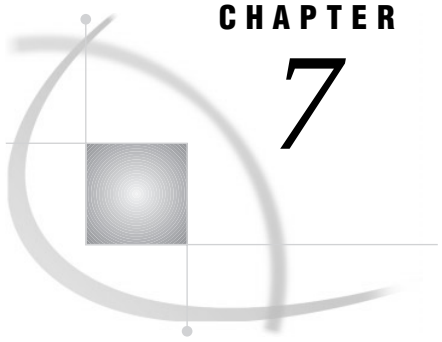

CHAPTER 7

# Informats



## Definition

An *informat* is an instruction that SAS uses to read data values into a variable. For example, the following value contains a dollar sign and commas:

```
$1,000,000
```

To remove the dollar sign ($) and commas (,) before storing the numeric value 1000000 in a variable, read this value with the COMMA11. informat.

Unless you explicitly define a variable first, SAS uses the informat to determine whether the variable is numeric or character. SAS also uses the informat to determine the length of character variables.

# Syntax

SAS informats have the following form:

<$>*informat*<*w*>.<*d*>

where

*$*
indicates a character informat; its absence indicates a numeric informat.

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

*w*
specifies the informat width, which for most informats is the number of columns in the input data.

*d*
specifies an optional decimal scaling factor in the numeric informats. SAS divides the input data by 10 to the power of *d*.

*Note:* Even though SAS can read up to 31 decimal places when you specify some numeric informats, floating-point numbers with more than 12 decimal places might lose precision due to the limitations of the eight-byte floating point representation used by most computers. △

Informats always contain a period (.) as a part of the name. If you omit the *w* and the *d* values from the informat, SAS uses default values. If the data contains decimal points, SAS ignores the *d* value and reads the number of decimal places that are actually in the input data.

If the informat width is too narrow to read all the columns in the input data, you may get unexpected results. The problem frequently occurs with the date and time informats. You must adjust the width of the informat to include blanks or special characters between the day, month, year, or time. For more information about date and time values, see the discussion on SAS date and time values in Chapter 13, “Dates, Times, and Intervals,” on page 147 .

When a problem occurs with an informat, SAS writes a note to the SAS log and assigns a missing value to the variable. Problems occur if you use an incompatible informat, such as a numeric informat to read character data, or if you specify the width of a date and time informat that causes SAS to read a special character in the last column.

# Using Informats

## Ways to Specify Informats

You can specify informats in the following ways:

- in an INPUT statement
- with the INPUT, INPUTC, and INPUTN functions
- in an INFORMAT statement in a DATA or a PROC step

- in an ATTRIB statement in a DATA or a PROC step.

## INPUT Statement

The INPUT statement with an informat after a variable name is the simplest way to read values into a variable. For example, the following INPUT statement uses two informats:

```
input @15 style $3. @21 price 5.2;
```

The $*w*. character informat reads values into the variable STYLE. The *w*.*d* numeric informat reads values into the variable PRICE.

For a complete discussion of the INPUT statement, see *SAS Language Reference: Dictionary*.

## INPUT Function

The INPUT function reads a SAS character expression using a specified informat. The informat determines whether the resulting value is numeric or character. Thus, the INPUT function is useful for converting data. For example,

```
TempCharacter='98.6';
TemperatureNumber=input(TempCharacter,4.);
```

Here, the INPUT function in combination with the *w*.*d* informat reads the character value of TempCharacter as a numeric value and assigns the numeric value 98.6 to TemperatureNumber.

Use the PUT function with a SAS format to convert numeric values to character values. For an example of a numeric-to-character conversion, see the PUT function in *SAS Language Reference: Dictionary*. For a complete discussion of the INPUT function, see the INPUT function in *SAS Language Reference: Dictionary*.

## INFORMAT Statement

The INFORMAT statement associates an informat with a variable. SAS uses the informat in any subsequent INPUT statement to read values into the variable. For example, in the following statements the INFORMAT statement associates the DATE*w*. informat with the variables Birthdate and Interview:

```
informat Birthdate Interview date9.;
input @63 Birthdate Interview;
```

An informat that is associated with an INFORMAT statement behaves like an informat that you specify with a colon (:) format modifier in an INPUT statement. (For details about using the colon (:) modifier, see the INPUT, List statement in *SAS Language Reference: Dictionary*.) Therefore, SAS uses a modified list input to read the variable so that

- the *w* value in an informat does not determine column positions or input field widths in an external file
- the blanks that are embedded in input data are treated as delimiters unless you change the DELIMITER= option in an INFILE statement
- for character informats, the *w* value in an informat specifies the length of character variables
- for numeric informats, the *w* value is ignored
- for numeric informats, the *d* value in an informat behaves in the usual way for numeric informats

If you have coded the INPUT statement to use another style of input, such as formatted input or column input, that style of input is not used when you use the INFORMAT statement.

For more information on how to use modified list input to read data, see the INPUT, List statement in *SAS Language Reference: Dictionary*.

### ATTRIB Statement

The ATTRIB statement can also associate an informat, as well as other attributes, with one or more variables. For example, in the following statements, the ATTRIB statement associates the DATE*w*. informat with the variables Birthdate and Interview:

```
attrib Birthdate Interview informat=date9.;
input @63 Birthdate Interview;
```

An informat that is associated by using the INFORMAT= option in the ATTRIB statement behaves like an informat that you specify with a colon (:) format modifier in an INPUT statement. (For details about using the colon (:) modifier, see the INPUT, List statement in *SAS Language Reference: Dictionary*.) Therefore, SAS uses a modified list input to read the variable in the same way as it does for the INFORMAT statement.

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

## Permanent versus Temporary Association

When you specify an informat in an INPUT statement, SAS uses the informat to read input data values during that DATA step. SAS, however, does not permanently associate the informat with the variable. To permanently associate a format with a variable, use an INFORMAT statement or an ATTRIB statement. SAS permanently associates an informat with the variable by modifying the descriptor information in the SAS data set.

## User-Defined Informats

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

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

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

If you execute a program that cannot locate a user-defined informat, the result depends on the setting of the FMTERR= system option. If the user-defined informat 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 by substituting a default informat. |

Although using NOFMTERR enables SAS to process a variable, you lose the information that the user-defined informat supplies. This option can cause a DATA step to misread data, and it can produce incorrect results.

To avoid problems, make sure that users of your program have access to all the user-defined informats 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 the "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: IBM mainframe, HP-UX, AIX, Solaris, and Macintosh. The following platforms are considered little endian: VAX/VMS, AXP/VMS, Digital UNIX, Intel ABI, OS/2, and Windows.

## How the Bytes are Ordered

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.

## Reading 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 informat 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 7.1** SAS Informats and Byte Ordering

| Data created for … | Data read on … | Sign? | Informat |
|---|---|---|---|
| big endian | big endian | yes | IB or S370FIB |
| big endian | big endian | no | PIB, S370FPIB, S370FIBU |
| big endian | little endian | yes | IBR |
| big endian | little endian | no | PIBR |
| 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 in Different Programming Languages

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

**Table 7.2** 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.<br><br>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 |

# Informat Aliases

Several SAS informats operate identically but have different names. A list of these informat aliases follows. The dictionary of SAS informats uses the primary informat, not aliases, to provide a complete description of its operation.

**Table 7.3** SAS Informats with Aliases

| Primary Informat Name | Informat Alias(es) |
|---|---|
| COMMA*w.d* | DOLLAR*w.d* |
| COMMAX*w.d* | DOLLARX*w.d* |

| | |
|---|---|
| *w.d* | BEST*w.d*, D*w.d*, F*w.d*, E*w.d* |
| $*w.* | $F*w.* |

# Informats by Category

There are five categories of informats in SAS:

| Category | Description |
|---|---|
| CHARACTER | instructs SAS to read character data values into character variables. |
| COLUMN-BINARY | instructs SAS to read data stored in column-binary or multipunched form into character and numeric values. |
| DATE and TIME | instructs SAS to read data values into variables that represent dates, times, and datetimes. |
| NUMERIC | instructs SAS to read numeric data values into numeric variables. |
| USER-DEFINED | instructs SAS to read data values by using an informat that is created with an INVALUE statement in PROC FORMAT. |

For information on reading column-binary data, see "Reading Column-Binary Data" on page 299. For information on creating user-defined informats, see the FORMAT procedure in the *SAS Procedures Guide*.

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

**Table 7.4** Categories and Descriptions of Informats

| Category | Informat | Description |
|---|---|---|
| Character | $ASCII*w.* | Converts ASCII character data to native format |
| | $BINARY*w.* | Converts binary data to character data |
| | $CHAR*w.* | Reads character data with blanks |
| | $CHARZB*w.* | Converts binary 0s to blanks |
| | $EBCDIC*w.* | Converts EBCDIC character data to native format |
| | $HEX*w.* | Converts hexadecimal data to character data |
| | $OCTAL*w.* | Converts octal data to character data |
| | $PHEX*w.* | Converts packed hexadecimal data to character data |
| | $QUOTE*w* | Removes matching quotation marks from character data |
| | $REVERJ*w.* | Reads character data from right to left and preserves blanks |
| | $REVERS*w.* | Reads character data from right to left and left aligns |
| | $UPCASE*w.* | Converts character data to uppercase |
| | $VARYING*w.* | Reads character data of varying length |

| Category | Informat | Description |
|---|---|---|
| | $*w*. | Reads standard character data |
| Column Binary | $CB*w*. | Reads standard character data from column-binary files |
| | CB*w*.*d* | Reads standard numeric values from column-binary files |
| | PUNCH.*d* | Reads whether a row of column-binary data is punched |
| | ROW*w*.*d* | Reads a column-binary field down a card column |
| DBCS | $KANJI*w*. | Removes shift code data from DBCS data |
| | $KANJIX*w*. | Adds shift code data to DBCS data |
| Date and Time | DATE*w*. | Reads date values in the form *ddmmmyy* or *ddmmmyyyy* |
| | DATETIME*w*. | Reads datetime values in the form *ddmmmyy hh:mm:ss.ss* or *ddmmmyyyy hh:mm:ss.ss* |
| | DDMMYY*w*. | Reads date values in the form *ddmmyy* or *ddmmyyyy* |
| | EURDFDE*w*. | Reads international date values |
| | EURDFDT*w*. | Reads international datetime values in the form *ddmmmyy hh:mm:ss.ss* or *ddmmmyyyy hh:mm:ss.ss* |
| | EURDFMY*w*. | Reads month and year date values in the form *mmmyy* or *mmmyyyy* |
| | JDATEYMD*w*. | Reads Japanese kanji date values in the format *yymmmdd* or *yyyymmmdd* |
| | JNENGO*w*. | Reads Japanese Kanji date values in the form *yymmdd* |
| | JULIAN*w*. | Reads Julian dates in the form *yyddd* or *yyyyddd* |
| | MINGUO*w*. | Reads dates in Taiwanese form |
| | MMDDYY*w*. | Reads date values in the form *mmddyy* or *mmddyyyy* |
| | MONYY*w*. | Reads month and year date values in the form *mmmyy* or *mmmyyyy* |
| | MSEC*w*. | Reads TIME MIC values |
| | NENGO*w*. | Reads Japanese date values in the form *eyymmdd* |
| | PDJULG*w*. | Reads packed Julian date values in the hexadecimal form *yyyyddd*F for IBM |
| | PDJULI*w*. | Reads packed Julian dates in the hexadecimal format *ccyyddd* F for IBM |
| | PDTIME*w*. | Reads packed decimal time of SMF and RMF records |
| | RMFDUR*w*. | Reads duration intervals of RMF records |
| | RMFSTAMP*w*. | Reads time and date fields of RMF records |
| | SHRSTAMP*w*. | Reads date and time values of SHR records |
| | SMFSTAMP*w*. | Reads time and date values of SMF records |
| | TIME*w*. | Reads hours, minutes, and seconds in the form *hh:mm:ss.ss* |
| | TODSTAMP*w*. | Reads an eight-byte time-of-day stamp |
| | TU*w*. | Reads timer units |

| Category | Informat | Description |
|---|---|---|
| | YYMMDD*w*. | Reads date values in the form *yymmdd* or *yyyymmdd* |
| | YYMMN*w*. | Reads date values in the form *yyyymm* or *yymm* |
| | YYQ*w*. | Reads quarters of the year |
| Numeric | BINARY*w.d* | Converts positive binary values to integers |
| | BITS*w.d* | Extracts bits |
| | BZ*w.d* | Converts blanks to 0s |
| | COMMA*w.d* | Removes embedded characters |
| | COMMAX*w.d* | Removes embedded characters |
| | E*w.d* | Reads numeric values that are stored in scientific notation and double-precision scientific notation |
| | FLOAT*w.d* | Reads a native single-precision, floating-point value and divides it by 10 raised to the *d*th power |
| | HEX*w*. | Converts hexadecimal positive binary values to either integer (fixed-point) or real (floating-point) binary values |
| | IB*w.d* | Reads native integer binary (fixed-point) values, including negative values |
| | IBR*w.d* | Reads integer binary (fixed-point) values in Intel and DEC formats |
| | IEEE*w.d* | Reads an IEEE floating-point value and divides it by 10 raised to the *d* th power |
| | NUMX*w.d* | Reads numeric values with a comma in place of the decimal point |
| | OCTAL*w.d* | Converts positive octal values to integers |
| | PD*w.d* | Reads data that are stored in IBM packed decimal format |
| | PERCENT*w.d* | Reads percentages as numeric values |
| | PIB*w.d* | Reads positive integer binary (fixed-point) values |
| | PIBR*w.d* | Reads positive integer binary (fixed-point) values in Intel and DEC formats |
| | PK*w.d* | Reads unsigned packed decimal data |
| | RB*w.d* | Reads numeric data that are stored in real binary (floating-point) notation |
| | S370FF*w.d* | Reads EBCDIC numeric data |
| | S370FIB*w.d* | Reads integer binary (fixed-point) values, including negative values, in IBM mainframe format |
| | S370FIBU*w.d* | Reads unsigned integer binary (fixed-point) values in IBM mainframe format |
| | S370FPD*w.d* | Reads packed data in IBM mainframe format |
| | S370FPDU*w.d* | Reads unsigned packed decimal data in IBM mainframe format |
| | S370FPIB*w.d* | Reads positive integer binary (fixed-point) values in IBM mainframe format |

| Category | Informat | Description |
|---|---|---|
| | S370FRB*w.d* | Reads real binary (floating-point) data in IBM mainframe format |
| | S370FZD*w.d* | Reads zoned decimal data in IBM mainframe format |
| | S370FZDL*w.d* | Reads zoned decimal leading-sign data in IBM mainframe format |
| | S370FZDS*w.d* | Reads zoned decimal separate leading-sign data in IBM mainframe format |
| | S370FZDT*w.d* | Reads zoned decimal separate trailing-sign data in IBM mainframe format |
| | S370FZDU*w.d* | Reads unsigned zoned decimal data in IBM mainframe format |
| | VAXRB*w.d* | Reads real binary (floating-point) data in VMS format |
| | *w.d* | Reads standard numeric data |
| | YEN*w.d* | Removes embedded yen signs, commas, and decimal points |
| | ZD*w.d* | Reads zoned decimal data |
| | ZDB*w.d* | Reads zoned decimal data in which zeros have been left blank |
| | ZDV*w.d* | Reads and validates zoned decimal data |

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