CHAPTER

5

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 contain 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 *SAS Language Reference: Concepts*.

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 step or a PROC step
- in an ATTRIB statement in a DATA step 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 “INPUT” on page 876.

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. See "PUT" on page 503 for an example of a numeric-to-character conversion. For a complete discussion of the INPUT function, see "INPUT" on page 402.

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" on page 897.) 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.

See "INPUT, List" on page 897 for more information on how to use modified list input to read data.

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" on page 897.) Therefore, SAS uses a modified list input to read the variable in the same way as it does for the INFORMAT statement.

See "ATTRIB" on page 762 for more information.

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 5.1 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 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	S370ZDw.
PIC 9(W) DISPLAY	S370ZDUw.
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

Format	Type of data representation	Corresponding informat	Comments
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
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

Format	Type of data representation	Corresponding informat	Comments
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 5.3 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 variables.
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 *SAS Language Reference: Concepts*. 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 dictionary of SAS informats.

Table 5.4 Categories and Descriptions of Informats

Category	Informat	Description
Character	"$ASCII*w*." on page 644	Converts ASCII character data to native format
	"$BINARY*w*." on page 645	Converts binary data to character data
	"$CHAR*w*." on page 647	Reads character data with blanks
	"$CHARZB*w*." on page 648	Converts binary 0s to blanks
	"$EBCDIC*w*." on page 649	Converts EBCDIC character data to native format
	"$HEX*w*." on page 650	Converts hexadecimal data to character data
	"$OCTAL*w*." on page 652	Converts octal data to character data
	"$PHEX*w*." on page 653	Converts packed hexadecimal data to character data
	"$QUOTE*w*." on page 654	Removes matching quotation marks from character data
	"$REVERJ*w*." on page 654	Reads character data from right to left and preserves blanks
	"$REVERS*w*." on page 655	Reads character data from right to left and left aligns
	"$UPCASE*w*." on page 656	Converts character data to uppercase
	"$VARYING*w*." on page 657	Reads character data of varying length
	"$*w*." on page 659	Reads standard character data
Column Binary	"$CB*w*." on page 646	Reads standard character data from column-binary files
	"CB*w*.*d*" on page 663	Reads standard numeric values from column-binary files
	"PUNCH.*d*" on page 705	Reads whether a row of column-binary data is punched
	"ROW*w*.*d*" on page 710	Reads a column-binary field down a card column
DBCS	"$KANJI*w*." on page 651	Removes shift code data from DBCS data
	"$KANJIX*w*." on page 651	Adds shift code data to DBCS data
Date and Time	"DATE*w*." on page 666	Reads date values in the form *ddmmmyy* or *ddmmmyyyy*
	"DATETIME*w*. "on page 667	Reads datetime values in the form *ddmmmyy hh:mm:ss.ss* or *ddmmmyyyy hh:mm:ss.ss*
	"DDMMYY*w*." on page 669	Reads date values in the form *ddmmyy* or *ddmmyyyy*
	"EURDFDE*w*. "on page 671	Reads international date values
	"EURDFDT*w*." on page 673	Reads international datetime values in the form *ddmmmyy hh:mm:ss.ss* or *ddmmmyyyy hh:mm:ss.ss*
	"EURDFMY*w*. "on page 675	Reads month and year date values in the form *mmmyy* or *mmmyyyy*
	"JDATEYMD*w*." on page 683	Reads Japanese kanji date values in the format *yymmmdd* or *yyyymmmdd*
	"JNENGO*w*." on page 684	Reads Japanese Kanji date values in the form *yymmdd*
	"JULIAN*w*." on page 685	Reads Julian dates in the form *yyddd* or *yyyyddd*
	"MINGUO*w*." on page 686	Reads dates in Taiwanese form

	“MMDDYY*w*.” on page 687	Reads date values in the form *mmddyy* or *mmddyyyy*
	“MONYY*w*.” on page 689	Reads month and year date values in the form *mmmyy* or *mmmyyyy*
	“MSEC*w*.” on page 690	Reads TIME MIC values
	“NENGO*w*. ”on page 691	Reads Japanese date values in the form *eyymmdd*
	“PDJULG*w*.” on page 696	Reads packed Julian date values in the hexadecimal form *yyyyddd*F for IBM
	“PDJULI*w*.” on page 697	Reads packed Julian dates in the hexadecimal format *ccyyddd* F for IBM
	“PDTIME*w*.” on page 699	Reads packed decimal time of SMF and RMF records
	“RMFDUR*w*. ”on page 707	Reads duration intervals of RMF records
	“RMFSTAMP*w*.” on page 709	Reads time and date fields of RMF records
	“SHRSTAMP*w*. ”on page 711	Reads date and time values of SHR records
	“SMFSTAMP*w*. ”on page 713	Reads time and date values of SMF records
	“TIME*w*.” on page 727	Reads hours, minutes, and seconds in the form *hh:mm:ss.ss*
	“TODSTAMP*w*.” on page 729	Reads an eight-byte time-of-day stamp
	“TU*w*.” on page 730	Reads timer units
	“YYMMDD*w*.” on page 733	Reads date values in the form *yymmdd* or *yyyymmdd*
	“YYMMN*w*.” on page 735	Reads date values in the form *yyyymm* or *yymm*
	“YYQ*w*.” on page 736	Reads quarters of the year
Numeric	“BINARY*w*.*d* ”on page 660	Converts positive binary values to integers
	“BITS*w*.*d* ”on page 661	Extracts bits
	“BZ*w*.*d*” on page 662	Converts blanks to 0s
	“COMMA*w*.*d* ”on page 664	Removes embedded characters
	“COMMAX*w*.*d* ”on page 665	Removes embedded characters
	“E*w*.*d*” on page 670	Reads numeric values that are stored in scientific notation and double-precision scientific notation
	“FLOAT*w*.*d* ”on page 677	Reads a native single-precision, floating-point value and divides it by 10 raised to the *d*th power
	“HEX*w*.” on page 678	Converts hexadecimal positive binary values to either integer (fixed-point) or real (floating-point) binary values
	“IB*w*.*d*” on page 679	Reads native integer binary (fixed-point) values, including negative values
	“IBR*w*.*d*” on page 680	Reads integer binary (fixed-point) values in Intel and DEC formats
	“IEEE*w*.*d*” on page 682	Reads an IEEE floating-point value and divides it by 10 raised to the *d* th power

"NUMX*w.d*" on page 693	Reads numeric values with a comma in place of the decimal point
"OCTAL*w.d* "on page 694	Converts positive octal values to integers
"PD*w.d*" on page 695	Reads data that are stored in IBM packed decimal format
"PERCENT*w.d*" on page 700	Reads percentages as numeric values
"PIB*w.d*" on page 701	Reads positive integer binary (fixed-point) values
"PIBR*w.d*" on page 702	Reads positive integer binary (fixed-point) values in Intel and DEC formats
"PK*w.d*" on page 704	Reads unsigned packed decimal data
"RB*w.d*" on page 706	Reads numeric data that are stored in real binary (floating-point) notation
"S370FF*w.d*" on page 714	Reads EBCDIC numeric data
"S370FIB*w.d*" on page 715	Reads integer binary (fixed-point) values, including negative values, in IBM mainframe format
"S370FIBU*w.d*" on page 716	Reads unsigned integer binary (fixed-point) values in IBM mainframe format
"S370FPD*w.d* "on page 718	Reads packed data in IBM mainframe format
"S370FPDU*w.d*" on page 719	Reads unsigned packed decimal data in IBM mainframe format
"S370FPIB*w.d*" on page 720	Reads positive integer binary (fixed-point) values in IBM mainframe format
"S370FRB*w.d*" on page 721	Reads real binary (floating-point) data in IBM mainframe format
"S370FZD*w.d* "on page 722	Reads zoned decimal data in IBM mainframe format
"S370FZDL*w.d*" on page 723	Reads zoned decimal leading-sign data in IBM mainframe format
"S370FZDS*w.d*" on page 724	Reads zoned decimal separate leading-sign data in IBM mainframe format
"S370FZDT*w.d*" on page 725	Reads zoned decimal separate trailing-sign data in IBM mainframe format
"S370FZDU*w.d*" on page 726	Reads unsigned zoned decimal data in IBM mainframe format
"VAXRB*w.d* "on page 731	Reads real binary (floating-point) data in VMS format
"*w.d*" on page 731	Reads standard numeric data
"YEN*w.d*" on page 732	Removes embedded yen signs, commas, and decimal points
"ZD*w.d* "on page 738	Reads zoned decimal data
"ZDB*w.d*" on page 739	Reads zoned decimal data in which zeros have been left blank
"ZDV*w.d*" on page 740	Reads and validates zoned decimal data

Dictionary

$ASCII*w*.

Converts ASCII character data to native format

Category: Character

Syntax

$ASCII*w*.

Syntax Description

w
specifies the width of the input field.

Default: 1 if the length of the variable is undefined; otherwise, the length of the variable

Range: 1–32767

Details

If ASCII is the native format, no conversion occurs.

Comparisons

- On an IBM mainframe system, $ASCII*w*. converts ASCII data to EBCDIC.
- On all other systems, $ASCII*w*. behaves like the $CHAR*w*. informat except that the default length is different.

Examples

```
input @1 name $ascii3.;
```

Data Lines	Results*	
----+----1	EBCDIC	ASCII
abc	818283	616263
ABC	C1C2C3	414243
();	4D5D5E	28293B

* The results are hexadecimal representations of codes for characters. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one character value.

$BINARY*w*.

Converts binary data to character data

Category: Character

Syntax

$BINARY*w*.

Syntax Description

w
specifies the width of the input field. Because eight bits of binary information represent one character, every eight characters of input that $BINARY*w*. reads becomes one character value stored in a variable.

If *w*< 8, $BINARY*w*. reads the data as *w* characters followed by 0s. Thus, $BINARY4. reads the characters 0101 as 01010000, which converts to an EBCDIC **&** or an ASCII **P**. If *w*> 8 but is not a multiple of 8, $BINARY*w*. reads up to the largest multiple of 8 that is less than *w* before converting the data.

Default: 8

Range: 1–32767

Details

The $BINARY*w*. informat does not interpret actual binary data, but it converts a string of characters that contains only 0s or 1s as though it is actual binary information. Therefore, use only the character digits 1 and 0 in the input, with no embedded blanks. $BINARY*w*. ignores leading and trailing blanks.

To read representations of binary codes for unprintable characters, enter an ASCII or EBCDIC equivalent for a particular character as a string of 0s and 1s. The $BINARY*w*. informat converts the string to its equivalent character value.

Comparisons

- The BINARY*w*. informat reads eight characters of input that contain only 0s or 1s as a binary representation of one byte of numeric data.
- The $HEX*w*. informat reads hexadecimal digits that represent the ASCII or EBCDIC equivalent of character data.

Examples

```
input @1 name $binary16.;
```

Data Lines		Results
`----+----1----+----2`	ASCII	EBCDIC
`0100110001001101`	`LM`	`<(`

$CB*w.*

Reads standard character data from column-binary files

Category: Column Binary

Syntax

$CB*w.*

Syntax Description

w
specifies the width of the input field.

Default: none

Range: 1–32767

Details

The $CB*w.* informat reads standard character data from column-binary files, with each card column represented in 2 bytes, and it translates the data into standard character codes. If the combinations are invalid punch codes, SAS returns blanks and sets the automatic variable _ERROR_ to 1.

Examples

```
input @1 name $cb2.;
```

Data Lines*		Results
`----+----1`	EBCDIC	ASCII
`200A`	`+`	`N`

* The data line is a hexadecimal representation of the column binary. The punch card column for the example data has row 12, row 6, and row 8 punched. The binary representation is 0010 0000 0000 1010.

See Also

Informats:

"CB*w*.*d*" on page 663
"PUNCH.*d*" on page 705
"ROW*w*.*d*" on page 710

See the discussion on reading column-binary data in *SAS Language Reference: Concepts*.

$CHAR*w*.

Reads character data with blanks

Category: Character

Syntax

$CHAR*w*.

Syntax Description

w
specifies the width of the input field.

Default: 8 if the length of the variable is undefined; otherwise, the length of the variable

Range: 1–32767

Details

The $CHAR*w*. informat does not trim leading and trailing blanks or convert a single period in the input data field to a blank before storing values. If you use $CHAR*w*. in an INFORMAT or ATTRIB statement within a DATA step to read list input, then by default SAS interprets any blank embedded within data as a field delimiter, including leading blanks.

Comparisons

- The $CHAR*w*. informat is almost identical to the $*w*. informat. However $CHAR*w*. does not trim leading blanks or convert a single period in the input data field to a blank, while the $*w*. informat does.
- Use the table below to compare the SAS informat $CHAR8. with notation in other programming languages:

Language	Character Notation
SAS	$CHAR8.
IBM 370 assembler	CL8
C	char [8]

Language	Character Notation
COBOL	PIC x(8)
FORTRAN	A8
PL/I	CHAR(8)

Examples

```
input @1 name $char5.;
```

Data Lines	Results*
`----+----1`	
`XYZ`	`XYZ##`
` XYZ`	`#XYZ#`
`  .`	`##.##`
`X YZ`	`#X#YZ`

* The character # represents a blank space.

$CHARZB*w*.

Converts binary 0s to blanks

Category: Character

Syntax

$CHARZB*w*.

Syntax Description

w
specifies the width of the input field.

Default: 1 if the length of the variable is undefined; otherwise, the length of the variable

Range: 1–32767

Details

The $CHARZB*w*. informat does not trim leading and trailing blanks in character data before it stores values.

Comparisons

The $CHARZB*w*. informat is identical to the $CHAR*w*. informat except that $CHARZB*w*. converts any byte that contains a binary 0 to a blank character.

Examples

```
input @1 name $charzb5.;
```

Data Lines*		Results
EBCDIC	ASCII	
E7E8E90000	**58595A0000**	**XYZ##**
00E7E8E900	**0058595A00**	**#XYZ#**
00E700E8E9	**005800595A**	**#X#YZ**

* The data lines are hexadecimal representations of codes for characters. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one character.
** The character # represents a blank space.

$EBCDIC*w*.

Converts EBCDIC character data to native format

Category: Character

Syntax

$EBCDIC*w*.

Syntax Description

w
specifies the width of the input field.

Default: 1 if the length of the variable is undefined; otherwise, the length of the variable

Range: 1–32767

Details

If EBCDIC is the native format, no conversion occurs.

Comparisons

- On an IBM mainframe system, $EBCDIC*w*. behaves like the $CHAR*w*. informat.
- On all other systems, $EBCDIC*w*. converts EBCDIC data to ASCII.

Examples

```
input @1 name $ebcdic3.
```

Data Lines	Results*	
`----+----1`	ASCII	EBCDIC
`qrs`	`717273`	`9899A2`
`QRS`	`515253`	`D8D9E2`
`+;>`	`2B3B3E`	`4E5E6E`

* The results are hexadecimal representations of codes for characters. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one character value.

$HEX*w*.

Converts hexadecimal data to character data

Category: Character

Syntax

$HEX*w*.

Syntax Description

w
specifies the number of digits of hexadecimal data.
If *w*=1, $HEX*w*. pads a trailing hexadecimal 0. If *w* is an odd number that is greater than 1, then $HEX*w*. reads *w*–1 hexadecimal characters.

Default: 2

Range: 1–32767

Details

The $HEX*w*. informat converts every two digits of hexadecimal data into one byte of character data. Use $HEX*w*. to encode hexadecimal values into a character variable when your input method is limited to printable characters.

Comparisons

The HEX*w*. informat reads two digits of hexadecimal data at a time and converts them into one byte of numeric data.

Examples

```
input @1 name $hex4.;
```

Data Lines		Results
`----+----1`	ASCII	EBCDIC
`6C6C`	`11`	`%%`

$KANJI*w*.

Removes shift code data from DBCS data

Category: DBCS

Syntax

$KANJI*w*.

Syntax Description

w
specifies the width of the input field.

Restriction: The width must be an even number. If it is an odd number, it is truncated.

Range: The minimum width for the informat is 2.

Details

The data must start with SO and end with SI, unless single-byte blank data are returned. This informat always returns a blank for the DBCSTYPE data that does not use a shift-code mechanism. The input data length must be **`2 + (SO/SI length)*2`**.

$KANJIX*w*.

Adds shift code data to DBCS data

Category: DBCS

Syntax

$KANJIX*w*.

Syntax Description

w
specifies the width of the input field.

Restriction: The width must be an even number. If it is an odd number, it is truncated.

Range: The minimum width for the informat is **2 + (length of shift code used on the current DBCSTYPE= setting)*2.**

$OCTAL*w*.

Converts octal data to character data

Category: Character

Syntax

$OCTAL*w*.

Syntax Description

w
specifies the width of the input field in bits. Because one digit of octal data represents three bits of binary information, increment the value of *w* by three for every column of octal data that $OCTAL*w*. will read.

Default: 3

Range: 1–32767

Details

Eight bits of binary data represent the code for one digit of character data. Therefore, you need at least three digits of octal data to represent one digit of character data, which includes an extra bit. $OCTAL*w*. treats every three digits of octal data as one digit of character data, ignoring the extra bit.

Use $OCTAL*w*. to read octal representations of binary codes for unprintable characters. Enter an ASCII or EBCDIC equivalent for a particular character in octal notation. Then use $OCTAL*w*. to convert it to its equivalent character value.

Use only the digits 0 through 7 in the input, with no embedded blanks. $OCTAL*w*. ignores leading and trailing blanks.

Comparisons

The OCTAL*w*. informat reads octal data and converts them into the numeric equivalents.

Examples

```
input @1 name $octal9.;
```

Data Lines	Results	
`----+----1`	EBCDIC	ASCII
`114`	`<`	`L`

$PHEX*w*.

Converts packed hexadecimal data to character data

Category: Character

Syntax

$PHEX*w*.

Syntax Description

w
specifies the number of bytes in the input.

When you use $PHEX*w*. to read packed hexadecimal data, the length of the variable is the number of bytes that are required to store the resulting character value, not *w*. In general, a character variable whose length is implicitly defined with $PHEX*w*. has a length of 2*w*–1.

Default: 2

Range: 1–32767

Details

Packed hexadecimal data are like packed decimal data, except that all hexadecimal digits are valid. In packed hexadecimal data, the value of the low-order nibble has no meaning. In packed decimal data, the value of the low-order nibble indicates the sign of the numeric value that the data represent. The $PHEX*w*. informat returns a character value and treats the value of the sign nibble as if it were `X'F'`, regardless of its actual value.

Comparisons

The PD*w.d*. informat reads packed decimal data and converts them to numeric data.

Examples

```
input @1 devaddr $phex2.;
```

Data Lines*	Results
`0001111000001111`	`1E0`

* The data line represents two bytes of actual binary data, with each half byte corresponding to a single hexadecimal digit. The equivalent hexadecimal representation for the data line is 1E0F.

$QUOTE*w*.

Removes matching quotation marks from character data

Category: Character

Syntax

$QUOTE*w*.

Syntax Description

w
specifies the width of the input field.

Default: 8 if the length of the variable is undefined; otherwise, the length of the variable

Range: 1–32767

Examples

```
input @1 name $quote7.;
```

Data Lines	Results
`----+----1`	
`'SAS'`	SAS
`"SAS"`	SAS
`"SAS's"`	SAS's

$REVERJ*w*.

Reads character data from right to left and preserves blanks

Category: Character

Syntax

$REVERJ*w*.

Syntax Description

w
specifies the width of the input field.

Default: 1 if the length of the variable is undefined; otherwise, the length of the variable

Range: 1–32767

Details

The $REVERJ*w*. informat preserves all leading and trailing blanks when it reads text right to left.

Comparisons

The $REVERJ*w*. informat is similar to the $REVERS*w*. informat except that $REVERS*w*. left aligns the result by removing all leading blanks.

Examples

```
input @1 name $reverj7.;
```

Data Lines	Results*
`----+----1`	
ABCD	**###DCBA**
ABCD	**DCBA###**

* The character # represents a blank space.

$REVERS*w*.

Reads character data from right to left and left aligns

Category: Character

Syntax

$REVERS*w*.

Syntax Description

w
specifies the width of the input field.

Default: 1 if the length of the variable is undefined; otherwise, the length of the variable

Range: 1–32767

Comparisons

The $REVERS*w*. informat is similar to the $REVERJ*w*. informat except that $REVERJ*w*. preserves all leading and trailing blanks.

Examples

```
input @1 name $revers7.;
```

Data Lines	Results*
----+----1	
ABCD	**DCBA###**
ABCD	**DCBA###**

* The character # represents a blank space.

$UPCASE*w*.

Converts character data to uppercase

Category: Character

Syntax

$UPCASE*w*.

Syntax Description

w
specifies the width of the input field.

Default: 8 if the length of the variable is undefined; otherwise, the length of the variable

Range: 1–32767

Details

Special characters, such as hyphens, are not altered.

Examples

```
input @1 name $upcase3.;
```

Data Lines	Results
`----+----1`	
sas	**SAS**

$VARYING*w*.

Reads character data of varying length

Valid: in a DATA step

Category: Character

Syntax

$VARYING*w. length-variable*

Syntax Description

w
specifies the maximum width of a character field for all the records in an input file.

Default: 8 if the length of the variable is undefined; otherwise, the length of the variable

Range: 1–32767

length-variable
specifies a numeric variable that contains the width of the character field in the current record. SAS obtains the value of *length-variable* by reading it directly from a field that is described in an INPUT statement or by calculating its value in the DATA step.

Requirement: You must specify *length-variable* immediately after $VARYING*w*. in an INPUT statement.

Restriction: *Length-variable* cannot be an array reference.

Tip: If *length-variable* is less than 1 or is missing, SAS reads no data from the corresponding record. This enables you to read both zero-length records and fields. If *length-variable* is greater than 0 but less than *w*, SAS reads the number of columns that are specified by *length-variable*. Then SAS pads the value with trailing blanks up to the maximum width that is assigned to the variable. If *length-variable* is greater than or equal to *w*, SAS reads *w* columns.

Details

Use $VARYING*w*. when the length of a character value differs from record to record. After reading a data value with $VARYING*w*., the pointer's position is set to the first column after the value.

Examples

Example 1: Obtaining a Current Record Length Directly

```
input fwidth 1. name $varying9. fwidth;
```

Data Lines*	Results
----+----1	
5shark	**shark**
3sunfish	**sun**
8bluefish	**bluefish**

* Notice the result of reading the second data line.

Example 2: Obtaining a Record Length Indirectly Use the LENGTH= option in the INFILE statement to obtain a record length indirectly. The input data lines and results follow the explanation of the SAS statements.

```
data one;
   infile file-specification length=reclen;
   input @;
   fwidth=reclen-9;
   input name $ 1-9
         @10 class $varying20. fwidth;
run;
```

The LENGTH= option in the INFILE statement assigns the internally stored record length to RECLEN when the first INPUT statement executes. The trailing @ holds the record for another INPUT statement. Next, the assignment statement calculates the value of the varying-length field by subtracting the fixed-length portion of the record from the total record length. The variable FWIDTH contains the length of the last field and becomes the *length-variable* argument to the $VARYING20. informat.

Data Lines	Results
----+----1----+----2	
PATEL CHEMISTRY	**PATEL CHEMISTRY**

Data Lines	Results
JOHNSON GEOLOGY	**JOHNSON GEOLOGY**
WILCOX ART	**WILCOX ART**

$w.

Reads standard character data

Category: Character

Syntax

$*w*.

Syntax Description

w
specifies the width of the input field. You must specify *w* because SAS does not supply a default value.

Range: 1–32767

Details

The $*w*. informat trims leading blanks and left aligns the values before storing the text. In addition, if a field contains only blanks and a single period, $*w*. converts the period to a blank because it interprets the period as a missing value. The $*w*. informat treats two or more periods in a field as character data.

Comparisons

The $*w*. informat is almost identical to the $CHAR*w*. informat. However, $CHAR*w*. does not trim leading blanks nor does it convert a single period in an input field to a blank, while $*w*. does both.

Examples

```
input @1 name $5.;
```

Data Lines	Results*
----+----1	
XYZ	**XYZ##**
XYZ	**XYZ##**

Data Lines	Results*
.	
`X YZ`	`X#YZ#`

* The character # represents a blank space.

BINARY*w.d*

Converts positive binary values to integers

Category: Numeric

Syntax

BINARY*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 8

Range: 1–64

d
optionally specifies the power of 10 by which to divide the value. SAS uses the *d* value even if the data contain decimal points.

Range: 0–31

Details

Use only the character digits 1 and 0 in the input, with no embedded blanks. BINARY*w.d* ignores leading and trailing blanks.

BINARY*w.d* cannot read negative values. It treats all input values as positive (unsigned).

Examples

```
input @1 value binary8.1;
```

Data Lines	Results
`----+----1----+`	
`00001111`	`1.5`

BITS*w.d*

Extracts bits

Category: Numeric

Syntax

BITS*w.d*

Syntax Description

w
specifies the number of bits to read.
Default: 1
Range: 1–64

d
specifies the zero-based offset.
Range: 0–63

Details

The BITS*w.d* informat extracts particular bits from an input stream and assigns the numeric equivalent of the extracted bit string to a variable. Together, the *w* and *d* values specify the location of the string you want to read.

This informat is useful for extracting data from system records that have many pieces of information packed into single bytes.

Examples

```
input @1 value bits4.1;
```

Data Lines	Results*
`----+----1----+`	
`B`	`8`

* The EBCDIC binary code for a capital B is 11000010, and the ASCII binary code is 01000010.

The input pointer moves to column 2 (*d*=1). Then the INPUT statement reads four bits (*w*=4) which is the bit string 1000 and stores the numeric value 8, which is equivalent to this binary combination.

BZ*w.d*

Converts blanks to 0s

Category: Numeric

Syntax

BZ*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 1

Range: 1–32

d
optionally specifies the power of 10 by which to divide the value. If the data contain decimal points, the *d* value is ignored.

Range: 0–31

Details

The BZ*w.d* informat reads numeric values, converts any trailing or embedded blanks to 0s, and ignores leading blanks.

The BZ*w.d* informat can read numeric values that are located anywhere in the field. Blanks can precede or follow the numeric value, and a minus sign must precede negative values. The BZ*w.d* informat ignores blanks between a minus sign and a numeric value in an input field.

The BZ*w.d* informat interprets a single period in a field as a 0. The informat interprets multiple periods or other nonnumeric characters in a field as a missing value.

To use BZ*w.d* in a DATA step with list input, change the delimiter for list input with the DLM= option in the INFILE statement. By default, SAS interprets blanks between values in the data line as delimiters rather than 0s.

Comparisons

The BZ*w.d* informat converts trailing or embedded blanks to 0s. If you do not want to convert trailing blanks to 0s (for example, when reading values in E-notation), use either the *w.d* informat or the E*w.d* informat instead.

Examples

```
input @1 x bz4.;
```

Data Lines	Results
`----+----1`	
`34`	`3400`
`-2`	`-200`
`-2 1`	`-201`

CB*w.d*

Reads standard numeric values from column-binary files

Category: Column Binary

Syntax

CB*w.d*

Syntax Description

w
specifies the width of the input field.

Range: 1–32

d
optionally specifies the power of 10 by which to divide the value. SAS uses the *d* value even if the data contain decimal points.

Details

The CB*w.d* informat reads standard numeric values from column-binary files and translates the data into standard binary format.

SAS first stores each column of column-binary data you read with CB*w.d* in two bytes and ignores the two high-order bits of each byte. If the punch codes are valid, SAS stores the equivalent numeric value into the variable that you specify. If the combinations are not valid, SAS assigns the variable a missing value and sets the automatic variable _ERROR_ to 1.

Examples

```
input @1 x cb8.;
```

Data Lines*	Results
----+----1	
0009	**9**

* The data line is a hexadecimal representation of the column binary. The punch card column for the example data has row 9 punched. The binary representation is 0000 0000 0000 1001.

See Also

Informats:

"$CB*w*." on page 646
"PUNCH.*d*" on page 705
"ROW*w*.*d*" on page 710

COMMA*w.d*

Removes embedded characters

Category: Numeric

Syntax

COMMA*w.d*

Syntax Description

w
specifies the width of the input field.
Default: 1
Range: 1–32

d
optionally specifies the power of 10 by which to divide the value. If the data contain decimal points, the *d* value is ignored.
Range: 0–31

Details

The COMMA*w.d* informat reads numeric values and removes embedded commas, blanks, dollar signs, percent signs, dashes, and right parentheses from the input data. The COMMA*w.d* informat converts a left parenthesis at the beginning of a field to a minus sign.

Comparisons

The COMMA*w.d* informat operates like the COMMAX*w.d* informat, but it reverses the roles of the decimal point and the comma. This convention is common in European countries.

Examples

```
input @1 x comma10.;
```

Data Lines	Results
`----+----1----+`	
`$1,000,000`	`1000000`
`(500)`	`-500`

COMMAX*w.d*

Removes embedded characters

Category: Numeric

Syntax

COMMAX*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 1

Range: 1–32

d
optionally specifies the power of 10 by which to divide the value. If the data contain a comma, which represents a decimal point, the *d* value is ignored.

Range: 0–31

Details

The COMMAX*w.d* informat reads numeric values and removes embedded periods, blanks, dollar signs, percent signs, dashes, and right parentheses from the input data. The COMMAX*w.d* informat converts a left parenthesis at the beginning of a field to a minus sign.

Comparisons

The COMMAX*w.d* informat operates like the COMMA*w.d* informat, but it reverses the roles of the decimal point and the comma. This convention is common in European countries.

Examples

```
input @1 x commax10.;
```

Data Lines	Results
`----+----1----+`	
`$1.000.000`	`1000000`
`(500)`	`-500`

DATE*w*.

Reads date values in the form *ddmmmyy* or *ddmmmyyyy*

Category: Date and Time

Syntax

DATE*w*.

Syntax Description

w
specifies the width of the input field.
Default: 7
Range: 7–32
Tip: Use a width of 9 to read a 4–digit year.

Details

The date values must be in the form *ddmmmyy* or *ddmmmyyyy*, where

dd
is an integer from 01 through 31 that represents the day of the month.

mmm
is the first three letters of the month name.

yy or *yyyy*
is a two- or four-digit integer that represents the year.

You can separate the year, month, and day values by blanks or by special characters. Make sure the width of the input field allows space for blanks and special characters.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

Examples

```
input calendar_date date11.;
```

Data Lines	Results
----+----1----+	
16mar99	**14309**
16 mar 99	**14309**
16-mar-1999	**14309**

See Also

Format:
"DATE*w*." on page 89
Function:
"DATE" on page 312
System Option:
"YEARCUTOFF=" on page 1177

DATETIME*w*.

Reads datetime values in the form *ddmmmyy hh:mm:ss.ss* or *ddmmmyyyy hh:mm:ss.ss*

Category: Date and Time

Syntax

DATETIME*w*.

Syntax Description

w
specifies the width of the input field.
Default: 18
Range: 13–40

Details

The datetime values must be in the following form: *ddmmmyy* or *ddmmmyyyy*, followed by a blank or special character, followed by *hh:mm:ss.ss* (the time). In the date,

dd
is an integer from 01 through 31 that represents the day of the month.

mmm
is the first three letters of the month name.

yy or *yyyy*
is a two- or four-digit integer that represents the year.

In the time,

hh
is the number of hours ranging from 00 through 23.

mm
is the number of minutes ranging from 00 through 59.

ss.ss
is the number of seconds ranging from 00 through 59 with the fraction of a second following the decimal point.

DATETIME*w*. requires values for both the date and the time; however, the *ss.ss* portion is optional.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

Note: SAS can read time values with AM and PM in them. △

Examples

```
input date_and_time datetime20.;
```

Data Lines	Results
`----+----1----+----2`	
`16mar97:11:23:07.4`	`1237202587.4`

Data Lines	Results
`16mar1997/11:23:07.4`	`1237202587.4`
`16mar1997/11:23 PM`	`1237245780.0`

See Also

Formats:

"DATE*w*." on page 89
"DATETIME*w*.*d*" on page 91
"TIME*w*.*d*" on page 172

Function:

"DATETIME" on page 315

Informats:

"DATE*w*." on page 666
"TIME*w*." on page 727

System Option:

"YEARCUTOFF=" on page 1177

See the discussion on using SAS date and time values in *SAS Language Reference: Concepts*

DDMMYY*w*.

Reads date values in the form *ddmmyy* or *ddmmyyyy*

Category: Date and Time

Syntax

DDMMYY*w*.

Syntax Description

w
specifies the width of the input field.
Default: 6
Range: 6–32

Details

The date values must be in the form *ddmmyy* or *ddmmyyyy*, where

dd
is an integer from 01 through 31 that represents the day of the month.

mm
is an integer from 01 through 12 that represents the month.

yy or *yyyy*
is a two- or four-digit integer that represents the year.

You can place blanks and other special characters between day, month, and year values. However, if you use delimiters, place them between all the values. Blanks can also be placed before and after the date. Make sure the width of the input field allows space for blanks and special characters.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

Examples

```
input calendar_date ddmmyy10.;
```

Data Lines	Results
----+----1----+	
160399	14319
16/03/99	14319
16 03 1999	14319

See Also

Formats:
- "DATE*w*." on page 89
- "DDMMYY*w*." on page 94
- "MMDDYY*w*." on page 131
- "YYMMDD*w*." on page 188

Function:
- "MDY" on page 443

Informats:
- "DATE*w*." on page 666
- "MMDDYY*w*." on page 687
- "YYMMDD*w*." on page 733

System Option:
- "YEARCUTOFF=" on page 1177

Ew.d

Reads numeric values that are stored in scientific notation and double-precision scientific notation

Category: Numeric

Syntax

E*w.d*

Syntax Description

w
specifies the width of the field that contains the numeric value.

Default: 12

Range: 1–32

d
optionally specifies the number of digits to the right of the decimal point in the numeric value. If the data contain decimal points, the *d* value is ignored.

Range: 0–31

Comparisons

The E*w.d* informat is not used extensively because the SAS informat for standard numeric data, the *w.d* informat, can read numbers in scientific notation. Use E*w.d* to permit only scientific notation in your input data.

Examples

```
input @1 x e7.;
```

Data Lines	Results
`----+----1----+`	
`1.257E3`	`1257`
`12d3`	`12000`

EURDFDE*w*.

Reads international date values

Category: Date and Time

Syntax

EURDFDE*w*.

w
specifies the width of the input field.

Default: 7 (except Finnish)

Range: 7–32 (except Finnish)

Note: If you use the Finnish (FIN) language prefix, the *w* range is 10–32 and the default *w* is 10. △

Details

The date values must be in the form *ddmmmyy* or *ddmmmyyyy*, where

dd
is an integer from 01 through 31 that represents the day of the month.

mmm
is the first three letters of the month name.

yy or *yyyy*
is a two- or four-digit integer that represents the year.

You can place blanks and other special characters between day, month, and year values.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

You can set the language for the SAS session with the DFLANG= system option. (Because the SAS Installation Representative usually sets a default language for the site, you may be able to skip this step.) If you work with dates in multiple languages, you can replace the EUR prefix with a language prefix. See “DFLANG=” on page 1085 for the list of language prefixes on page 1086. When you specify the language prefix in the informat, SAS ignores the DFLANG= system option.

Examples

This INPUT statement uses the value of the DFLANG= system option to read the international date values in Spanish.

```
options dflang=spanish;
input day eurdfde10.;
```

This INPUT statement uses the Spanish language prefix in the informat to read the international date values in Spanish. The value of the DFLANG= option, therefore, is ignored.

```
input day espdfde10.;
```

Data Lines	Results
`----+----1----+`	
`01abr1999`	`14335`
`01-abr-99`	`14335`

See Also

Format:

"EURDFDE*w*." on page 103

Function:

"DATE" on page 312

Informats:

"DATE*w*." on page 666
"EURDFDT*w*." on page 673
"EURDFMY*w*. "on page 675

System Options:

"DFLANG=" on page 1085
"YEARCUTOFF=" on page 1177

EURDFDT*w*.

Reads international datetime values in the form *ddmmmyy hh:mm:ss.ss* or *ddmmmyyyy hh:mm:ss.ss*

Category: Date and Time

Syntax

EURDFDT*w*.

Syntax Description

w
specifies the width of the input field.

Default: 18

Range: 13–40

Details

The datetime values must be in the form *ddmmmyy* or *ddmmmyyyy*, followed by a blank or special character, and *hh:mm:ss.ss*. In the date,

dd
is an integer from 01 through 31 that represents the day of the month.

mmm
is the first three letters of the month name.

yy or *yyyy*
is a two- or four-digit integer that represents the year.

In the time,

hh
is the number of hours ranging from 00 through 23,

mm
is the number of minutes ranging from 00 through 59,

ss.ss
is the number of seconds ranging from 00 through 59 with the fraction of a second following the decimal point.

EURDFDT*w*. requires values for both the date and the time; however, the *ss.ss* portion is optional.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

You can set the language for the SAS session with the DFLANG= system option. (Because the SAS Installation Representative usually sets a default language for the site, you may be able to skip this step.) If you work with dates in multiple languages, you can replace the EUR prefix with a language prefix. See “DFLANG=” on page 1085 for the list of language prefixes on page 1086. When you specify the language prefix in the informat, SAS ignores the DFLANG= system option.

Examples

This INPUT statement uses the value of the DFLANG= system option to read the international datetime values in German.

```
options dflang=german;
input date eurdfdt20.;
```

This INPUT statement uses the German language prefix to read the international datetime values in German. The value of the DFLANG= option, therefore, is ignored.

```
input date deudfdt20.;
```

Data Lines	Results
----+----1----+----2	
23dez99:10:03:17.2	**1261562597.2**
23dez1999:10:03:17.2	**1261562597.2**

See Also

Formats:

"DATE*w*." on page 89
"DATETIME*w*.*d*" on page 91
"TIME*w*.*d*" on page 172

Function:

"DATETIME" on page 315

Informats:

"DATETIME*w*. "on page 667
"EURDFDE*w*. "on page 671
"EURDFMY*w*. "on page 675

System Options:

"DFLANG=" on page 1085
"YEARCUTOFF=" on page 1177

EURDFMY*w*.

Reads month and year date values in the form *mmmyy* or *mmmyyyy*

Category: Date and Time

Syntax

EURDFMY*w*.

Syntax Description

w
specifies the width of the input field.

Default: 5 (except Finnish)

Range: 5–32 (except Finnish)

Note: If you use the Finnish (FIN) language prefix, the *w* range is 7–32 and the default value for *w* is 7. △

Details

The date values must be in the form *mmmyy* or *mmmyyyy*, where

mmm
is the first three letters of the month name.

yy or *yyyy*
is a two- or four-digit integer that represents the year.

You can place blanks and other special characters between day, month, and year values. A value that is read with EURDFMY*w*. results in a SAS date value that corresponds to the first day of the specified month.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

You can set the language for the SAS session with the DFLANG= system option. (Because the SAS Installation Representative usually sets a default language for the site, you may be able to skip this step.) If you work with dates in multiple languages, you can replace the EUR prefix with a language prefix. See "DFLANG=" on page 1085 for the list of language prefixes on page 1086. When you specify the language prefix in the informat, SAS ignores the DFLANG= option.

Examples

This INPUT statement uses the value of DFLANG= system option to read the international date values in French.

```
options dflang=french;
input month eurdfmy7.;
```

The second INPUT statement uses the French language prefix, and DFLANG is not specified.

```
input month fradfmy7.;
```

Data Lines	Results
`----+----1`	
avr1999	**14335**
avr 99	**14335**

See Also

Formats:

"DDMMYY*w*." on page 94
"MMDDYY*w*." on page 131
"MONYY*w*." on page 138
"YYMMDD*w*." on page 188

Functions:

"MONTH" on page 451
"YEAR" on page 618

Informats:

"EURDFDE*w*. "on page 671
"EURDFDT*w*." on page 673
"MONYY*w*." on page 689

System Options:

"DFLANG=" on page 1085
"YEARCUTOFF=" on page 1177

FLOAT*w.d*

Reads a native single-precision, floating-point value and divides it by 10 raised to the *d*th power

Category: Numeric

Syntax

FLOAT*w.d*

Syntax Description

w
specifies the width of the input field.
Requirement: *w* must be 4.

d
optionally specifies the power of 10 by which to divide the value.

Details

The FLOAT*w.d* informat is useful in operating environments where a float value is not the same as a truncated double.

On the IBM mainframe systems, a four-byte floating-point number is the same as a truncated eight-byte floating-point number. However, in operating environments that use the IEEE floating-point standard, such as the IBM PC-based operating environments and most UNIX platforms, a four-byte floating-point number is not the same as a truncated double. Therefore, the RB4. informat does not produce the same

results as FLOAT4. Floating-point representations other than IEEE may have this same characteristic. Values read with FLOAT4. typically come from some other external program that is running in your operating environment.

Comparisons

The following table compares the names of float notation in several programming languages:

Language	Float Notation
SAS	FLOAT4.
FORTRAN	REAL*4
C	float
IBM 370 ASM	E
PL/I	FLOAT BIN(21)

Examples

```
input x float4.;
```

Data Lines*	Results
`----+----1----+----2`	
`3F800000`	`1`

* The data line is a hexadecimal representation of a binary number that is stored in IEEE form.

HEX*w.*

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

Category: Numeric

Syntax

HEX*w.*

Syntax Description

w
specifies the field width of the input value and also specifies whether the final value is fixed-point or floating-point.

Default: 8

Range: 1–16

Tip: If *w*<16, HEX*w*. converts the input value to positive integer binary values, treating all input values as positive (unsigned). If *w* is 16, HEX*w*. converts the input value to real binary (floating-point) values, including negative values.

Details

Note: Different operating environments store floating-point values in different ways. However, HEX16. reads hexadecimal representations of floating-point values with consistent results if the values are expressed in the same way that your operating environment stores them. △

The HEX*w*. informat ignores leading or trailing blanks.

Examples

```
input @1 x hex3. @5 y hex16.;
```

Data Lines*	Results
`----+----1----+----2`	
`88F 4152000000000000`	`2191 5.125`

* The data line shows IBM mainframe hexadecimal data.

IB*w.d*

Reads native integer binary (fixed-point) values, including negative values

Category: Numeric

Syntax

IB*w*.*d*

Syntax Description

w
specifies the width of the input field.
Default: 4
Range: 1–8

d
optionally specifies the power of 10 by which to divide the value.
Range: 0–10

Details

The IB*w.d* informat reads integer binary (fixed-point) values, including negative values represented in two's complement notation. IB*w.d* reads integer binary values with

consistent results if the values are created in the same type of operating environment that you use to run SAS.

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see "Byte Ordering on Big Endian and Little Endian Platforms" on page 634. △

Comparisons

The IB*w.d* and PIB*w.d* informats are used to read native format integers. (Native format allows you to read and write values created in the same operating environment.) The IBR*w.d* and PIBR*w.d* informats are used to read little endian integers in any operating environment.

To view a table that shows the type of informat to use with big endian and little endian integers, see Table 5.1 on page 635.

To view a table that compares integer binary notation in several programming languages, see Table 5.2 on page 636.

Examples

You can use the INPUT statement and specify the IB informat. However, these examples use the informat with the INPUT function, where binary input values are described using a hex literal.

```
x=input('0080'x,ib2.);
y=input('8000'x,ib2.);
```

SAS Statements	Results on Big Endian Platforms	Results on Little Endian Platforms
put x=;	**128**	**-32768**
put y=;	**-32768**	**128**

See Also

Informat:

"IBR*w.d*" on page 680

IBR*w.d*

Reads integer binary (fixed-point) values in Intel and DEC formats

Category: Numeric

Syntax

IBR*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 4

Range: 1–8

d
optionally specifies the power of 10 by which to divide the value.

Range: 0–10

Details

The IBR*w.d* informat reads integer binary (fixed-point) values, including negative values that are represented in two's complement notation. IBR*w.d* reads integer binary values that are generated by and for Intel and DEC platforms. Use IBR*w.d* to read integer binary data from Intel or DEC environments in other operating environments. The IBR*w.d* informat in SAS code allows for a portable implementation for reading the data in any operating environment.

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see "Byte Ordering on Big Endian and Little Endian Platforms" on page 634. △

Comparisons

The IB*w.d* and PIB*w.d* informats are used to read native format integers. (Native format allows you to read and write values that are created in the same operating environment.) The IBR*w.d* and PIBR*w.d* informats are used to read little endian integers in any operating environment.

On Intel and DEC operating environments, the IB*w.d* and IBR*w.d* informats are equivalent.

To view a table that shows the type of informat to use with big endian and little endian integers, see Table 5.1 on page 635.

To view a table that compares integer binary notation in several programming languages, see Table 5.2 on page 636.

Examples

You can use the INPUT statement and specify the IBR informat. However, in these examples we use the informat with the INPUT function, where binary input values are described using a hex literal.

```
x=input('0100'x,ibr2.);
y=input('0001'x,ibr2.);
```

SAS Statements	Results on Big Endian Platforms	Results on Little Endian Platforms
put x=;	**1**	**1**
put y=;	**256**	**256**

See Also

Informat:

"IB*w.d*" on page 679

IEEE*w.d*

Reads an IEEE floating-point value and divides it by 10 raised to the *d* th power

Category: Numeric

Syntax

IEEE*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 8

Range: 2–8

Tip: If *w* is 8, an IEEE double-precision, floating-point number is read. If *w* is 5, 6, or 7, an IEEE double-precision, floating-point number is read, which assumes truncation of the appropriate number of bytes. If *w* is 4, an IEEE single-precision, floating-point number is read. If *w* is 3, an IEEE single-precision, floating-point number is read, which assumes truncation of one byte.

d
specifies the power of 10 by which to divide the value.

Details

The IEEE*w.d* informat is useful in operating environments where IEEE is the floating-point representation that is used. In addition, you can use the IEEE*w.d* informat to read files that are created by programs on operating environments that use the IEEE floating point representation.

Typically, programs generate IEEE values in single precision (4 bytes) or double precision (8 bytes). Truncation is performed by programs solely to save space on output files. Machine instructions require that the floating-point number be of one of the two lengths. The IEEE*w.d* informat allows other lengths, which enables you to read data from files that contain space-saving truncated data.

Examples

```
input test1 ieee4.;
input test2 ieee5.;
```

Data Lines*	Results
`----+----1----+`	
`3F800000`	`1`
`3FF0000000`	`1`

* The data lines are hexadecimal representations of binary numbers that are stored in IEEE format.

The first INPUT statement reads the first data line, and the second INPUT statement reads the next data line.

JDATEYMD*w*.

Reads Japanese kanji date values in the format *yymmmdd* or *yyyymmmdd*

Category: Date and Time

Syntax

JDATEYMD*w*.

Syntax Description

w
specifies the width of the input field.

Default: 12

Range: 12–32

Details

The date values must be in the form *yymmmdd* or *yyyymmmdd*.

You can separate the year, month, and day values by blanks or by special characters. Note that in the example, the date values in the datalines are separated by special characters.

When using this informat, make sure that the width of the input field allows space for blanks and special characters.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

Examples

The following examples show how to use the JDATEYMD informat to convert Kanji values to SAS date values.

See Also

"JNENGO*w*." on page 684

JNENGO*w*.

Reads Japanese Kanji date values in the form *yymmdd*

Category: Date and Time
Alignment: left

Syntax

JNENGO*w*.

Syntax Description

w
specifies the width of the output field.
Default: 16
Range: 16–32

Details

The JNENGO*w*. informat reads Japanese Kanji values in the form *yymmdd*.

You can separate the year, month, and day values by blanks or by special characters. Note that in the example, the date values in the datalines are separated by special characters.

When using this informat, make sure that the width of the input field allows space for blanks and special characters.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

Examples

The following examples show how to use the JNENGO informat to convert Kanji values to SAS date values.

```
data _null_;
   input x jnengo.;
   datalines;
明治1年4月6日
明治45年7月29日
大正1年7月30日
大正15年12月24日
昭和1年12月25日
昭和64年 1月 7日
平成1年1月8日
平成10年12月 8日
;
```

See Also

Informat:

"JDATEYMD*w*." on page 683

JULIAN*w*.

Reads Julian dates in the form *yyddd* or *yyyyddd*

Category: Date and Time

Syntax

JULIAN*w*.

Syntax Description

w
specifies the width of the input field.

Default: 5

Range: 5–32

Details

The date values must be in the form *yyddd* or *yyyyddd*, where

yy or *yyyy*
is a two- or four-digit integer that represents the year.

dd or *ddd*
is an integer from 01 through 365 that represents the day of the year.

Julian dates consist of strings of contiguous numbers, which means that zeros must pad any space between the year and the day values.

Julian dates that contain year values before 1582 are invalid for the conversion to Gregorian dates.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

Examples

```
input julian_date julian7.;
```

Data Lines	Results*
`----+----1`	
`99075`	`14319`
`1999075`	`14319`

* The input values correspond to the seventy-fifth day of 1999, which is March 16.

See Also

Format:

"JULIAN*w*." on page 129

Functions:

"DATEJUL" on page 313

"JULDATE" on page 416

System Option:

"YEARCUTOFF=" on page 1177

MINGUO*w*.

Reads dates in Taiwanese form

Category: Date and Time

Syntax

MINGUO*w*.

Syntax Description

w

specifies the width of the input field.

Default: 6
Range: 6–10

Details

The general form of a Taiwanese date is *yyyymmdd*, where

yyyy
is an integer that represents the year.

mm
is an integer from 01 through 12 that represents the month.

dd
is an integer from 01 through 31 that represents the day of the month.

The Taiwanese calendar uses 1912 as the base year (01/01/01 is January , 1912). Dates prior to 1912 are not valid. Year values do not roll around after 100 years; instead, they continue to increase.

You can separate the year, month, and day values with any delimiters, such as blanks, slashes, or dashes, that are permitted by the YYMMDD*w*. informat. If delimiters are used, place them between all the values. If you omit delimiters, be sure to use a leading zero for days or months less than 10.

Examples

```
input date minguo10.;
put date date9.;
```

Data Lines	Results
`----+----1----+`	
49/01/01	**01JAN1960**
891215	**15DEC2000**
103-01-01	**01JAN2014**

MMDDYY*w*.

Reads date values in the form *mmddyy* or *mmddyyyy*

Category: Date and Time

Syntax

MMDDYY*w*.

Syntax Description

w
specifies the width of the input field.

Default: 6

Range: 6–32

Details

The date values must be in the form *mmddyy* or *mmddyyyy*, where

mm
is an integer from 01 through 12 that represents the month.

dd
is an integer from 01 through 31 that represents the day of the month.

yy or *yyyy*
is a two- or four-digit integer that represents the year.

You can separate the month, day, and year fields by blanks or by special characters. However, if you use delimiters, place them between all fields in the value. Blanks can also be placed before and after the date.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

Examples

```
input calendar_date mmddyy8.;
```

Data Lines	Results
`----+----1----+`	
`031699`	`14319`
`03/16/99`	`14319`
`03 16 99`	`14319`

See Also

Formats:

- “DATE*w*.” on page 89
- “DDMMYY*w*.” on page 94
- “MMDDYY*w*.” on page 131
- “YYMMDD*w*.” on page 188

Functions:

- “DAY” on page 315
- “MDY” on page 443
- “MONTH” on page 451
- “YEAR” on page 618

Informats:

- “DATE*w*.” on page 666
- “DDMMYY*w*.” on page 669
- “YYMMDD*w*.” on page 733

System Option:

- “YEARCUTOFF=” on page 1177

MONYY*w*.

Reads month and year date values in the form *mmmyy* or *mmmyyyy*

Category: Date and Time

Syntax

MONYY*w*.

Syntax Description

w
specifies the width of the input field.
Default: 5
Range: 5–32

Details

The date values must be in the form *mmmyy* or *mmmyyyy*, where

mmm
is the first three letters of the month name.

yy or *yyyy*
is a two- or four-digit integer that represents the year.

A value read with the MONYY*w*. informat results in a SAS date value that corresponds to the first day of the specified month.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOF= system option. △

Examples

```
input month_and_year monyy7.;
```

Data Lines	Results
----+----1----+	
mar 99	14304
mar1999	14304

See Also

Formats:
- "DDMMYY*w*." on page 94
- "MMDDYY*w*." on page 131
- "MONYY*w*." on page 138
- "YYMMDD*w*." on page 188

Functions:
- "MONTH" on page 451
- "YEAR" on page 618

Informats:
- "DDMMYY*w*." on page 669
- "MMDDYY*w*." on page 687
- "YYMMDD*w*." on page 733

System Option:
- "YEARCUTOFF=" on page 1177

MSEC*w*.

Reads TIME MIC values

Category: Date and Time

Syntax

MSEC*w*.

Syntax Description

w
specifies the width of the input field.

Requirement: *w* must be 8 because the OS TIME macro or the STCK System/370 instruction on IBM mainframes each return an eight-byte value.

Details

The MSEC*w*. informat reads time values that are produced by IBM mainframe operating environments and converts the time values to SAS time values.

Use the MSEC*w*. informat to find the difference between two IBM mainframe TIME values, with precision to the nearest microsecond.

Comparisons

The MSEC*w*. and TODSTAMP*w*. informats both read IBM time-of-day clock values, but the MSEC*w*. informat assigns a time value to a variable, and the TODSTAMP*w*. informat assigns a datetime value.

Examples

```
input btime msec8.;
```

Data Lines*	Results
0000EA044E65A000	**62818.412122**

* The data line is a hexadecimal representation of a binary 8-byte time-of-day clock value. Each byte occupies one column of the input field. The result is a SAS time value corresponding to 5:26:58.41 PM.

See Also

Informat:

"TODSTAMP*w*." on page 729

NENGO*w*.

Reads Japanese date values in the form *eyymmdd*

Category: Date and Time

Syntax

NENGO*w*.

Syntax Description

w
specifies the width of the input field.
Default: 10
Range: 7–32

Details

The general form of a Japanese date is *eyymmdd*, where

e
is the first letter of the name of the emperor (Meiji, Taisho, Showa, or Heisei).

yy
is an integer that represents the year.

mm
is an integer from 01 through 12 that represents the month.

dd
is an integer from 01 through 31 that represents the day of the month.

The *e* value can be separated from the integers by a period. If you omit *e*, SAS uses the current emperor. You can separate the year, month, and day values by blanks or any nonnumeric character. However; if delimiters are used, place them between all the values. If you omit delimiters, be sure to use a leading zero for days or months that are less than 10.

Examples

```
input nengo_date nengo8.;
put nengo_date date9.;
```

Data Lines	Results
`----+----1----+`	
h11108	**08OCT1999**

Data Lines	Results
`h.11108`	`08OCT1999`
`11/10/08`	`08OCT1999`

See Also

Format:

"NENGO*w*." on page 140.

NUMX*w.d*

Reads numeric values with a comma in place of the decimal point

Category: Numeric

Syntax

NUMX*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 12

Range: 1–32

d

optionally specifies the number of digits to the right of the decimal. If the data contain decimal points, the *d* value is ignored.

Range: 0–31

Details

The NUMX*w.d* informat reads numeric values and interprets a comma as a decimal point.

Comparisons

The NUMX*w.d* informat is similar to the *w.d* informat except that it reads numeric values that contain a comma in place of the decimal point.

Examples

```
input @1 x numx10.;
```

Data Lines	Results
`----+----1----+`	
`896,48`	`896.48`
`3064,1`	`3064.1`
`6489`	`6489`

See Also

Formats:

"NUMX*w.d*" on page 141
" *w.d*" on page 176

OCTAL*w.d*

Converts positive octal values to integers

Category: Numeric

Syntax

OCTAL*w.d*

Syntax Description

w
specifies the width of the input field.
Default: 3
Range: 1–24

d
optionally specifies the power of 10 by which to divide the value.
Range: 1–31
Restriction: must be greater than or equal to the *w* value.

Details

Use only the digits 0 through 7 in the input, with no embedded blanks. The OCTAL*w.d* informat ignores leading and trailing blanks.

OCTAL*w.d* cannot read negative values. It treats all input values as positive (unsigned).

Examples

```
input @1 value octal3.1;
```

Data Lines	Results
`----+----1`	
`177`	`12.7`

PD*w.d*

Reads data that are stored in IBM packed decimal format

Category: Numeric

Syntax

PD*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 1

Range: 1–16

d
optionally specifies the power of 10 by which to divide the value.

Range: 0–10

Details

The PD*w.d* informat is useful because many programs write data in packed decimal format for storage efficiency, fitting two digits into each byte and using only a half byte for a sign.

Note: Different operating environments store packed decimal values in different ways. However, PD*w.d* reads packed decimal values with consistent results if the values are created on the same type of operating environment that you use to run SAS. △

Comparisons

The following table compares packed decimal notation in several programming languages:

Language	Notation
SAS	PD4.
COBOL	COMP-3 PIC S9(7)
IBM 370 Assembler	PL4
PL/I	FIXED DEC

Examples

Example 1: Reading Packed Decimal Data

```
input @1 x pd4.;
```

Data Lines*	Results
`----+----1`	
`0000128C`	`128`

* The data line is a hexadecimal representation of a binary number stored in packed decimal form. Each byte occupies one column of the input field.

Example 2: Creating a SAS Date with Packed Decimal Data

```
input mnth pd4.;
date=input(put(mnth,6.),mmddyy6.);
```

Data Lines*	Results
`----+----1`	
`0122599C`	`14603`

* The data line is a hexadecimal representation of a binary number that is stored in packed decimal form on an IBM mainframe operating environment. Each byte occupies one column of the input field. The result is a SAS date value that corresponds to December 25, 1999.

PDJULG*w*.

Reads packed Julian date values in the hexadecimal form *yyyyddd*F for IBM

Category: Date and Time

Syntax

PDJULG*w*.

Syntax Description

w
specifies the width of the input field.

Default: 4
Range: 4

Details

The PDJULG*w*. informat reads IBM packed Julian date values in the form of *yyyyddd*F, converting them to SAS date values, where

yyyy
is the two-byte representation of the four-digit Gregorian year.

ddd
is the one-and-a-half byte representation of the three-digit integer that corresponds to the Julian day of the year, 1–365 (or 1–366 for leap years).

F
is the half byte that contains all binary 1s, which assigns the value as positive.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

Examples

```
input date pdjulg4.;
```

Data Line in Hexadecimal	Results*
`----+----1`	
`1999003F`	**`14247`**

* SAS date value 14247 represents January 3, 1999.

See Also

Formats:

“JULDAY*w*.” on page 128
“JULIAN*w*.” on page 129
“PDJULG*w*.” on page 145
“PDJULI*w*.” on page 146

Functions:

“DATEJUL” on page 313
“JULDATE” on page 416

Informats:

“JULIAN*w*.” on page 685
“PDJULI*w*.” on page 697

System Option:

“YEARCUTOFF=” on page 1177

PDJULI*w*.

Reads packed Julian dates in the hexadecimal format *ccyyddd*F for IBM

Category: Date and Time

Syntax

PDJULI*w*.

Syntax Description

w
specifies the width of the input field.

Default: 4

Range: 4

Details

The PDJULI*w*. informat reads IBM packed Julian date values in the form *ccyyddd*F, converting them to SAS date values, where

cc
is the one-byte representation of a two-digit integer that represents the century.

yy
is the one-byte representation of a two-digit integer that represents the year. The PDJULI*w* informat makes an adjustment to the one-byte century representation by adding 1900 to the two-byte *ccyy* value in order to produce the correct four–digit Gregorian year. This adjustment causes *ccyy* values of 0098 to become 1998, 0101 to become 2001, and 0218 to become 2118.

ddd
is the one-and-a-half bytes representation of the three-digit integer that corresponds to the Julian day of the year, 1–365 (or 1–366 for leap years).

F
is the half byte that contains all binary 1s, which assigns the value as positive.

Examples

```
input date pdjuli4.;
```

Data Lines in Hexadecimal	Results*
`----+----1`	
0099001F	**14245**
0110015F	**18277**

* SAS date value 14245 is January 1, 1999. SAS date value 18277 is January 15, 2010.

See Also

Formats:

"JULDAY*w*." on page 128
"JULIAN*w*." on page 129
"PDJULG*w*." on page 145
"PDJULI*w*." on page 146

Functions:

"DATEJUL" on page 313
"JULDATE" on page 416

Informats:

"JULIAN*w*." on page 685
"PDJULG*w*." on page 696

System Option:

"YEARCUTOFF=" on page 1177

PDTIME*w*.

Reads packed decimal time of SMF and RMF records

Category: Date and Time

Syntax

PDTIME*w*.

Syntax Description

w
specifies the width of the input field.

Requirement: *w* must be 4 because packed decimal time values in RMF and SMF records contain four bytes of information.

Details

The PDTIME*w*. informat reads packed decimal time values that are contained in SMF and RMF records that are produced by IBM mainframe systems and converts the values to SAS time values.

The general form of a packed decimal time value in hexadecimal notation is 0*hhmmss*F, where

0
is a half byte that contains all 0s.

hh
is one byte that represents two digits that correspond to hours.

mm
is one byte that represents two digits that correspond to minutes.

ss
is one byte that represents two digits that correspond to seconds.

F
is a half byte that contains all 1s.

If a field contains all 0s, PDTIME*w*. treats it as a missing value.

PDTIME*w*. enables you to read packed decimal time values from files that are created on an IBM mainframe on any operating environment.

Examples

```
input begin pdtime4.;
```

Data Lines*	Results
0142225F	**51745**

* The data line is a hexadecimal representation of a binary time value that is stored in packed decimal form. Each byte occupies one column of the input field. The result is a SAS time value this corresponds to 2:22.25 PM.

PERCENT*w.d*

Reads percentages as numeric values

Category: Numeric

Syntax

PERCENT*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 6

Range: 1–32

d
optionally specifies the power of 10 by which to divide the value. If the data contain decimal points, the *d* value is ignored.

Range: 0–2

Details

The PERCENT*w.d* informat converts the numeric portion of the input data to a number using the same method as the COMMA*w.d* informat. If a percent sign (%) follows the number in the input field, PERCENT*w.d* divides the number by 100.

Examples

```
input @1 x percent3. @4 y percent5.;
```

Data Lines	Results
`----+----1----+`	
`1% (20%)`	`0.01 -0.2`

PIB*w.d*

Reads positive integer binary (fixed-point) values

Category: Numeric

Syntax

PIB*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 1

Range: 1–8

d
optionally specifies the power of 10 by which to divide the value.

Range: 0–10

Details

All values are treated as positive. PIB*w.d* reads positive integer binary values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

Note: Different operating environments store positive integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering on Big Endian and Little Endian Platforms” on page 634. △

Comparisons

- Positive integer binary values are the same as integer binary values except that the sign bit is part of the value, which is always a positive integer. The PIB*w.d* informat treats all values as positive and includes the sign bit as part of the value.
- The PIB*w.d* informat with a width of 1 results in a value that corresponds to the binary equivalent of the contents of a byte. This is useful if your data contain values between hexadecimal 80 and hexadecimal FF, where the high-order bit can be misinterpreted as a negative sign.
- The IB*w.d* and PIB*w.d* informats are used to read native format integers. (Native format allows you to read and write values that are created in the same operating environment.) The IBR*w.d* and PIBR*w.d* informats are used to read little endian integers in any operating environment.

  To view a table that shows the type of informat to use with big endian and little endian integers, see Table 5.1 on page 635.

  To view a table that compares integer binary notation in several programming languages, see Table 5.2 on page 636.

Examples

You can use the INPUT statement and specify the PIB informat. However, in these examples we use the informat with the INPUT function, where binary input values are described by using a hex literal.

```
x=input('0100'x,pib2.);
y=input('0001'x,pib2.);
```

SAS Statements	Results on Big Endian Platforms	Results on Little Endian Platforms
put x=;	**256**	**1**
put y=;	**1**	**256**

See Also

Informat:

"PIBR*w.d*" on page 702

PIBR*w.d*

Reads positive integer binary (fixed-point) values in Intel and DEC formats

Category: Numeric

Syntax

PIBR*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 1

Range: 1–8

d
optionally specifies the power of 10 by which to divide the value.

Range: 0–10

Details

All values are treated as positive. PIBR*w.d* reads positive integer binary values that have been generated by and for Intel and DEC operating environments. Use PIBR*w.d* to read positive integer binary data from Intel or DEC environments on other operating environments. The PIBR*w.d* informat in SAS code allows for a portable implementation for reading the data in any operating environment.

Note: Different operating environments store positive integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see "Byte Ordering on Big Endian and Little Endian Platforms" on page 634. △

Comparisons

- Positive integer binary values are the same as integer binary values except that the sign bit is part of the value, which is always a positive integer. The PIBR*w.d* informat treats all values as positive and includes the sign bit as part of the value.
- The PIBR*w.d* informat with a width of 1 results in a value that corresponds to the binary equivalent of the contents of a byte. This is useful if your data contain values between hexadecimal 80 and hexadecimal FF, where the high-order bit can be misinterpreted as a negative sign.
- On Intel and DEC platforms, the PIB*w.d* and PIBR*w.d* informats are equivalent.
- The IB*w.d* and PIB*w.d* informats are used to read native format integers. (Native format allows you to read and write values that are created in the same operating environment.) The IBR*w.d* and PIBR*w.d* informats are used to read little endian integers in any operating environment.

  To view a table that shows the type of informat to use with big endian and little endian integers, see Table 5.1 on page 635.

  To view a table that compares integer binary notation in several programming languages, see Table 5.2 on page 636.

Examples

You can use the INPUT statement and specify the PIBR informat. However, these examples use the informat with the INPUT function, where binary input values are described using a hex literal.

```
x=input('0100'x,pibr2.);
y=input('0001'x,pibr2.);
```

SAS Statements	Results on Big Endian Platforms	Results on Little Endian Platforms
`put x=;`	`1`	`1`
`put y=;`	`256`	`256`

See Also

Informat:

"PIB*w.d*" on page 701

PK*w.d*

Reads unsigned packed decimal data

Category: Numeric

Syntax

PK*w.d*

Syntax Description

w
specifies the number of bytes of unsigned packed decimal data, each of which contains two digits.

Default: 1

Range: 1–16

d
optionally specifies the power of 10 by which to divide the value.

Range: 0–10

Details

Each byte of unsigned packed decimal data contains two digits.

Comparisons

The PK*w.d* informat is the same as the PD*w.d* informat, except that PK*w.d* treats the sign half of the field's last byte as part of the value, not as the sign of the value.

Examples

```
input @1 x pk3.;
```

Data Lines*	Results
`----+----1`	
`001234`	`1234`

* The data line is a hexadecimal representation of a binary number stored in unsigned packed decimal form. Each byte occupies one column of the input field.

PUNCH.*d*

Reads whether a row of column-binary data is punched

Category: Column Binary

Syntax

PUNCH.*d*

Syntax Description

d
specifies which row in a card column to read.

Range: 1–12

Details

This informat assigns the value 1 to the variable if row *d* of the current card column is punched, or 0 if row *d* of the current card column is not punched. After PUNCH.*d* reads a field, the pointer does not advance to the next column.

Examples

Data Lines*	SAS Statements	Results
`12-7-8`	`input x punch.12`	`1`
	`input x punch.11`	`0`
	`input x punch0.7`	`1`

* The data line is punched card code. The punch card column for the example data has row 12, row 7, and row 8 punched.

See Also

Informats:

"$CB*w*." on page 646
"CB*w.d*" on page 663
"ROW*w.d*" on page 710

RB*w.d*

Reads numeric data that are stored in real binary (floating-point) notation

Category: Numeric

Syntax

RB*w.d*

Syntax Description

w
specifies the width of the input field.
Default: 4
Range: 2–8

d
optionally specifies the power of 10 by which to divide the value.
Range: 0–10

Details

Note: Different operating environments store real binary values in different ways. However, the RB*w.d* informat reads real binary values with consistent results if the values are created on the same type of operating environment that you use to run SAS. △

Comparisons

The following table compares the names of real binary notation in several programming languages:

	Real Binary Notation	
Language	4 Bytes	8 Bytes
SAS	RB4.	RB8.
FORTRAN	REAL*4	REAL*8
C	float	double

	Real Binary Notation	
Language	4 Bytes	8 Bytes
IBM 370 assembler	F	D
PL/I	FLOAT BIN(21)	FLOAT BIN(53)

CAUTION:
Using the RBw.d informat to read real binary information on equipment that conforms to the IEEE standard for floating-point numbers results in a truncated eight-byte number (double-precision), rather than in a true four-byte floating-point number (single-precision).
△

Examples

```
input @1 x rb8.;
```

Data Lines*	Results
`----+----1`	
4280000000000000	**128**

* The data line is a hexadecimal representation of a real binary (floating-point) number on an IBM mainframe operating environment. Each byte occupies one column of the input field.

See Also

Informat:

"IEEE*w.d*" on page 682

RMFDUR*w.*

Reads duration intervals of RMF records

Category: Date and Time

Syntax

RMFDUR*w*.

Syntax Description

w
specifies the width of the input field.

Requirement: *w* must be 4 because packed decimal duration values in RMF records contain four bytes of information.

Details

The RMFDUR*w*. informat reads the duration of RMF measurement intervals of RMF records that are produced as packed decimal data by IBM mainframe systems and converts them to SAS time values.

The general form of the duration interval data in an RMF record in hexadecimal notation is *mmssttt*F, where

mm
is the one-byte representation of two digits that correspond to minutes.

ss
is the one-byte representation of two digits that correspond to seconds.

ttt
is the one-and-a-half-bytes representation of three digits that correspond to thousandths of a second.

F
is a half byte that contains all binary 1s, which assigns the value as positive.

If the field does not contain packed decimal data, RMFDUR*w*. results in a missing value.

Comparisons

- Both the RMFDUR*w*. informat and the RMFSTAMP*w*. informat read packed decimal information from RMF records that are produced by IBM mainframe systems.
- The RMFDUR*w*. informat reads duration data and results in a time value.
- The RMFSTAMP*w*. informat reads time-of-day data and results in a datetime value.

Examples

```
input dura rmfdur4.;
```

Data Lines *	Results
`----+----1----+`	
3552226F	2152.266

* The data line is a hexadecimal representation of a binary duration value that is stored in packed decimal form as it would appear in an RMF record. Each byte occupies one column of the input field. The result is a SAS time value corresponding to 00:35:52.226.

See Also

Informats:

"RMFSTAMP*w*." on page 709
"SMFSTAMP*w*. "on page 713

RMFSTAMP*w*.

Reads time and date fields of RMF records

Category: Date and Time

Syntax

RMFSTAMP*w*.

Syntax Description

w
specifies the width of the input field.

Requirement: *w* must be 8 because packed decimal time and date values in RMF records contain eight bytes of information: four bytes of time data that are followed by four bytes of date data.

Details

The RMFSTAMP*w*. informat reads packed decimal time and date values of RMF records that are produced by IBM mainframe systems, and converts the time and date values to SAS datetime values.

The general form of the time and date information in an RMF record in hexadecimal notation is 0*hhmmss*F*ccyyddd*F, where

0
is the half byte that contains all binary 0s.

hh
is the one-byte representation of two digits that correspond to the hour of the day.

mm
is the one-byte representation of two digits that correspond to minutes.

ss
is 1 byte that represents two digits that correspond to seconds.

cc
is the one-byte representation of two digits that correspond to the century.

yy
is the one-byte representation of two digits that correspond to the year.

ddd
is the one-and-a-half bytes that contain three digits that correspond to the day of the year.

F
is the half byte that contains all binary 1s.
The century indicators 00 correspond to 1900, 01 to 2000, and 02 to 2100.

RMFSTAMP*w*. enables you to read, on any operating environment, packed decimal time and date values from files that are created on an IBM mainframe.

Comparisons

Both the RMFSTAMP*w*. informat and the PDTIME*w*. informat read packed decimal values from RMF records. The RMFSTAMP*w*. informat reads both time and date values and results in a SAS datetime value. The PDTIME*w*. informat reads only time values and results in a SAS time value.

Examples

```
input begin rmfstamp8.;
```

Data Lines*	Results
`----+----1----+----2`	
`0142225F0102286F`	`1350138145`

* The data line is a hexadecimal representation of a binary time and date value that is stored in packed decimal form as it would appear in an RMF record. Each byte occupies one column of the input field. The result is a SAS datetime value that corresponds to October 13, 2002, 2:22.25 PM.

ROW*w.d*

Reads a column-binary field down a card column

Category: Column Binary

Syntax

ROW*w.d*

Syntax Description

w
specifies the row where the field begins.
Range: 0–12

d

specifies the length in rows of the field.

Default: 1

Range: 1–25

Details

The ROW*w*.*d* informat assigns the relative position of the punch in the field to a numeric variable.

If the field that you specify has more than one punch, ROW*w*.*d* assigns the variable a missing value and sets the automatic variable _ERROR_ to 1. If the field has no punches, ROW*w*.*d* assigns the variable a missing value.

ROW*w*.*d* can read fields across columns, continuing with row 12 of the new column and going down through the rest of the rows. After ROW*w*.*d* reads a field, the pointer moves to the next row.

Examples

```
input x row5.3
input x row7.1
input x row5.2
input x row3.5
```

Data Lines*	Results
`----+----1`	
`00`	
`04`	`3`
	`1`
	`.`
	`5`

* The data line is a hexadecimal representation of the column binary. The punch card column for the example data has row 7 punched. The binary representation is 0000 0000 0000 0100.

See Also

Informats:

"$CB*w*." on page 646

"CB*w*.*d*" on page 663

"PUNCH.*d*" on page 705

SHRSTAMP*w*.

Reads date and time values of SHR records

Category: Date and Time

Syntax

SHRSTAMP*w*.

Syntax Description

w
specifies the width of the input field.

Requirement: *w* must be 8 because packed decimal date and time values in SHR records contain eight bytes of information: four bytes of date data that are followed by four bytes of time data.

Details

The SHRSTAMP*w*. informat reads packed decimal date and time values of SHR records that are produced by IBM mainframe environments and converts the date and time values to SAS datetime values.

The general form of the date and time information in an SHR record in hexadecimal notation is *yyyydddF hhmmssth*, where

yyyy
is the two-bytes representation of four digits that correspond to the year.

ddd
is the one-and-a-half bytes that contain three digits that correspond to the day of the year.

F
is the half byte that contains all binary 1s.

hh
is the one-byte representation of two digits that correspond to the hour of the day.

mm
is the one-byte representation of two digits that correspond to minutes.

ss
is the one-byte representation of two digits that correspond to seconds.

th
is the one-byte representation of two digits that correspond to a hundredth of a second.

The SHRSTAMP*w*. informat enables you to read, on any operation environment, packed decimal date and time values from files that are created on an IBM mainframe.

Examples

```
input begin shrstamp8.;
```

Data Lines*	Results
`----+----1----+----2`	
`0097239F12403576`	`1188304835.8`

* The data line is a hexadecimal representation of a packed decimal date and time value that is stored as it would appear in an SHR record. Each byte occupies one column of the input field. The result is a SAS datetime value that corresponds to Aug. 27, 1997 12:40:36 PM.

SMFSTAMP*w*.

Reads time and date values of SMF records

Category: Date and Time

Syntax

SMFSTAMP*w*.

Syntax Description

w

specifies the width of the input field.

Requirement: *w* must be 8 because time and date values in SMF records contain eight bytes of information: four bytes of time data that are followed by four bytes of date data.

Tip: The time portion of an SMF record is a four-byte integer binary number that represents time as the number of hundredths of a second past midnight.

Details

The SMFSTAMP*w*. informat reads integer binary time values and packed decimal date values of SMF records that are produced by IBM mainframe systems and converts the time and date values to SAS datetime values.

The date portion of an SMF record in hexadecimal notation is *ccyyddd*F, where

cc
is the one-byte representation of two digits that correspond to the century.

yy
is the one-byte representation of two digits that correspond to the year.

ddd
is the one-and-a-half bytes that contain three digits that correspond to the day of the year.

F
is the half byte that contains all binary 1s.

The SMFSTAMP*w*. informat enables you to read, on any operating environment, integer binary time values and packed decimal date values from files that are created on an IBM mainframe.

Examples

```
input begin smfstamp8.;
```

Data Lines*	Results
`----+----1----+----2`	
`0058DC0C0098200F`	`1216483835`

* The data line is a hexadecimal representation of a binary time and date value that is stored as it would appear in an SMF record. Each byte occupies one column of the input field. The result is a SAS datetime value that corresponds to July 19, 1998 4:10:35 PM.

S370FF*w.d*

Reads EBCDIC numeric data

Category: Numeric

Syntax

S370FF*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 12

Range: 1–32

d

optionally specifies the power of 10 by which to divide the value.

Range: 0–31

Details

The S370FF*w.d* informat reads numeric data that are represented in EBCDIC and converts the data to native format. If EBCDIC is the native format, S370FF*w.d* performs no conversion.

S370FF*w.d* reads EBCDIC numeric values that are represented with one byte per digit. Use S370FF*w.d* on other operating environments to read numeric data from IBM mainframe files.

S370FF*w.d* reads numeric values located anywhere in the input field. EBCDIC blanks can precede or follow a numeric value with no effect. If a value is negative, an EBCDIC minus sign should immediately precede the value. S370FF*w.d* reads values with EBCDIC decimal points and values in scientific notation, and it interprets a single EBCDIC period as a missing value.

Comparisons

The S370FF*w.d* informat performs the same role for numeric data that the $EBCDIC*w.d* informat does for character data. That is, on an IBM mainframe system, S370FF*w.d* has the same effect as the standard *w.d* informat. On all other systems, using S370FF*w.d* is equivalent to using $EBCDIC*w.d* as well as using the standard *w.d* informat.

Examples

```
input @1 x s370ff3.;
```

Data Lines*	Results
`----+----1`	
`F1F2F3`	`123`
`F2F4F0`	`240`

* The data lines are hexadecimal representations of codes for characters. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one character value.

S370FIB*w.d*

Reads integer binary (fixed-point) values, including negative values, in IBM mainframe format

Category: Numeric

Syntax

S370FIB*w.d*

Syntax Description

w
specifies the width of the input field.
Default: 4
Range: 1–8

d
optionally specifies the power of 10 by which to divide the value.
Range: 0–10

Details

The S370FIB*w.d* informat reads integer binary (fixed-point) values that are stored in IBM mainframe format, including negative values that are represented in two's

complement notation. S370FIB*w.d* reads integer binary values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

Use S370FIB*w.d* for integer binary data that are created in IBM mainframe format for reading in other operating environments.

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see "Byte Ordering on Big Endian and Little Endian Platforms" on page 634.
△

Comparisons

- If you use SAS on an IBM mainframe, S370FIB*w.d* and IB*w.d* are identical.
- S370FPIB*w.d*, S370FIBU*w.d*, and S370FIB*w.d* are used to read big endian integers in any operating environment.

  To view a table that shows the type of informat to use with big endian and little endian integers, see Table 5.1 on page 635.

  To view a table that compares integer binary notation in several programming languages, see Table 5.2 on page 636.

Examples

You can use the INPUT statement and specify the S370FIB informat. However, this example uses the informat with the INPUT function, where the binary input value is described by using a hex literal.

```
x=input('0080'x,s370fib2.);
```

SAS Statement	Results
put x=;	**128**

See Also

Informats:

"S370FIBU*w.d*" on page 716

"S370FPIB*w.d*" on page 720

S370FIBU*w.d*

Reads unsigned integer binary (fixed-point) values in IBM mainframe format

Category: Numeric

Syntax

S370FIBU*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 4

Range: 1–8

d
optionally specifies the power of 10 by which to divide the value. SAS uses the *d* value even if the data contain decimal points.

Range: 0–10

Details

The S370FIBU*w.d* informat reads unsigned integer binary (fixed-point) values that are stored in IBM mainframe format, including negative values that are represented in two's complement notation. Unsigned integer binary values are the same as integer binary values, except that all values are treated as positive. S370FIBU*w.d* reads integer binary values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

Use S370FIBU*w.d* for unsigned integer binary data that are created in IBM mainframe format for reading in other operating environments.

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see "Byte Ordering on Big Endian and Little Endian Platforms" on page 634. △

Comparisons

- The S370FIBU*w.d* informat is equivalent to the COBOL notation PIC 9(*n*) BINARY, where *n* is the number of digits.
- The S370FIBU*w.d* and S370FPIB*w.d* informats are identical.
- S370FPIB*w.d*, S370FIBU*w.d*, and S370FIB*w.d* are used to read big endian integers in any operating environment.

  To view a table that shows the type of informat to use with big endian and little endian integers, see Table 5.1 on page 635.

  To view a table that compares integer binary notation in several programming languages, see Table 5.2 on page 636.

Examples

You can use the INPUT statement and specify the S370FIBU informat. However, these examples use the informat with the INPUT function, where binary input values are described by using a hex literal.

```
x=input('7F'x,s370fibu1.);
y=input('F6'x,s370fibu1.);
```

SAS Statements	Results
put x=;	**127**
put y=;	**246**

See Also

Informats:

"S370FIB*w.d*" on page 715
"S370FPIB*w.d*" on page 720

S370FPD*w.d*

Reads packed data in IBM mainframe format

Category: Numeric

Syntax

S370FPD*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 1

Range: 1–16

d
optionally specifies the power of 10 by which to divide the value.

Range: 0–10

Details

Packed decimal data contain two digits per byte, but only one digit in the input field represents the sign. The last half of the last byte indicates the sign: a C or an F for positive numbers and a D for negative numbers.

Use S370FPD*w.d* to read packed decimal data from IBM mainframe files on other operating environments.

Comparisons

- If you use SAS on an IBM mainframe, the S370FPD*w.d* and the PD*w.d* informats are identical.
- The following table compares the equivalent packed decimal notation by programming language:

Language	Packed Decimal Notation
SAS	S370FIB4.
PL/I	FIXED DEC(7,0)
COBOL	COMP-3 PIC 9(7)
assembler	PL4

S370FPDU*w.d*

Reads unsigned packed decimal data in IBM mainframe format

Category: Numeric

Syntax

S370FPDU*w.d*

Syntax Description

w
specifies the width of the input field.
Default: 1
Range: 1–16

d
optionally specifies the power of 10 by which to divide the value
Range: 0–10

Details

Packed decimal data contain two digits per byte. The last half of the last byte, which indicates the sign for signed packed data, is always F for unsigned packed data.

Use S370FPDU*w.d* on other operating environments to read unsigned packed decimal data from IBM mainframe files.

Comparisons

- The S370FPDU*w.d* informat is similar to the S370FPD*w.d* informat except that the S370FPDU*w.d* informat rejects all sign digits except F.
- The S370FPDU*w.d* informat is equivalent to the COBOL notation PIC 9(*n*) PACKED-DECIMAL, where the *n* value is the number of digits.

Examples

```
input @1 x s370fpdu3.;
```

Data Lines*	Results
`----+----1`	
`12345F`	`12345`

* The data line is a hexadecimal representation of a binary number that is stored in packed decimal form. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one column of the input field.

S370FPIB*w.d*

Reads positive integer binary (fixed-point) values in IBM mainframe format

Category: Numeric

Syntax

S370FPIB*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 4

Range: 1–8

d
optionally specifies the power of 10 by which to divide the value.

Range: 0–10

Details

Positive integer binary values are the same as integer binary values, except that all values are treated as positive. S370FPIB*w.d* reads integer binary values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

Use S370FPIB*w.d* for positive integer binary data that are created in IBM mainframe format for reading in other operating environments.

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see "Byte Ordering on Big Endian and Little Endian Platforms" on page 634.
△

Comparisons

- If you use SAS on an IBM mainframe, S370FPIB*w.d* and PIB*w.d* are identical.
- S370FPIB*w.d*, S370FIBU*w.d*, and S370FIB*w.d* are used to read big endian integers in any operating environment.

  To view a table that shows the type of informat to use with big endian and little endian integers, see Table 5.1 on page 635.

  To view a table that compares integer binary notation in several programming languages, see Table 5.2 on page 636.

Examples

You can use the INPUT statement and specify the S370FPIB informat. However, this example uses the informat with the INPUT function, where the binary input value is described using a hex literal.

```
x=input('0100'x,s370fpib2.);
```

SAS Statement	Results
put x=;	**256**

See Also

Informats:

"S370FIB*w.d*" on page 715

"S370FIBU*w.d*" on page 716

S370FRB*w.d*

Reads real binary (floating-point) data in IBM mainframe format

Category: Numeric

Syntax

S370FRB*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 6

Range: 2–8

d

optionally specifies the power of 10 by which to divide the value.

Range: 0–10

Details

Real binary values are represented in two parts: a mantissa that gives the value, and an exponent that gives the value's magnitude.

Use S370FRB*w.d* to read real binary data from IBM mainframe files on other operating environments.

Comparisons

- If you use SAS on an IBM mainframe, S370FRB*w.d* and RB*w.d* are identical.
- The following table shows the equivalent real binary notation for several programming languages:

	Real Binary Notation	
Language	4 Bytes	8 Bytes
SAS	S370FRB4.	S370FRB8.
PL/I	FLOAT BIN(21)	FLOAT BIN(53)
FORTRAN	REAL*4	REAL*8
COBOL	COMP-1	COMP-2
assembler	E	D
C	float	double

See Also

Informat:

"RB*w.d*" on page 706

S370FZD*w.d*

Reads zoned decimal data in IBM mainframe format

Category: Numeric

Syntax

S370FZD*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 8

Range: 1–32

d

optionally specifies the power of 10 by which to divide the value. If the data contain decimal points, the *d* value is ignored.

Range: 0–10

Details

Zoned decimal data are similar to standard decimal data in that every digit requires one byte. However, the value's sign is stored in the last byte, along with the last digit.

Use S370FZD*w.d* on other operating environments to read zoned decimal data from IBM mainframe files.

Comparisons

- If you use SAS on an IBM mainframe, S370FZD*w.d* and ZD*w.d* are identical.
- The following table shows the equivalent zoned decimal notation for several programming languages:

Language	Zoned Decimal Notation
SAS	S370FZD3.
PL/I	PICTURE'99T'
COBOL	PIC S9(3) DISPLAY
assembler	ZL3

Examples

```
input @1 x s370fzd3.;
```

Data Lines*	Results
`----+----1`	
F1F2C3	**123**
F1F2D3	**-123**

* The data line contains a hexadecimal representation of a binary number stored in zoned decimal format on an IBM mainframe operating environment. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one column of the input field.

See Also

Informat:

"ZD*w.d* "on page 738

S370FZDL*w.d*

Reads zoned decimal leading-sign data in IBM mainframe format

Category: Numeric

Syntax

S370FZDL*w.d*

Syntax Description

w
specifies the width of the input field.
Default: 8
Range: 1–32

d
optionally specifies the power of 10 by which to divide the value.
Range: 0–10

Details

Use S370FZDL*w.d* on other operating environments to read zoned decimal data from IBM mainframe files.

Comparisons

- Zoned decimal leading-sign data is similar to standard zoned decimal data except that the sign of the value is stored in the first byte of zoned decimal leading-sign data, along with the first digit.
- The S370FZDL*w.d* informat is equivalent to the COBOL notation PIC S9(*n*) DISPLAY SIGN LEADING, where the *n* value is the number of digits.

Examples

```
input @1 x s370fzdl3.;
```

Data Lines*	Results
`----+----1`	
`C1F2F3`	`123`
`D1F2F3`	`-123`

* The data lines contain a hexadecimal representation of a binary number stored in zoned decimal format on an IBM mainframe operating environment. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one column of the input field.

S370FZDS*w.d*

Reads zoned decimal separate leading-sign data in IBM mainframe format

Category: Numeric

Syntax

S370FZDS*w.d*

Syntax Description

w
specifies the width of the input field.
Default: 8
Range: 2–32

d
optionally specifies the power of 10 by which to divide the value.
Range: 0–10

Details

Use S370FZDS*w.d* on other operating environments to read zoned decimal data from IBM mainframe files.

Comparisons

- Zoned decimal separate leading-sign data is similar to standard zoned decimal data except that the sign of the value is stored in the first byte of zoned decimal leading sign data, and the first digit of the value is stored in the second byte.
- The S370FZDS*w.d* informat is equivalent to the COBOL notation PIC S9(*n*) DISPLAY SIGN LEADING SEPARATE, where the *n* value is the number of digits.

Examples

```
input @1 x s370fzds4.;
```

Data Lines*	Results
`----+----1`	
`4EF1F2F3`	`123`
`60F1F2F3`	`-123`

* The data line contains a hexadecimal representation of a binary number that is stored in zoned decimal format on an IBM mainframe operating environment. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one column of the input field.

S370FZDT*w.d*

Reads zoned decimal separate trailing-sign data in IBM mainframe format

Category: Numeric

Syntax

S370FZDT*w.d*

Syntax Description

w
specifies the width of the input field.
Default: 8
Range: 2–32

d
optionally specifies the power of 10 by which to divide the value.
Range: 0–10

Details

Use S370FZDT*w.d* on other operating environments to read zoned decimal data from IBM mainframe files.

Comparisons

- Zoned decimal separate trailing-sign data are similar to zoned decimal separate leading-sign data except that the sign of the value is stored in the last byte of zoned decimal separate trailing-sign data.
- The S370FZDT*w.d* informat is equivalent to the COBOL notation PIC S9(*n*) DISPLAY SIGN TRAILING SEPARATE, where the *n* value is the number of digits.

Examples

```
input @1 x s370fzdt4.;
```

Data Lines*	Results
`----+----1`	
`F1F2F34E`	`123`
`F1F2F360`	`-123`

* The data line contains a hexadecimal representation of a binary number that is stored in zoned decimal format on an IBM mainframe operating environment. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one column of the input field.

S370FZDU*w.d*

Reads unsigned zoned decimal data in IBM mainframe format

Category: Numeric

Syntax

S370FZDU*w*.*d*

Syntax Description

w
specifies the width of the input field.
Default: 8
Range: 1–32

d
optionally specifies the power of 10 by which to divide the value.
Range: 0–10

Details

Use S370FZDU*w*.*d* on other operating environments to read unsigned zoned decimal data from IBM mainframe files.

Comparisons

- The S370FZDU*w*.*d* informat is similar to the S370FZD*w*.*d* informat except that the S370FZDU*w*.*d* informat rejects all sign digits except F.
- The S370FZDU*w*.*d* informat is equivalent to the COBOL notation PIC 9(*n*) DISPLAY, where the *n* value is the number of digits.

Examples

```
input @1 x s370fzdu3.;
```

Data Lines*	Results
`----+----1`	
`F1F2F3`	`123`

* The data line contains a hexadecimal representation of a binary number that is stored in zoned decimal format on an IBM mainframe operating environment. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one column of the input field.

TIME*w*.

Reads hours, minutes, and seconds in the form *hh:mm:ss.ss*

Category: Date and Time

Syntax

TIME*w.*

Syntax Description

w
specifies the width of the input field.

Default: 8

Range: 5–32

Details

Time values must be in the form *hh:mm:ss.ss*, where

hh
is the number of hours that range from 00 through 23.

mm
is the number of minutes that range from 00 through 59.

ss.ss
is the number of seconds ranging from 00 through 59 with the fraction of a second following the decimal point.

Separate *hh*, *mm*, and *ss.ss* with a special character. If you do not enter a value for seconds, SAS assumes a value of 0.

The stored value is the total number of seconds in the time value.

Examples

```
input begin time10.;
```

Data Lines	Results
`----+----1----+`	
`11:23:07.4`	`40987.4`

The TIME informat can read time values with AM or PM in the value.

```
input begin time8.;
```

Data Lines	Results
`----+----1----+`	
`1:13 PM`	`47580.0`

See Also

Formats:

"HHMM*w*.*d*" on page 121
"HOUR*w*.*d*" on page 123
"MMSS*w*.*d*" on page 134
"TIME*w*.*d*" on page 172

Functions:

"HOUR" on page 395
"MINUTE" on page 445
"SECOND" on page 542
"TIME" on page 563

TODSTAMP*w*.

Reads an eight-byte time-of-day stamp

Category: Date and Time

Syntax

TODSTAMP*w*.

Syntax Description

w
specifies the width of the input field.

Requirement: *w* must be 8 because the OS TIME macro or the STCK System/370 instruction on IBM mainframes each return an eight-byte value.

Details

The TODSTAMP*w*. informat reads time-of-day clock values that are produced by IBM mainframe operating systems and converts the clock values to SAS datetime values.

If the time-of-day value is all 0s, TODSTAMP*w*. results in a missing value.

Use TODSTAMP*w*. on other operating environments to read time-of-day values that are produced by an IBM mainframe.

Examples

```
input btime todstamp8.;
```

Data Lines*	Results
`----+----1----+----2`	
`B361183D5FB80000`	`1262303998`

* The data line is a hexadecimal representation of a binary, 8-byte time-of-day clock value. Each byte occupies one column of the input field. The result is a SAS datetime value that corresponds to December 31, 1999, 11:59:58 PM.

TU*w*.

Reads timer units

Category: Date and Time

Syntax

TU*w*.

Syntax Description

w
specifies the width of the input field.

Requirement: *w* must be 4 because the OS TIME macro returns a four-byte value.

Details

The TU*w*. informat reads timer unit values that are produced by an IBM mainframe operating environment and OS/VS software and converts the timer unit values to SAS time values.

There are exactly 38,400 software timer units per second. The low-order bit in a timer unit value represents approximately 26.041667 microseconds.

Use the TU*w*. informat to read timer unit values that are produced by an IBM mainframe on other operating environments.

Examples

```
input btime tu4.;
```

Data Lines*	Results
`----+----1----+`	
`8FC7A9BC`	`62818.411563`

* The data line is a hexadecimal representation of a binary, four-byte timer unit value. Each byte occupies one column of the input field. The result is a SAS time value that corresponds to 5:26:58.41 p.m.

VAXRB*w.d*

Reads real binary (floating-point) data in VMS format

Category: Numeric

Syntax

VAXRB*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 4

Range: 2–8

d
optionally specifies the power of 10 by which to divide the value.

Range: 0–10

Details

Use the VAXRB*w.d* informat to read floating-point data from VMS files on other operating environments.

Comparisons

If you use SAS that is running under VMS, the VAXRB*w.d* and the RB*w.d* informats are identical.

See Also

Informat:

"RB*w.d*" on page 706

w.d

Reads standard numeric data

Category: Numeric

Syntax

w.d

Syntax Description

w
specifies the width of the input field.
Range: 1–32

d
optionally specifies the power of 10 by which to divide the value. If the data contain decimal points, the *d* value is ignored.
Range: 0–31

Details

The *w.d* informat reads numeric values that are located anywhere in the field. Blanks can precede or follow a numeric value with no effect. A minus sign with no separating blank should immediately precede a negative value. The *w.d* informat reads values with decimal points and values in scientific E-notation, and it interprets a single period as a missing value.

Comparisons

- The *w.d* informat is identical to the BZ*w.d* informat, except that the *w.d* informat ignores trailing blanks in the numeric values. To read trailing blanks as 0s, use the BZ*w.d* informat.
- The *w.d* informat can read values in scientific E-notation exactly as the E*w.d* informat does.

Examples

```
input @1 x 6. @10 y 6.2;
put x @7 y;
```

Data Lines	Results	
----+----1----+----+		
23 2300	23	23
23 2300	23	23
23 -2300	23	-23
23.0 23.	23	23
2.3E1 2.3	23	2.3
-23 0	-23	0

YEN*w.d*

Removes embedded yen signs, commas, and decimal points

Category: Numeric

Syntax

YEN*w.d*

Syntax Description

w
specifies the width of the input field.
Default: 1
Range: 1–32

d
optionally specifies the power of 10 by which to divide the value.
Requirement: *d* must be 0 or 2
Tip: If the *d* is 2, then YEN*w.d* reads a decimal point and two decimal digits. If *d* is 0, YEN*w.d* reads the value without a decimal point.

Details

The hexadecimal representation of the code for the yen sign character is 5B on EBCDIC systems and 5C on ASCII systems. The monetary character that these codes represent may be different in other countries.

Examples

```
input value yen10.2;
```

Data Lines	Results
`----+----1----+`	
¥1254.71	**1254.71**

YYMMDD*w*.

Reads date values in the form *yymmdd* or *yyyymmdd*

Category: Date and Time

Syntax

YYMMDD*w.*

Syntax Description

w
specifies the width of the input field.

Default: 6

Range: 6–32

Details

The date values must be in the form *yymmdd* or *yyyymmdd*, where

yy or*yyyy*
is a two- or four-digit integer that represents the year.

mmm
is the first three letters of the month name.

dd
is an integer from 01 through 31 that represents the day of the month.

You can separate the year, month, and day values by blanks or by special characters. However, if delimiters are used, place them between all the values. You can also place blanks before and after the date. Make sure the width of the input field allows space for blanks and special characters.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

Examples

```
input calendar_date yymmdd10.;
```

Data Lines	Results
`----+----1----+`	
020316	**15415**
02/03/16	**15415**

Data Lines	Results
`02 03 16`	`15415`
`2002-03-16`	`15415`

See Also

Formats:
- "DATE*w*." on page 89
- "DDMMYY*w*." on page 94
- "MMDDYY*w*." on page 131
- "YYMMDD*w*." on page 188

Functions:
- "DAY" on page 315
- "MDY" on page 443
- "MONTH" on page 451
- "YEAR" on page 618

Informats:
- "DATE*w*." on page 666
- "DDMMYY*w*." on page 669
- "MMDDYY*w*." on page 687

System Option:
- "YEARCUTOFF=" on page 1177

YYMMN*w*.

Reads date values in the form *yyyymm* or *yymm*

Category: Date and Time

Syntax

YYMMN*w*.

Syntax Description

w
specifies the width of the input field.
Default: 4
Range: 4–6

Details

The date values must be in the form *yyyymm* or *yymm*, where

yy or *yyyy*
is a two- or four-digit integer that represents the year.

mm
is a two-digit integer that represents the month.

The *N* in the informat name must be used and indicates that you cannot separate the year and month values by blanks or by special characters. SAS automatically adds a day value of 01 to the value to make a valid SAS date variable.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

Examples

```
input date1 yymmn6.;
```

Data Lines	Results
----+----1----+	
200208	01AUG2002

See Also

Formats:
- "DATE*w*." on page 89
- "DDMMYY*w*." on page 94
- "YYMMDD*w*." on page 188
- "YYMM*xw*." on page 187
- "YYMON*w*." on page 192

Functions:
- "DAY" on page 315
- "MONTH" on page 451
- "MDY" on page 443
- "YEAR" on page 618

Informats:
- "DATE*w*." on page 666
- "DDMMYY*w*." on page 669
- "MMDDYY*w*." on page 687
- "YYMMDD*w*." on page 733

System Option:
- "YEARCUTOFF=" on page 1177

YYQ*w*.

Reads quarters of the year

Category: Date and Time

Syntax

YYQ*w*.

Syntax Description

w
specifies the width of the input field.

Default: 4

Range: 4–32

Details

The quarter must be in the form *yy*Q*q* or *yyyy*Q*q*, where

yy or *yyyy*
is an integer that represents the two- or four-digit year.

q
is an integer (1, 2, 3, or 4) that represents the quarter of the year. You can also represent the quarter as 01, 02, 03, or 04.

The letter Q must separate the year value and the quarter value. The year value, the letter Q, and the quarter value cannot be separated by blanks. A value that is read with YYQ*w*. produces a SAS date value that corresponds to the first day of the specified quarter.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option. △

Examples

```
input quarter yyq9.;
```

Data Lines	Results
`----+----1----+`	
02Q2	**15431**

Data Lines	Results
02Q02	**15431**
2002Q02	**15431**

See Also

Functions:

"QTR" on page 509

"YEAR" on page 618

"YYQ" on page 621

System Option:

"YEARCUTOFF=" on page 1177

ZD*w.d*

Reads zoned decimal data

Category: Numeric

Syntax

ZD*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 1

Range: 1–32

d

optionally specifies the power of 10 by which to divide the value.

Range: 1–31

Details

The ZD*w.d* informat reads zoned decimal data in which every digit requires one byte and in which the last byte contains the value's sign along with the last digit.

Note: Different operating environments store zoned decimal values in different ways. However, ZD*w.d* reads zoned decimal values with consistent results if the values are created in the same type of operating environment that you use to run SAS. △

You can enter positive values in zoned decimal format from a terminal. Some keying devices enable you to enter negative values by overstriking the last digit with a minus sign.

Comparisons

- Like the *w.d* informat, the ZD*w.d* informat reads data in which every digit requires one byte. Use ZDV*w.d* or ZD*w.d* to read zoned decimal data in which the last byte contains the last digit and the sign.
- The ZD*w.d* informat functions like the ZDV*w.d* informat with one exception: ZDV*w.d* validates the input string and disallows invalid data.
- The following table compares the zoned decimal informat with notation in several programming languages:

Language	Zoned Decimal Notation
SAS	ZD3.
PL/I	PICTURE'99T'
COBOL	DISPLAY PIC S 999
IBM 370 assembler	ZL3

Examples

```
input @1 x zd4.;
```

Data Lines*	Results
`----+----1`	
F0F1F2C8	**128**

* The data line contains a hexadecimal representation of a binary number that is stored in zoned decimal format on an IBM mainframe computer system. Each byte occupies one column of the input field.

See Also

Informats:

"*w.d*" on page 731
"ZDV*w.d*" on page 740

ZDB*w.d*

Reads zoned decimal data in which zeros have been left blank

Category: Numeric

Syntax

ZDB*w.d*

Syntax Description

w
specifies the width of the input field.
Default: 1
Range: 1–32

d
optionally specifies the power of 10 by which to divide the value.
Range: 0–31

Details

The ZDB*w.d* informat reads zoned decimal data that are produced in IBM 1410, 1401, and 1620 form, where 0s are left blank rather than being punched.

Examples

```
input @1 x zdb3.;
```

Data Lines*	Results
`----+----1`	
`F140C2`	`102`

* The data line contains a hexadecimal representation of a binary number that is stored in zoned decimal form, including the codes for spaces, on an IBM mainframe operating environment. Each byte occupies one column of the input field.

ZDV*w.d*

Reads and validates zoned decimal data

Category: Numeric

Syntax

ZDV*w.d*

Syntax Description

w
specifies the width of the input field.
Default: 1
Range: 1–32

d
optionally specifies the power of 10 by which to divide the value.
Range: 1–31

Details

The ZDV*w.d* informat reads data in which every digit requires one byte and in which the last byte contains the value's sign along with the last digit. It also validates the input string and disallows invalid data.

ZDV*w.d* is dependent on the operating environment. For example, on IBM mainframes, ZDV*w.d* requires an F for all high-order nibbles except the last. (In contrast, the ZD*w.d* informat ignores the high-order nibbles for all bytes except those that are associated with the sign.) The last high-order nibble accepts values ranging from A-F, where A, C, E, and F are positive values and B and D are negative values. The low-order nibble on IBM mainframes must be a numeric digit that ranges from 0-9, as with ZD.

Note: Different operating environments store zoned decimal values in different ways. However, the ZDV*w.d* informat reads zoned decimal values with consistent results if the values are created in the same type of operating environment that you use to run SAS. △

Comparisons

The ZDV*w.d* informat functions like the ZD*w.d* informat with one exception: ZDV*w.d* validates the input string and disallows invalid data.

Examples

```
input @1 test zdv4.;
```

Data Lines*	Results
`----+----1`	
F0F1F2C8	**128**

* The data line contains a hexadecimal representation of a binary number stored in zoned decimal form. The example was run on an IBM mainframe. The results may vary depending on your operating environment.

See Also

Informats:

"*w.d*" on page 731

"ZD*w.d* "on page 738

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Language Reference, Version 8,* Cary, NC: SAS Institute Inc., 1999.

SAS® Language Reference, Version 8

ISBN 1–58025–369–5

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

1st printing, October 1999