *SAS Language Reference: Dictionary*.

w.d

Writes standard numeric data one digit per byte using EBCDIC code

Numeric

Width range: 1– 32

Decimal range: *d* < *w*

Alignment: right

CMS specifics: minimum and maximum values

Syntax

w.d

w
specifies the field width of the output value.

d
specifies the number of digits to the right of the decimal point in the numeric value.

Details

Use this format to print values without additional formatting. If *d* is 0 or if it is omitted, the value is written without a decimal point. Negative numbers are written with a leading minus sign.

Numbers written with the *w.d* format are rounded to the nearest number that can be represented in the output field. If the number is too large to fit, the field is filled with asterisks (*). Under CMS, acceptable values that can be written with the *w.d* format can range from 5.398E-79 to 7.237E+75.

When choosing *w* and *d* values, allow enough space for the decimal point and minus sign if necessary. The following table illustrates the use of the *w.d* format.

Decimal Number	Format	Data Pattern Written
1234	4.	’F1F2F3F4’x
1234	5.	’40F1F2F3F4’x
12345	4.	’F1F2C5F3’x
12345	5.	’F1F2F3F4F5’x
123.4	6.2	’F1F2F34BF4F0’x
-1234	6.	’4060F1F2F3F4’x

Note:

- ’40’x=blank
- ’F1’x=1, ’F2’x=2, and so on
- ’4B’x=decimal point
- ’C5’x=E
- ’4E’x=plus sign
- ’60’x=minus sign.

△

See Also

- □ Informat: “w.d” on page 171
- □ *SAS Language Reference: Dictionary*

ZD

Writes zoned decimal data, one digit per byte

Numeric

Width range: 1– 32

Default width: 1

Decimal range: 0– 31

Alignment: left

CMS specifics: IBM zoned decimal format

Syntax

ZD*w.d*

w
specifies the field width of the output value.

d
specifies a multiplier for the output value. If the format includes a *d* value, the output value is multiplied by 10^{d}.

Details

The ZD format fills in zeros to the left of the data value. Like standard format, zoned decimal digits are represented as EBCDIC characters. Each digit requires one byte of storage space. The rightmost byte represents both the least significant digit and the sign of the number. Digits to the left of the least significant digit are written as the EBCDIC characters 0 through 9.

The character written for the least significant digit depends on the sign of the number. In the least significant byte, negative numbers are coded with the high-order nibble being an ’D’x and with the low-order nibble being the last digit of the number. Positive values are represented with the high-order nibble being a ’C’x. For example, compare the zoned decimal data for 123 and -123 in the following table.

Decimal Number	**Format**	**Zoned Decimal Data Pattern Written**	**Actual Value Value**
123	zd8.	’F0F0F0F0F0F1F2C3’x	0000012C
1234	zd8.	’F0F0F0F0F1F2F3C4’x	0000123D
123	zd8.1	’F0F0F0F0F1F2F3C0’x	0000123{
123	zd8.2	’F0F0F0F1F2F3F0C0’x	0001230{

Decimal Number	**Format**	**Zoned Decimal Data Pattern Written**	**Actual Value Value**
-123	zd8.	’F0F0F0F0F0F1F2D3’x	0000012L
0.000123	zd8.6	’F0F0F0F0F0F1F2C3’x	0000012C
0.00123	zd8.6	’F0F0F0F0F1F2F3C0’x	0000123{
1E-6	zd8.6	’F0F0F0F0F0F0F0C1’x	0000000A

Note:
- ’F0’x=0, ’F1’x=1, and so on
- ’C0’x=+0, ’C1’x=+1, and so on
- ’D0’x=-0, ’D1’x=-1, and so on.

△

See Also

- Informat: “ZD” on page 173
- Informat: “ZDB” on page 174
- *SAS Language Reference: Dictionary*

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Companion for the CMS Environment, Version 8* , Cary, NC: SAS Institute Inc., 1999.

SAS® Companion for the CMS Environment, Version 8

Copyright © 1999 by SAS Institute Inc., Cary, NC, USA.

ISBN 1–58025–481–0

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999