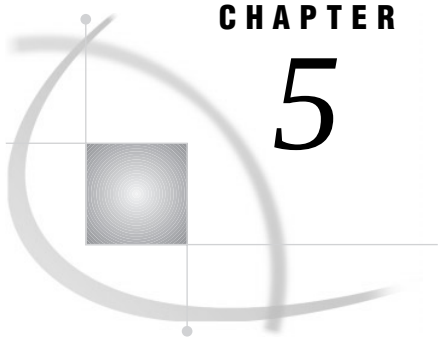

CHAPTER 5

Formats

Definition

A *format* is an instruction that SAS uses to write data values. You use formats to control the written appearance of data values, or, in some cases, to group data values together for analysis. For example, the WORDS22. format, which converts numeric values to their equivalent in words, writes the numeric value 692 as **`six hundred ninety-two`**.

Syntax

SAS formats have the following form:

<$>*format*<*w*>.<*d*>

where

$
indicates a character format; its absence indicates a numeric format.

format
names the format. The format is a SAS format or a user-defined format that was previously defined with the VALUE statement in PROC FORMAT. For more information on user-defined formats, see the FORMAT procedure in the *SAS Procedures Guide*.

w
specifies the format width, which for most formats is the number of columns in the output data.

d
specifies an optional decimal scaling factor in the numeric formats.

Formats always contain a period (.) as a part of the name. If you omit the *w* and the *d* values from the format, SAS uses default values. The *d* value that you specify with a format tells SAS to display that many decimal places, regardless of how many decimal places are in the data. Formats never change or truncate the internally stored data values.

For example, in DOLLAR10.2, the *w* value of 10 specifies a maximum of 10 columns for the value. The *d* value of 2 specifies that two of these columns are for the decimal part of the value, which leaves eight columns for all the remaining characters in the value. This includes the decimal point, the remaining numeric value, a minus sign if the value is negative, the dollar sign, and commas, if any.

If the format width is too narrow to represent a value, SAS tries to squeeze the value into the space available. Character formats truncate values on the right. Numeric formats sometimes revert to the BEST*w.d* format. SAS prints asterisks if you do not specify an adequate width. In the following example, the result is x=**.

```
x=123;
put x=2.;
```

If you use an incompatible format, such as using a numeric format to write character values, SAS first attempts to use an analogous format of the other type. If this is not feasible, an error message that describes the problem appears in the SAS log.

Using Formats

Ways to Specify Formats

You can use formats in the following ways:

- in a PUT statement
- with the PUT, PUTC, or PUTN functions
- with the %SYSFUNC macro function
- in a FORMAT statement in a DATA step or a PROC step
- in an ATTRIB statement in a DATA step or a PROC step.

PUT Statement

The PUT statement with a format after the variable name uses a format to write data values in a DATA step. For example, this PUT statement uses the DOLLAR. format to write the numeric value for AMOUNT as a dollar amount:

```
amount=1145.32;
put amount dollar10.2;
```

The DOLLAR*w.d* format in the PUT statement produces this result:

```
$1,145.32
```

For more information, see the PUT statement in *SAS Language Reference: Dictionary*.

PUT Function

The PUT function writes a numeric variable, a character variable, or a constant with any valid format and returns the resulting character value. For example, the following statement converts the values of a numeric variable into a two-character hexadecimal representation:

```
num=15;
char=put(num,hex2.);
```

The PUT function creates a character variable named CHAR that has a value of 0F.

The PUT function is useful for converting a numeric value to a character value. For more information, see the PUT function in *SAS Language Reference: Dictionary*.

%SYSFUNC

The %SYSFUNC (or %QSYSFUNC) macro function executes SAS functions or user-defined functions and applies an optional format to the result of the function outside a DATA step. For example, the following program writes a numeric value in a macro variable as a dollar amount.

```
%macro tst(amount);
  %put %sysfunc(putn(&amount,dollar10.2));
%mend tst;

%tst (1154.23);
```

For more information, see *SAS Macro Language: Reference*.

FORMAT Statement

The FORMAT statement permanently associates a format with a variable. SAS uses the format to write the values of the variable that you specify. For example, the following statement in a DATA step associates the COMMA*w.d* numeric format with the variables SALES1 through SALES3:

```
format sales1-sales3 comma10.2;
```

Because the FORMAT statement permanently associates a format with a variable, any subsequent DATA step or PROC step uses COMMA10.2 to write the values of SALES1, SALES2, and SALES3. For more information, see the FORMAT statement in *SAS Language Reference: Dictionary*.

Note: Formats that you specify in a PUT statement behave differently from those that you associate with a variable in a FORMAT statement. The major difference is that formats that are specified in the PUT statement will preserve leading blanks. If

you assign formats with a FORMAT statement prior to a PUT statement, all leading blanks are trimmed. The result is the same as if you used the colon (:) format modifier. For details about using the colon (:) format modifier, see the PUT, List statement in *SAS Language Reference: Dictionary*. △

ATTRIB Statement

The ATTRIB statement can also associate a format, as well as other attributes, with one or more variables. For example, in the following statement the ATTRIB statement permanently associates the COMMA*w.d* format with the variables SALES1 through SALES3:

```
attrib sales1-sales3 format=comma10.2;
```

Because the ATTRIB statement permanently associates a format with a variable, any subsequent DATA step or PROC step uses COMMA10.2 to write the values of SALES1, SALES2, and SALES3. For more information, see the ATTRIB statement in *SAS Language Reference: Dictionary*.

Permanent versus Temporary Association

When you specify a format in a PUT statement, SAS uses the format to write data values during the DATA step but does not permanently associate the format with a variable. To permanently associate a format with a variable, use a FORMAT statement or an ATTRIB statement in a DATA step. SAS permanently associates a format with the variable by modifying the descriptor information in the SAS data set.

Using a FORMAT statement or an ATTRIB statement in a PROC step associates a format with a variable for that PROC step, as well as for any output data sets that the procedure creates that contain formatted variables. For more information on using formats in SAS procedures, see the *SAS Procedures Guide*.

User-Defined Formats

In addition to the formats that are supplied with base SAS software, you can create your own formats. In base SAS software, PROC FORMAT allows you to create your own formats for both character and numeric variables. For more information, see the FORMAT procedure in the *SAS Procedures Guide*.

When you execute a SAS program that uses user-defined formats, these formats should be available. The two ways to make these formats available are

- to create permanent, not temporary, formats with PROC FORMAT
- to store the source code that creates the formats (the PROC FORMAT step) with the SAS program that uses them.

To create permanent SAS formats, see the FORMAT procedure in the *SAS Procedures Guide*.

If you execute a program that cannot locate a user-defined format, the result depends on the setting of the FMTERR system option. If the user-defined format is not found, then these system options produce these results:

System Options	Results
FMTERR	SAS produces an error that causes the current DATA or PROC step to stop.
NOFMTERR	SAS continues processing and substitutes a default format, usually the BEST*w*. or $*w*. format.

Although using NOFMTERR enables SAS to process a variable, you lose the information that the user-defined format supplies.

To avoid problems, make sure that your program has access to all user-defined formats that are used.

Byte Ordering on Big Endian and Little Endian Platforms

Definitions

Integer values are typically stored in one of three sizes: one-byte, two-byte, or four-byte. The ordering of the bytes for the integer varies depending on the platform (operating environment) on which the integers were produced.

The ordering of bytes differs between the "big endian" and "little endian" platforms. These colloquial terms are used to describe byte ordering for IBM mainframes (big endian) and for Intel-based platforms (little endian). In the SAS System, the following platforms are considered big endian: AIX, HP-UX, IBM mainframe, Macintosh, and Solaris. The following platforms are considered little endian: AXP/VMS, Digital UNIX, Intel ABI, OS/2, VAX/VMS, and Windows.

How Bytes are Ordered Differently

On big endian platforms, the value 1 is stored in binary and is represented here in hexadecimal notation. One byte is stored as 01, two bytes as 00 01, and four bytes as 00 00 00 01. On little endian platforms, the value 1 is stored in one byte as 01 (the same as big endian), in two bytes as 01 00, and in four bytes as 01 00 00 00.

If an integer is negative, the "two's complement" representation is used. The high-order bit of the most significant byte of the integer will be set on. For example, –2 would be represented in one, two, and four bytes on big endian platforms as FE, FF FE, and FF FF FF FE respectively. On little endian platforms, the representation would be FE, FE FF, and FE FF FF FF.

Writing Data Generated on Big Endian or Little Endian Platforms

SAS can read signed and unsigned integers regardless of whether they were generated on a big endian or a little endian system. Likewise, SAS can write signed and unsigned integers in both big endian and little endian format. The length of these integers can be up to eight bytes.

The following table shows which format to use for various combinations of platforms. In the Sign? column, "no" indicates that the number is unsigned and cannot be negative. "Yes" indicates that the number can be either negative or positive.

Table 5.1 SAS Formats and Byte Ordering

Data created for ...	Data written by ...	Sign?	Format
big endian	big endian	yes	IB or S370FIB
big endian	big endian	no	PIB, S370FPIB, S370FIBU
big endian	little endian	yes	S370FIB
big endian	little endian	no	S370FPIB
little endian	big endian	yes	IBR
little endian	big endian	no	PIBR
little endian	little endian	yes	IB or IBR
little endian	little endian	no	PIB or PIBR
big endian	either	yes	S370FIB
big endian	either	no	S370FPIB
little endian	either	yes	IBR
little endian	either	no	PIBR

Integer Binary Notation and Different Programming Languages

The following table compares integer binary notation according to programming language.

Table 5.2 Integer Binary Notation and Programming Languages

Language	2 Bytes	4 Bytes
SAS	IB2., IBR2., PIB2., PIBR2., S370FIB2., S370FIBU2., S370FPIB2.	IB4., IBR4., PIB4., PIBR4., S370FIB4., S370FIBU4., S370FPIB4.
PL/I	FIXED BIN(15)	FIXED BIN(31)
FORTRAN	INTEGER*2	INTEGER*4
COBOL	COMP PIC 9(4)	COMP PIC 9(8)
IBM assembler	H	F
C	short	long

Working with Packed Decimal and Zoned Decimal Data

Definitions

Packed decimal	specifies a method of encoding decimal numbers by using each byte to represent two decimal digits. Packed decimal representation stores decimal data with exact precision. The fractional part of the number is determined by the informat or format because there is no separate mantissa and exponent. An advantage of using packed decimal data is that exact precision can be maintained. However, computations involving decimal data may become inexact due to the lack of native instructions.
Zoned decimal	specifies a method of encoding decimal numbers in which each digit requires one byte of storage. The last byte contains the number's sign as well as the last digit. Zoned decimal data produces a printable representation.
Nibble	specifies 1/2 of a byte.

Types of Data

Packed Decimal Data

A packed decimal representation stores decimal digits in each "nibble" of a byte. Each byte has two nibbles, and each nibble is indicated by a hexadecimal digit. For example, the value 15 is stored in two nibbles, using the hexadecimal digits 1 and 5.

The sign indication is dependent on your operating environment. On IBM mainframes, the sign is indicated by the last nibble. With formats, C indicates a positive value, and D indicates a negative value. With informats, A, C, E, and F indicate positive values, and B and D indicate negative values. Any other nibble is invalid for signed packed decimal data. In all other operating environments, the sign is indicated in its own byte. If the high-order bit is 1, then the number is negative. Otherwise, it is positive.

The following applies to packed decimal data representation:

- You can use the S370FPD format on all platforms to obtain the IBM mainframe configuration.
- You can have unsigned packed data with no sign indicator. The packed decimal format and informat handles the representation. It is consistent between ASCII and EBCDIC platforms.
- Note that the S370FPDU format and informat expects to have an F in the last nibble, while packed decimal expects no sign nibble.

Zoned Decimal Data

The following applies to zoned decimal data representation:

- A zoned decimal representation stores a decimal digit in the low order nibble of each byte. For all but the byte containing the sign, the high-order nibble is the numeric zone nibble (F on EBCDIC and 3 on ASCII).

- The sign can be merged into a byte with a digit, or it can be separate, depending on the representation. But the standard zoned decimal format and informat expects the sign to be merged into the last byte.
- The EBCDIC and ASCII zoned decimal formats produce the same printable representation of numbers. There are two nibbles per byte, each indicated by a hexadecimal digit. For example, the value 15 is stored in two bytes. The first byte contains the hexadecimal value F1 and the second byte contains the hexadecimal value C5.

Packed Julian Dates

The following applies to packed Julian dates:

- The two formats and informats that handle Julian dates in packed decimal representation are PDJULI and PDJULG. PDJULI uses the IBM mainframe year computation, while PDJULG uses the Gregorian computation.
- The IBM mainframe computation considers 1900 to be the base year, and the year values in the data indicate the offset from 1900. For example, 98 means 1998, 100 means 2000, and 102 means 2002. 1998 would mean 3898.
- The Gregorian computation allows for 2-digit or 4-digit years. If you use 2-digit years, SAS uses the setting of the YEARCUTOFF value to determine the true year.

Platforms Supporting Packed Decimal and Zoned Decimal Data

Some platforms have native instructions to support packed and zoned decimal data, while others must use software to emulate the computations. For example, the IBM mainframe has an Add Pack instruction to add packed decimal data, but the Intel-based platforms have no such instruction and must convert the decimal data into some other format.

Languages Supporting Packed Decimal and Zoned Decimal Data

Several different languages support packed decimal and zoned decimal data. The following table shows how COBOL picture clauses correspond to SAS formats and informats.

IBM VS COBOL II clauses	Corresponding S370Fxxx formats/informats
PIC S9(X) PACKED-DECIMAL	S370FPDw.
PIC 9(X) PACKED-DECIMAL	S370FPDUw.
PIC S9(W) DISPLAY	S370FZDw.
PIC 9(W) DISPLAY	S370FZDUw.
PIC S9(W) DISPLAY SIGN LEADING	S370FZDLw.
PIC S9(W) DISPLAY SIGN LEADING SEPARATE	S370FZDSw.
PIC S9(W) DISPLAY SIGN TRAILING SEPARATE	S370FZDTw.

For the packed decimal representation listed above, X indicates the number of digits represented, and W is the number of bytes. For PIC S9(X) PACKED-DECIMAL, W is **`ceil((x+1)/2)`**. For PIC 9(X) PACKED-DECIMAL, W is **`ceil (x/2)`**. For example,

PIC S9(5) PACKED-DECIMAL represents five digits. If a sign is included, six nibbles are needed. **ceil((5+1)/2)** has a length of three bytes, and the value of W is 3.

Note that you can substitute COMP-3 for PACKED-DECIMAL.

In IBM assembly language, the P directive indicates packed decimal, and the Z directive indicates zoned decimal. The following shows an excerpt from an assembly language listing, showing the offset, the value, and the DC statement:

```
offset  value (in hex)      inst label     directive

+000000 00001C                2 PEX1       DC PL3'1'
+000003 00001D                3 PEX2       DC PL3'-1'
+000006 F0F0C1                4 ZEX1       DC ZL3'1'
+000009 F0F0D1                5 ZEX2       DC ZL3'1'
```

In PL/I, the FIXED DECIMAL attribute is used in conjunction with packed decimal data. You must use the PICTURE specification to represent zoned decimal data. There is no standardized representation of decimal data for the FORTRAN or the C languages.

Summary of Packed Decimal and Zoned Decimal Formats and Informats

SAS uses a group of formats and informats to handle packed and zoned decimal data. The following table lists the type of data representation for these formats and informats. Note that the formats and informats that begin with S370 refer to IBM mainframe representation.

Format	Type of data representation	Corresponding informat	Comments
PD	Packed decimal	PD	Local signed packed decimal
PK	Packed decimal	PK	Unsigned packed decimal; not specific to your operating environment
ZD	Zoned decimal	ZD	Local zoned decimal
none	Zoned decimal	ZDB	Translates EBCDIC blank (hex 40) to EBCDIC zero (hex F0), then corresponds to the informat as zoned decimal
none	Zoned decimal	ZDV	Non-IBM zoned decimal representation
S370FPD	Packed decimal	S370FPD	Last nibble C (positive) or D (negative)
S370FPDU	Packed decimal	S370FPDU	Last nibble always F (positive)
S370FZD	Zoned decimal	S370FZD	Last byte contains sign in upper nibble: C (positive) or D (negative)
S370FZDU	Zoned decimal	S370FZDU	Unsigned; sign nibble always F

Format	Type of data representation	Corresponding informat	Comments
S370FZDL	Zoned decimal	S370FZDL	Sign nibble in first byte in informat; separate leading sign byte of hex C0 (positive) or D0 (negative) in format
S370FZDS	Zoned decimal	S370FZDS	Leading sign of - (hex 60) or + (hex 4E)
S370FZDT	Zoned decimal	S370FZDT	Trailing sign of - (hex 60) or + (hex 4E)
PDJULI	Packed decimal	PDJULI	Julian date in packed representation - IBM computation
PDJULG	Packed decimal	PDJULG	Julian date in packed representation - Gregorian computation
none	Packed decimal	RMFDUR	Input layout is: *mmssttt*F
none	Packed decimal	SHRSTAMP	Input layout is: *yyyyddd*F*hhmmssth*, where *yyyyddd*F is the packed Julian date; *yyyy* is a 0-based year from 1900
none	Packed decimal	SMFSTAMP	Input layout is: *xxxxxxxxyyyyddd*F, where *yyyyddd*F is the packed Julian date; *yyyy* is a 0-based year from 1900
none	Packed decimal	PDTIME	Input layout is: 0*hhmmss*F
none	Packed decimal	RMFSTAMP	Input layout is: 0*hhmmss*F*yyyyddd*F, where *yyyyddd*F is the packed Julian date; yyyy is a 0-based year from 1900

Formats by Category

There are four categories of formats in SAS:

Category	Description
CHARACTER	instructs SAS to write character data values from character variables.
DATE and TIME	instructs SAS to write data values from variables that represent dates, times, and datetimes.
DBCS	instructs SAS to handle various Asian languages

Category	Description
NUMERIC	instructs SAS to write numeric data values from numeric variables.
USER-DEFINED	instructs SAS to write data values by using a format that is created with PROC FORMAT.

Storing user-defined formats is an important consideration if you associate these formats with variables in permanent SAS data sets, especially those shared with other users. For information on creating and storing user-defined formats, see the FORMAT procedure in the *SAS Procedures Guide*.

The following table provides brief descriptions of the SAS formats. For more detailed descriptions, see the "Formats" chapter of *SAS Language Reference: Dictionary*.

Table 5.3 Categories and Descriptions of Formats

Category	Format	Description
Character	$ASCII*w*.	Converts native format character data to ASCII representation
	$BINARY*w*.	Converts character data to binary representation
	$CHAR*w*.	Writes standard character data
	$EBCDIC*w*.	Converts native format character data to EBCDIC representation
	$HEX*w*.	Converts character data to hexadecimal representation
	$MSGCASE*w*.	Writes character data in uppercase when the MSGCASE system option is in effect
	$OCTAL*w*.	Converts character data to octal representation
	$QUOTE*w*.	Writes data values that are enclosed in double quotation marks
	$REVERJ*w*.	Writes character data in reverse order and preserves blanks
	$REVERS*w*.	Writes character data in reverse order and left aligns
	$UPCASE*w*.	Converts character data to uppercase
	$VARYING*w*.	Writes character data of varying length
	$*w*.	Writes standard character data
DBCS	$KANJI*w*.	Adds shift-code data to DBCS data
	$KANJIX*w*.	Removes shift code data from DBCS data
Date and Time	DATE*w*.	Writes date values in the form *ddmmmyy* or *ddmmmyyyy*
	DATEAMPM*w.d*	Writes datetime values in the form *ddmmmyy:hh:mm:ss.ss* with AM or PM
	DATETIME*w.d*	Writes datetime values in the form *ddmmmyy:hh:mm:ss.ss*
	DAY*w*.	Writes date values as the day of the month
	DDMMYY*w*.	Writes date values in the form *ddmmyy* or *ddmmyyyy*

Category	Format	Description
	DDMMYY*x*w.	Writes date values in the form *ddmmyy* or *ddmmyyyy* with a specified separator
	DOWNAME*w*.	Writes date values as the name of the day of the week
	EURDFDD*w*.	Writes international date values in the form *dd.mm.yy* or *dd.mm.yyyy*
	EURDFDE*w*.	Writes international date values in the form *ddmmmyy* or *ddmmmyyyy*
	EURDFDN*w*.	Writes international date values as the day of the week
	EURDFDT*w*.*d*	Writes international datetime values in the form *ddmmmyy:hh:mm:ss.ss* or *ddmmmyyyy hh:mm:ss.ss*
	EURDFDWN*w*.	Writes international date values as the name of the day
	EURDFMN*w*.	Writes international date values as the name of the month
	EURDFMY*w*.	Writes international date values in the form *mmmyy* or *mmmyyyy*
	EURDFWDX*w*.	Writes international date values as the name of the month, the day, and the year in the form *dd month-name yy* (or *yyyy*)
	EURDFWKX*w*.	Writes international date values as the name of the day and date in the form *day-of-week*, *dd month-name yy* (or *yyyy*)
	HHMM*w*.*d*	Writes time values as hours and minutes in the form *hh:mm*
	HOUR*w*.*d*	Writes time values as hours and decimal fractions of hours
	JULDAY*w*.	Writes date values as the Julian day of the year
	JULIAN*w*.	Writes date values as Julian dates in the form *yyddd* or *yyyyddd*
	MINGUO*w*.	Writes date values as Taiwanese dates in the form *yyymmdd*
	MMDDYY*w*.	Writes date values in the form *mmddyy* or *mmddyyyy*
	MMDDYY*x*w.	Writes date values in the form *mmddyy* or *mmddyyyy* with a specified separator
	MMSS*w*.*d*	Writes time values as the number of minutes and seconds since midnight
	MMYY*x*w.	Writes date values as the month and the year and separates them with a character
	MONNAME*w*.	Writes date values as the name of the month
	MONTH*w*.	Writes date values as the month of the year
	MONYY*w*	Writes date values as the month and the year in the form *mmmyy* or *mmmyyyy*
	NENGO*w*.	Writes date values as Japanese dates in the form *e.yymmdd*

Category	Format	Description
	PDJULG*w*.	Writes packed Julian date values in the hexadecimal format *yyyyddd*F for IBM
	PDJULI*w*.	Writes packed Julian date values in the hexadecimal format *ccyyddd*F for IBM
	QTR*w*.	Writes date values as the quarter of the year
	QTRR*w*.	Writes date values as the quarter of the year in Roman numerals
	TIME*w*.	Writes time values as hours, minutes, and seconds in the form *hh:mm:ss.ss*
	TIMEAMPM*w.d*	Writes time values as hours, minutes, and seconds in the form *hh:mm:ss.ss* with AM or PM
	TOD*w.d*	Writes the time portion of datetime values in the form *hh:mm:ss.ss*
	WEEKDATE*w*.	Writes date values as the day of the week and the date in the form *day-of-week, month-name dd, yy* (or *yyyy*)
	WEEKDATX*w*.	Writes date values as day of week and date in the form *day-of-week, dd month-name yy* (or *yyyy*)
	WEEKDAY*w*.	Writes date values as the day of the week
	WORDDATE*w*.	Writes date values as the name of the month, the day, and the year in the form *month-name dd, yyyy*
	WORDDATX*w*.	Writes date values as the day, the name of the month, and the year in the form *dd month-name yyyy*
	YEAR*w*.	Writes date values as the year
	YYMM*xw*.	Writes date values as the year and month and separates them with a character
	YYMMDD*w*.	Writes date values in the form *yymmdd* or *yyyymmdd*
	YYMMDD*xw*.	Writes date values in the form *yymmdd* or *yyyymmdd* with a specified separator
	YYMON*w*.	Writes date values as the year and the month abbreviation
	YYQ*xw*.	Writes date values as the year and the quarter and separates them with a character
	YYQR*xw*.	Writes date values as the year and the quarter in Roman numerals and separates them with characters
Numeric	BEST*w*.	SAS chooses the best notation
	BINARY*w*.	Converts numeric values to binary representation
	COMMA*w.d*	Writes numeric values with commas and decimal points
	COMMAX*w.d*	Writes numeric values with periods and commas
	D*w.s*	Prints variables, possibly with a great range of values, lining up decimal places for values of similar magnitude
	DOLLAR*w.d*	Writes numeric values with dollar signs, commas, and decimal points

Category	Format	Description
	DOLLARX*w.d*	Writes numeric values with dollar signs, periods, and commas
	E*w.*	Writes numeric values in scientific notation
	FLOAT*w.d*	Generates a native single-precision, floating-point value by multiplying a number by 10 raised to the *d*th power
	FRACT*w.*	Converts numeric values to fractions
	HEX*w.*	Converts real binary (floating-point) values to hexadecimal representation
	IB*w.d*	Writes native integer binary (fixed-point) values, including negative values
	IBR*w.d*	Writes integer binary (fixed-point) values in Intel and DEC formats
	IEEE*w.d*	Generates an IEEE floating-point value by multiplying a number by 10 raised to the *d*th power
	NEGPAREN*w.d*	Writes negative numeric values in parentheses
	NUMX*w.d*	Writes numeric values with a comma in place of the decimal point
	OCTAL*w.*	Converts numeric values to octal representation
	PD*w.*	Writes data in packed decimal format
	PERCENT*w.d*	Writes numeric values as percentages
	PIB*w.d*	Writes positive integer binary (fixed-point) values
	PIBR*w.d*	Writes positive integer binary (fixed-point) values in Intel and DEC formats
	PK*w.d*	Writes data in unsigned packed decimal format
	PVALUE*w.d*	Writes *p*-values
	RB*w.d*	Writes real binary data (floating-point) in real binary format
	ROMAN*w.*	Writes numeric values as Roman numerals
	SSN*w.*	Writes Social Security numbers
	S370FF*w.d*	Writes native standard numeric data in IBM mainframe format
	S370FIB*w.d*	Writes integer binary (fixed-point) values, including negative values, in IBM mainframe format
	S370FIBU*w.d*	Writes unsigned integer binary (fixed-point) values in IBM mainframe format
	S370FPD*w.*	Writes packed decimal data in IBM mainframe format
	S370FPDU*w.*	Writes unsigned packed decimal data in IBM mainframe format
	S370FPIB*w.d*	Writes positive integer binary (fixed-point) values in IBM mainframe format

Category	Format	Description
	S370FRB*w.d*	Writes real binary (floating-point) data in IBM mainframe format
	S370FZD*w.d*	Writes zoned decimal data in IBM mainframe format
	S370FZDL*w.d*	Writes zoned decimal leading sign data in IBM mainframe format
	S370FZDS*w.d*	Writes zoned decimal separate leading-sign data in IBM mainframe format
	S370FZDT*w.d*	Writes zoned decimal separate trailing-sign data in IBM mainframe format
	S370FZDU*w.d*	Writes unsigned zoned decimal data in IBM mainframe format
	w.d	Writes standard numeric data one digit per byte
	WORDF*w.*	Writes numeric values as words with fractions that are shown numerically
	WORDS*w.*	Writes numeric values as words
	YEN*w.d*	Writes numeric values with yen signs, commas, and decimal points
	Z*w.d*	Writes standard numeric data with leading 0s
	ZD*w.d*	Writes numeric data in zoned decimal format

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Language Reference: Concepts*, Cary, NC: SAS Institute Inc., 1999. 554 pages.

SAS Language Reference: Concepts

ISBN 1–58025–441–1

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

1st printing, November 1999