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.

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 in 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. For more information, see the discussion of automatically setting variable values to missing in SAS Language Reference: Concepts.

Using Informat

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 841.

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 481 for an example of a numeric-to-character conversion. For a complete discussion of the INPUT function, see “INPUT” on page 393.

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 DATEw. informat with the variables Birthdate and Interview:

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

An informat that is associated with an INFORMAT statement behaves like an informat that you specify with a colon (:) format modifier in an INPUT statement. (For details about using the colon (:) modifier, see the INPUT, List statement.) 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 w value in an informat specifies the length of character variables
- the w value is ignored for numeric informats
- the d value in an informat behaves in the usual way for numeric informats
- the blanks that are embedded in input data are treated as delimiters unless you change the DELIMITER= option in an INFILE statement.

See “INPUT, List” on page 862 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 DATEw. informat with the variables Birthdate and Interview:

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

See “ATTRIB” on page 732 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:

<table>
<thead>
<tr>
<th>System Options</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMTERR</td>
<td>SAS produces an error that causes the current DATA or PROC step to stop.</td>
</tr>
<tr>
<td>NOFMTERR</td>
<td>SAS continues processing by substituting a default informat.</td>
</tr>
</tbody>
</table>

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), 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>
<thead>
<tr>
<th>Data created for...</th>
<th>Data read on...</th>
<th>Sign?</th>
<th>Informat</th>
</tr>
</thead>
<tbody>
<tr>
<td>big endian</td>
<td>big endian</td>
<td>yes</td>
<td>IB or S370FIB</td>
</tr>
<tr>
<td>big endian</td>
<td>big endian</td>
<td>no</td>
<td>PIB, S370FPIB, S370FIBU</td>
</tr>
<tr>
<td>big endian</td>
<td>little endian</td>
<td>yes</td>
<td>IBR</td>
</tr>
<tr>
<td>big endian</td>
<td>little endian</td>
<td>no</td>
<td>PIBR</td>
</tr>
<tr>
<td>little endian</td>
<td>big endian</td>
<td>yes</td>
<td>IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>big endian</td>
<td>no</td>
<td>PIBR</td>
</tr>
<tr>
<td>little endian</td>
<td>little endian</td>
<td>yes</td>
<td>IB or IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>little endian</td>
<td>no</td>
<td>PIB or PIBR</td>
</tr>
<tr>
<td>big endian</td>
<td>either</td>
<td>yes</td>
<td>S370FIB</td>
</tr>
<tr>
<td>big endian</td>
<td>either</td>
<td>no</td>
<td>S370FPIB</td>
</tr>
</tbody>
</table>
### 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**

<table>
<thead>
<tr>
<th>Language</th>
<th>2 Bytes</th>
<th>4 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>IB2., IBR2., PIB2., PIBR2.,</td>
<td>IB4., IBR4., PIB4., PIBR4.,</td>
</tr>
<tr>
<td></td>
<td>S370FIB2., S370FIBU2.,</td>
<td>S370FIBU4., S370FIBU4.,</td>
</tr>
<tr>
<td>PL/I</td>
<td>FIXED BIN(15)</td>
<td>FIXED BIN(31)</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>INTEGER*2</td>
<td>INTEGER*4</td>
</tr>
<tr>
<td>COBOL</td>
<td>COMP PIC 9(4)</td>
<td>COMP PIC 9(8)</td>
</tr>
<tr>
<td>IBM assembler</td>
<td>H</td>
<td>F</td>
</tr>
<tr>
<td>C</td>
<td>short</td>
<td>long</td>
</tr>
</tbody>
</table>

### 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**

<table>
<thead>
<tr>
<th>Primary Informat Name</th>
<th>Informat Alias(es)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAw.d</td>
<td>DOLLARw.d</td>
</tr>
<tr>
<td>COMMAXw.d</td>
<td>DOLLARXw.d</td>
</tr>
<tr>
<td>w.d</td>
<td>BESTw.d, Dw.d, Fw.d, Ew.d</td>
</tr>
<tr>
<td>$w.</td>
<td>$Fw.</td>
</tr>
</tbody>
</table>

### List of Informats

$ASCIIw.$
$BINARYw.
$CBw.
$CHArw.
$CHARZBw.
$EBCDICw.
$HEXw.
$KANJ Iw.
$KANJ IXw.
$OCTALw.
$PHEXw.
$QUOTEw.
$REVERJ w.
$REVERSw.
$UPCASEw.
$VARYINGw.
$w.
BINARyw.d
BITSw.d
BZW.d
CBw.d
COMMAw.d
COMMAXw.d
DATEw.
DATETIMEw.
DDMMYYw.
Ew.d
EURDFDEw.
EURDFDTw.
EURDFTYW.
FLOATw.
HEXw.
IBw.d
IBRw.d
IEEw.d
JULIANw.
MINGUOW.
MMDDYYW.
MONYYw.
MSECw.
NENGOw.
NUMXw.d
OCTALw.d
PDw.d
PDJ ULGw.
PDJ ULIw.
PDTIMEw.
PERCENTw.d
PIBw.d
PIBRw.d
PKw.d
PUNCHd.
RBw.d
RMFDURw.
RMFSTAMPw.
ROWw.d
SMFSTAMPw.
S370FFw.d
S370FIBw.d
S370FIBUw.d
S370FPDw.d
S370FPDUw.d
S370FPIBw.d
S370FRBw.d
S370FZDw.d
S370FZDLw.d
S370FZDSw.d
S370FZDTw.d
S370FZDUw.d
TIMEw.
TODSTAMPw.
TUw.
VAXRBw.d
  w.d
YENw.d
YYMDDw.
YYMMNw.
YYQw.
ZDw.d
ZDBw.d
ZDVw.d

Informats by Category

There are five categories of informats in SAS:

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

For information on reading and writing 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

<table>
<thead>
<tr>
<th>Category</th>
<th>Informat</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td><code>$ASCIIw.</code> on page 612</td>
<td>Converts ASCII character data to native format</td>
</tr>
<tr>
<td></td>
<td><code>$BINARYw.</code> on page 613</td>
<td>Converts binary data to character data</td>
</tr>
<tr>
<td></td>
<td><code>$CHARw.</code> on page 615</td>
<td>Reads character data with blanks</td>
</tr>
<tr>
<td></td>
<td><code>$CHARZBw.</code> on page 616</td>
<td>Converts binary 0s to blanks</td>
</tr>
<tr>
<td></td>
<td><code>$EBCDICw.</code> on page 617</td>
<td>Converts EBCDIC character data to native format</td>
</tr>
<tr>
<td></td>
<td><code>$HEXw.</code> on page 618</td>
<td>Converts hexadecimal data to character data</td>
</tr>
<tr>
<td></td>
<td><code>$OCTALw.</code> on page 619</td>
<td>Converts octal data to character data</td>
</tr>
<tr>
<td></td>
<td><code>$PHEXw.</code> on page 620</td>
<td>Converts packed hexadecimal data to character data</td>
</tr>
<tr>
<td></td>
<td><code>$QUOTEw.</code> on page 621</td>
<td>Removes matching quotation marks from character data</td>
</tr>
<tr>
<td></td>
<td><code>$REVERJw.</code> on page 622</td>
<td>Reads character data from right to left and preserves blanks</td>
</tr>
</tbody>
</table>
Chapter 5

“$REVERSw.” on page 623  Reads character data from right to left and left aligns

“$UPCASEw.” on page 624  Converts character data to uppercase

“$VARYINGw.” on page 624  Reads character data of varying length

“$w.” on page 626  Reads standard character data

Column binary

“$CBw.” on page 614  Reads standard character data from column-binary files

“CBw.d” on page 630  Reads standard numeric values from column-binary files

“PUNCH.d” on page 672  Reads whether a row of column-binary data is punched

“ROWw.d” on page 677  Reads a column-binary field down a card column

Date and time

“DATEw.” on page 633  Reads date values in the form ddmmmyy or ddmmmyyyy

“DATETIMEw. “on page 634  Reads datetime values in the form ddmmmyy hh:mm:ss.ss or ddmmmyyyy hh:mm:ss.ss

“DDMMYYw.” on page 636  Reads date values in the form ddmmmyy or ddmmmyyyy

“EURDFDEw. “on page 639  Reads international date values

“EURDFDTw.” on page 641  Reads international datetime values in the form ddmmmyy hh:mm:ss.ss or ddmmmyyyy hh:mm:ss.ss

“EURDFMYw. “on page 643  Reads month and year date values in the form mmmyy or mmmyyyy

“JULIANw.” on page 651  Reads J ulian dates in the form yyyddd or yyyyddd

“MINGUOw.” on page 652  Reads dates in Taiwanese form

“MMDDYYw.” on page 653  Reads date values in the form mmddyy or mmddyyyy

“MONYYw.” on page 655  Reads month and year date values in the form mmmyy or mmmyyyy

“MSECw.” on page 657  Reads TIME MIC values

“NENGOw. “on page 658  Reads J apanese date values in the form eymmd

“PDJ ULGw.” on page 663  Reads packed J ulian date values in the hexadecimal form yyydddDF for IBM

“PDJ ULIw.” on page 664  Reads packed J ulian dates in the hexadecimal format ccyydddDF for IBM

“PDTIMEw.” on page 666  Reads packed decimal time of SMF and RMF records

“RMFDURw. “on page 674  Reads duration intervals of RMF records

“RMFSTAMPw.” on page 676  Reads time and date fields of RMF records

“SHRSTAMPw. “on page 679  Reads date and time values of SHR records

“SMFSTAMPw. “on page 680  Reads time and date values of SMF records

“TIMEw.” on page 695  Reads hours, minutes, and seconds in the form hh:mm:ss.ss

“TODSTAMPw.” on page 697  Reads an eight-byte time-of-day stamp
<table>
<thead>
<tr>
<th>Informat</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TUw.</code></td>
<td>Reads timer units</td>
</tr>
<tr>
<td><code>YYMMDDw.</code></td>
<td>Reads date values in the form yymmdd or yyyyymmdd</td>
</tr>
<tr>
<td><code>YYMMNw.</code></td>
<td>Reads date values in the form yyyymm or yymm</td>
</tr>
<tr>
<td><code>YYQw.</code></td>
<td>Reads quarters of the year</td>
</tr>
<tr>
<td><code>BINARYw.d</code></td>
<td>Converts positive binary values to integers</td>
</tr>
<tr>
<td><code>BITSw.d</code></td>
<td>Extracts bits</td>
</tr>
<tr>
<td><code>BZw.d</code></td>
<td>Converts blanks to 0s</td>
</tr>
<tr>
<td><code>COMMAw.d</code></td>
<td>Removes embedded characters</td>
</tr>
<tr>
<td><code>COMMAXw.d</code></td>
<td>Removes embedded characters</td>
</tr>
<tr>
<td><code>Ew.d</code></td>
<td>Reads numeric values that are stored in scientific notation and double-precision scientific notation</td>
</tr>
<tr>
<td><code>FLOATw.d</code></td>
<td>Reads a native single-precision, floating-point value and divides it by 10 raised to the dth power</td>
</tr>
<tr>
<td><code>HEXw.</code></td>
<td>Converts hexadecimal positive binary values to either integer (fixed-point) or real (floating-point) binary values</td>
</tr>
<tr>
<td><code>IBw.d</code></td>
<td>Reads native integer binary (fixed-point) values, including negative values</td>
</tr>
<tr>
<td><code>IBRw.d</code></td>
<td>Reads integer binary (fixed-point) values in Intel and DEC formats</td>
</tr>
<tr>
<td><code>IEEEw.d</code></td>
<td>Reads an IEEE floating-point value and divides it by 10 raised to the dth power</td>
</tr>
<tr>
<td><code>NUMXw.d</code></td>
<td>Reads numeric values with a comma in place of the decimal point</td>
</tr>
<tr>
<td><code>OCTALw.d</code></td>
<td>Converts positive octal values to integers</td>
</tr>
<tr>
<td><code>PDw.d</code></td>
<td>Reads data that are stored in IBM packed decimal format</td>
</tr>
<tr>
<td><code>PERCENTw.d</code></td>
<td>Reads percentages as numeric values</td>
</tr>
<tr>
<td><code>Pibw.d</code></td>
<td>Reads positive integer binary (fixed-point) values</td>
</tr>
<tr>
<td><code>PIBRw.d</code></td>
<td>Reads positive integer binary (fixed-point) values in Intel and DEC formats</td>
</tr>
<tr>
<td><code>PKw.d</code></td>
<td>Reads unsigned packed decimal data</td>
</tr>
<tr>
<td><code>RBw.d</code></td>
<td>Reads numeric data that are stored in real binary (floating-point) notation</td>
</tr>
<tr>
<td><code>S370FFw.d</code></td>
<td>Reads EBCDIC numeric data</td>
</tr>
<tr>
<td><code>S370FIBw.d</code></td>
<td>Reads integer binary (fixed-point) values, including negative values, in IBM mainframe format</td>
</tr>
<tr>
<td><code>S370FIBUw.d</code></td>
<td>Reads unsigned integer binary (fixed-point) values in IBM mainframe format</td>
</tr>
<tr>
<td><code>S370FPDw.d</code></td>
<td>Reads packed data in IBM mainframe format</td>
</tr>
<tr>
<td><code>S370FPUw.d</code></td>
<td>Reads unsigned packed decimal data in IBM mainframe format</td>
</tr>
</tbody>
</table>
"S370FPIBw.d" on page 687  Reads positive integer binary (fixed-point) values in IBM mainframe format

"S370F RBw.d" on page 688  Reads real binary (floating-point) data in IBM mainframe format

"S370FZDw.d" on page 690  Reads zoned decimal data in IBM mainframe format

"S370FZDLw.d" on page 691  Reads zoned decimal leading-sign data in IBM mainframe format

"S370FZDSw.d" on page 692  Reads zoned decimal separate leading-sign data in IBM mainframe format

"S370FZDTw.d" on page 693  Reads zoned decimal separate trailing-sign data in IBM mainframe format

"S370FZDUw.d" on page 694  Reads unsigned zoned decimal data in IBM mainframe format

"VAXRBw.d " on page 699  Reads real binary (floating-point) data in VMS format

"w.d" on page 699  Reads standard numeric data

"YENw.d" on page 700  Removes embedded yen signs, commas, and decimal points

"ZDw.d " on page 706  Reads zoned decimal data

"ZDBw.d" on page 707  Reads zoned decimal data in which zeros have been left blank

"ZDVw.d" on page 708  Reads and validates zoned decimal data

---

**Dictionary**

**$ASCIIw.**

Converts ASCII character data to native format

**Category:** Character

**Syntax**

$ASCIIw.

**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, $ASCIIw. converts ASCII data to EBCDIC.
- On all other systems, $ASCIIw. behaves like the $CHARw. informat except that the default length is different.

Examples

input @1 name $ascii3.;

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>EBCDIC</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc</td>
<td>818283</td>
<td>616263</td>
</tr>
<tr>
<td>ABC</td>
<td>C1C2C3</td>
<td>414243</td>
</tr>
<tr>
<td>()</td>
<td>4D5D5E</td>
<td>28293B</td>
</tr>
</tbody>
</table>

* 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.

$BINARYw.

Converts binary data to character data

Category: Character

Syntax

$BINARYw.

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 $BINARYw. reads becomes one character value stored in a variable.

If w<8, $BINARYw. 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 \( 8 \), \$$BINARYw.$$ \text{ reads up to the largest multiple of } 8 \text{ that is less than } w \text{ before converting the data.}

**Default:** 8  
**Range:** 1–32767

**Details**

The \$$BINARYw.$$ informat does not interpret actual binary data, but it converts a string of characters that contains only 0s or 1s as though it is actual binary information. Therefore, use only the character digits 1 and 0 in the input, with no embedded blanks. \$$BINARYw.$$ 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 \$$BINARYw.$$ informat converts the string to its equivalent character value.

**Comparisons**

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

**Examples**

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

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100110001001101</td>
<td>LM</td>
</tr>
</tbody>
</table>

\$$CBw.$$  

Reads standard character data from column-binary files  
Category: Column binary

**Syntax**

\$$CBw.$$  

**Syntax Description**

\( w \)  

specifies the width of the input field.
**Details**

The `$CBw.` 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**

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

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>EBCDIC</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>200A + N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* 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:
- “CBw.d” on page 630
- “PUNCH.d” on page 672
- “ROWw.d” on page 677

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

---

**$CHARw.**

Reads character data with blanks

**Category:** Character

**Syntax**

`$CHARw.`

**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 $CHARw. 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 $CHARw. 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 $CHARw. informat is almost identical to the $w. informat. However $CHARw. 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:

<table>
<thead>
<tr>
<th>Language</th>
<th>Character Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>$CHAR8.</td>
</tr>
<tr>
<td>IBM 370 assembler</td>
<td>CL8</td>
</tr>
<tr>
<td>C</td>
<td>char [8]</td>
</tr>
<tr>
<td>COBOL</td>
<td>PIC x(8)</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>A8</td>
</tr>
<tr>
<td>PL/I</td>
<td>CHAR(8)</td>
</tr>
</tbody>
</table>

**Examples**

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

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results*</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZbb</td>
<td>XYZbb</td>
</tr>
<tr>
<td>bXYZbb</td>
<td>bXYZbb</td>
</tr>
<tr>
<td>bb.bb</td>
<td>bb.bb</td>
</tr>
<tr>
<td>bxyzbb</td>
<td>bxyzbb</td>
</tr>
</tbody>
</table>

* The character b represents a blank space.

$CHARZBw.$

Converts binary 0s to blanks
Syntax

$CHARZBw.

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 $CHARZBw. informat does not trim leading and trailing blanks in character data before it stores values.

Comparisons

The $CHARZBw. informat is identical to the $CHARw. informat except that $CHARZBw. converts any byte that contains a binary 0 to a blank character.

Examples

input @1 name $charzb5.;

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBCDIC</td>
<td>ASCII</td>
</tr>
<tr>
<td>E7E8E90000</td>
<td>58595A0000</td>
</tr>
<tr>
<td>00E7E8E900</td>
<td>0058595A00</td>
</tr>
<tr>
<td>00E700E8E9</td>
<td>005800595A</td>
</tr>
</tbody>
</table>

* 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 b represents a blank space.
Syntax
$EBCDICw.

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, $EBCDICw. behaves like the $CHARw. informat.
- On all other systems, $EBCDICw. converts EBCDIC data to ASCII.

Examples

input @1 name $ebcdic3.

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>ASCII</th>
<th>EBCDIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>qrs</td>
<td>717273</td>
<td>9899A2</td>
</tr>
<tr>
<td>qrs</td>
<td>515253</td>
<td>D8D9E2</td>
</tr>
<tr>
<td>+;&gt;</td>
<td>2B3B3E</td>
<td>4E5E6E</td>
</tr>
</tbody>
</table>

* 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.

$HEXw.

Converts hexadecimal data to character data
Category: Character

Syntax
$HEXw.
Syntax Description

w

specifies the number of digits of hexadecimal data.

If \( w = 1 \), $\text{HEX}w \) pads a trailing hexadecimal 0. If \( w \) is an odd number that is greater than 1, then $\text{HEX}w \) reads \( w - 1 \) hexadecimal characters.

Default: 2
Range: 1–32767

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

Comparisons
The $\text{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 $\text{hex}4.;
```

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>ASCII</th>
<th>EBCDIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>6c6c</td>
<td>11</td>
<td>%%</td>
</tr>
</tbody>
</table>

$\text{OCTAL}w\)  

Converts octal data to character data

Category: Character

Syntax

$\text{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 $\text{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. `$OCTALw` treats every three digits of octal data as one digit of character data, ignoring the extra bit.

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

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

**Comparisons**

The `$OCTALw` informat reads octal data and converts them into the numeric equivalents.

**Examples**

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

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBCDIC</td>
<td>ASCII</td>
</tr>
<tr>
<td>114</td>
<td>&lt;</td>
</tr>
</tbody>
</table>

**$PHEXw**

Converts packed hexadecimal data to character data

**Category:** Character

**Syntax**

```
$PHEXw.
```

**Syntax Description**

**w**

specifies the number of bytes in the input.

When you use `$PHEXw` 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 \textit{w}. In general, a character variable whose length is implicitly defined with \$PHEXw. has a length of $2w-1$.

\textbf{Default:} 2
\textbf{Range:} 1–32767

\section*{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 \$PHEXw. informat returns a character value and treats the value of the sign nibble as if it were \texttt{X'F'}, regardless of its actual value.

\section*{Comparisons}

The \texttt{PD\textit{w}.d}. informat reads packed decimal data and converts them to numeric data.

\section*{Examples}

\begin{verbatim}
input @1 devaddr $phex2.;
\end{verbatim}

\begin{tabular}{ll}
\textbf{Data Lines*} & \textbf{Results} \\
0001111000001111 & \texttt{1E0} \\
\end{tabular}

\* 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 \texttt{1EOF}.

\section*{$\texttt{QUOTE}w.$}

\texttt{\textbf{Remove$}}$ \texttt{matching quotation marks from character data}

\textbf{Category:} Character

\section*{Syntax}

\texttt{$\texttt{QUOTE}w.$}

\section*{Syntax Description}

\texttt{w}

\texttt{\textbf{w}} specifies the width of the input field.

\textbf{Default:} 8 if the length of the variable is undefined; otherwise, the length of the variable
\textbf{Range:} 1–32767
Examples

input @1 name $quote7.;

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>'SAS'</td>
<td>SAS</td>
</tr>
<tr>
<td>&quot;SAS&quot;</td>
<td>SAS</td>
</tr>
<tr>
<td>&quot;SAS’s&quot;</td>
<td>SAS’s</td>
</tr>
</tbody>
</table>

$REVERJw.

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.;

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results*</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCD</td>
<td>bbbDCBA</td>
</tr>
<tr>
<td>ABCD</td>
<td>DCBAabb</td>
</tr>
</tbody>
</table>

* The character b 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.;

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results*</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCD</td>
<td>DCBAabb</td>
</tr>
<tr>
<td>ABCD</td>
<td>DCBAabb</td>
</tr>
</tbody>
</table>

* The character b represents a blank space.
**$UPCASEw.**

Converts character data to uppercase

**Category:** Character

**Syntax**

$UPCASEw.$

**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**

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

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>sas</td>
<td>SAS</td>
</tr>
</tbody>
</table>

---

**$VARYINGw.**

Reads character data of varying length

**Valid:** in a DATA step

**Category:** Character
**Syntax**

$\textit{VARYING}w. \textit{length-variable}$

**Syntax Description**

\textit{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

\textit{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 $\textit{VARYING}w.$ 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 $\textit{VARYING}w.$ when the length of a character value differs from record to record. After reading a data value with $\textit{VARYING}w.$, 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;
```

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>5shark</td>
<td>shark</td>
</tr>
<tr>
<td>3sunfish</td>
<td>sun</td>
</tr>
<tr>
<td>8bluefish</td>
<td>bluefish</td>
</tr>
</tbody>
</table>

* 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.

```sas
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 RECLLEN 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.

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATEL CHEMISTRY</td>
<td>PATEL CHEMISTRY</td>
</tr>
<tr>
<td>JOHNSON GEOLOGY</td>
<td>JOHNSON GEOLOGY</td>
</tr>
<tr>
<td>WILCOX ART</td>
<td>WILCOX ART</td>
</tr>
</tbody>
</table>

$w.$

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 $\text{CHAR}w.$ informat. However, $\text{CHAR}w.$ does not trim leading blanks nor does it convert a single period in an input field to a blank, while $w.$ does both.

**Examples**

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

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results*</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZXYZbb</td>
<td>X YZ XbYZb</td>
</tr>
</tbody>
</table>

* The character b 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. BINARYw.d ignores leading and trailing blanks.

BINARYw.d cannot read negative values. It treats all input values as positive (unsigned).

Examples

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

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001111</td>
<td>1.5</td>
</tr>
</tbody>
</table>

BITSw.d

Extracts bits

Category: Numeric

Syntax

BITSw.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 BITSw.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;
```

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results*</th>
</tr>
</thead>
<tbody>
<tr>
<td>B8</td>
<td>8</td>
</tr>
</tbody>
</table>

* 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.

**BZw.d**

Converts blanks to 0s

Category: Numeric

**Syntax**

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

The BZw.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 BZw.d informat ignores blanks between a minus sign and a numeric value in an input field.

The BZw.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 BZw.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 BZw.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 Ew.d informat instead.

**Examples**

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

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>3400</td>
</tr>
<tr>
<td>-2</td>
<td>-200</td>
</tr>
<tr>
<td>-2 1</td>
<td>-201</td>
</tr>
</tbody>
</table>

**CBw.d**

*Reads standard numeric values from column-binary files*

**Category:** Column binary

**Syntax**

`CBw.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 CBw.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 CBw.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**

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

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>0009</td>
<td>9</td>
</tr>
</tbody>
</table>

* 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:

- "$CBw." on page 614
- "PUNCH.d" on page 672
- "ROWw.d" on page 677

---

**COMMAw.d**

Removes embedded characters

**Category:** Numeric

**Syntax**

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

Comparisons
The COMMAw.d informat operates like the COMMAXw.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.;

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1,000,000</td>
<td>1000000</td>
</tr>
<tr>
<td>(500)</td>
<td>-500</td>
</tr>
</tbody>
</table>

COMMAXw.d

Removes embedded characters

Category: Numeric

Syntax

COMMAXw.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.
**Details**

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

**Comparisons**

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

**Examples**

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

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1,000,000</td>
<td>1000000</td>
</tr>
<tr>
<td>(500)</td>
<td>-500</td>
</tr>
</tbody>
</table>

**DATEw.**

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

Category: Date and time

**Syntax**

```plaintext
DATEw.
```

**Syntax Description**

```plaintext
w
```

specifies the width of the input field.

*Default*: 7

*Range*: 7–32

**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.

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.;

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>16mar97</td>
<td>13589</td>
</tr>
<tr>
<td>16 mar 97</td>
<td>13589</td>
</tr>
<tr>
<td>16-mar-1997</td>
<td>13589</td>
</tr>
</tbody>
</table>

See Also

Format:
   “DATEw.” on page 83
Function:
   “DATEw.” on page 83
System Option:
   “YEARCUTOFF=” on page 1116

DATETIMEw.

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

Category: Date and time

Syntax

DATETIMEw.
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**

```plaintext
input date_and_time datetime20.;
```
### Data Lines Results

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>16mar97:11:23:07.4</td>
<td>1174130587.4</td>
</tr>
<tr>
<td>16mar1997/11:23:07.4</td>
<td>1174130587.4</td>
</tr>
<tr>
<td>16mar1997/11:23 PM</td>
<td>1174173780.0</td>
</tr>
</tbody>
</table>

### See Also

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

Concepts

Formats:
- "DATEw." on page 83
- "DATETIMEw.d" on page 86
- "TIMEw.d" on page 166

Function:
- "DATETIME" on page 308

Informats:
- "DATEw." on page 633
- "TIMEw." on page 695

System Option:
- "YEARCUTOFF=" on page 1116

---

### DDMMYYw.

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

Category: Date and time

---

### Syntax

**DDMMYYw.**

---

### 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
    ---------------+---------
    160397         13589
### Data Lines Results

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>16/03/97</td>
<td>13589</td>
</tr>
<tr>
<td>16 03 1997</td>
<td>13589</td>
</tr>
</tbody>
</table>

### See Also

- **Formats:**
  - “DATEw.” on page 83
  - “DDMMYYw.” on page 88
  - “MMDDYYw.” on page 125
  - “YYMMDDw.” on page 182

- **Function:**
  - “MDY” on page 420

- **Informats:**
  - “DATEw.” on page 633
  - “MMDDYYw.” on page 653
  - “YYMMDDw.” on page 701

- **System Option:**
  - “YEARCUTOFF=” on page 1116

---

**Ew.d**

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

**Category:** Numeric

### Syntax

\[ Ew.d \]

### 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.;
```

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.257E3</td>
<td>1257</td>
</tr>
<tr>
<td>12d3</td>
<td>12000</td>
</tr>
</tbody>
</table>

**EURDFDEw.**

Reads international date values
Category: Date and time

**Syntax**
EURDFDEw.

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 ddmmyy 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 1038 for the list of language prefixes on page 1038. 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.;
Informats

Data Lines Results

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>01abr1998</td>
<td>13970</td>
</tr>
<tr>
<td>01-abr-98</td>
<td>13970</td>
</tr>
</tbody>
</table>

See Also

Format:

“EURDFDEw.” on page 97

Function:

“DATEw.” on page 83

Informs:

“DATEw.” on page 633
“EURDFDTw.” on page 641
“EURDFMYw.” on page 643

System Options:

“DFLANG=” on page 1038
“YEARCUTOFF=” on page 1116

---

EURDFDTw.

Reads international datetime values in the form ddmmyy hh:mm:ss.ss or ddmmyyyy hh:mm:ss.ss

Category: Date and time

Syntax

EURDFDTw.

Syntax Description

w

specifies the width of the input field.

Default: 18

Range: 13–40

Details

The datetime values must be in the form ddmmyy or ddmmyyyy, 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.

EURDFDTw. 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 1038
for the list of language prefixes on page 1038. 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.;

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>23dez97:10:03:17.2</td>
<td>1198490597.2</td>
</tr>
<tr>
<td>23dez1997:10:03:17.2</td>
<td>1198490597.2</td>
</tr>
</tbody>
</table>
See Also

Formats:
“DATEw.” on page 83
“DATETIMEw.d” on page 86
“TIMEw.d” on page 166

Function:
“DATETIME” on page 308

Informs:
“DATETIMEw. “ on page 634
“EURDFDEw. “ on page 639
“EURDFMYw. “ on page 643

System Options:
“DFLANG=” on page 1038
“YEARCUTOFF=” on page 1116

---

**EURDFMY w.**

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

Category: Date and time

---

**Syntax**

**EURDFMYw.**

**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 EURDFMYw. 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 1038 for the list of language prefixes on page 1038. 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.;

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr1998</td>
<td>13970</td>
</tr>
<tr>
<td>avr 98</td>
<td>13970</td>
</tr>
</tbody>
</table>
See Also

Formats:
“DDMMYYw.” on page 88
“MMDDYYw.” on page 125
“MONYYw.” on page 132
“YYMMDDw.” on page 182

Functions:
“MONTHw.” on page 131
“YEAR” on page 592

Informats:
“EURDFDEw.” on page 639
“EURDFDTw.” on page 641
“MONYYw.” on page 655

System Options:
“DFLANG=” on page 1038
“YEARCUTOFF=” on page 1116

FLOAT\(w.d\)

Reads a native single-precision, floating-point value and divides it by \(10^{d}\) raised to the \(d\)th power

Category: Numeric

Syntax

\[\text{FLOAT}w.d\]

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:

<table>
<thead>
<tr>
<th>Language</th>
<th>Float Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>FLOAT4.</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>REAL*4</td>
</tr>
<tr>
<td>C</td>
<td>float</td>
</tr>
<tr>
<td>IBM 370 ASM</td>
<td>E</td>
</tr>
<tr>
<td>PL/I</td>
<td>FLOAT BIN(21)</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
input x float4.;
```

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>3F800000</td>
<td>1</td>
</tr>
</tbody>
</table>

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

**HEXw.**

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

Category: Numeric

**Syntax**

**HEXw.**

**Syntax Description**
**IBw.d**

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

**Category:** Numerical

---

**Syntax**

`IBw.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 IBw.d informat reads integer binary (fixed-point) values, including negative values represented in two's complement notation. IBw.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 604.

Comparisons

The IBw.d and PIBw.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 IBRw.d and PIBRw.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 605.

To view a table that compares integer binary notation in several programming languages, see Table 5.2 on page 606.

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.

\[ x = \text{input('0080'x,ib2.);} \]
\[ y = \text{input('8000'x,ib2.);} \]

<table>
<thead>
<tr>
<th>SAS Statements</th>
<th>Results on Big Endian Platforms</th>
<th>Results on Little Endian Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>128</td>
<td>-32768</td>
</tr>
<tr>
<td>put y=;</td>
<td>-32768</td>
<td>128</td>
</tr>
</tbody>
</table>

See Also

Informat:

“IBRw.d” on page 648

IBRw.d

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

Syntax

IBRw.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 IBRw.d informat reads integer binary (fixed-point) values, including negative values that are represented in two's complement notation. IBRw.d reads integer binary values that are generated by and for Intel and DEC platforms. Use IBRw.d to read integer binary data from Intel or DEC environments in other operating environments. The IBRw.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 604.

Comparisons

The IBw.d and PIBw.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 IBRw.d and PIBRw.d informats are used to read little endian integers in any operating environment.

On Intel and DEC operating environments, the IBw.d and IBRw.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 605.

To view a table that compares integer binary notation in several programming languages, see Table 5.2 on page 606.

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.);
IEEE\textsubscript{w.d} reads an IEEE floating-point value and divides it by 10 raised to the \textit{d}th power.

\textbf{Syntax}

\texttt{IEEEw.d}

\textbf{Syntax Description}

\textbf{w}

- specifies the width of the input field.
  - Default: 8
  - Range: 2–8
  - Tip: If \textit{w} is 8, an IEEE double-precision, floating-point number is read. If \textit{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 \textit{w} is 4, an IEEE single-precision, floating-point number is read. If \textit{w} is 3, an IEEE single-precision, floating-point number is read, which assumes truncation of one byte.

\textbf{d}

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

\textbf{Details}

The \texttt{IEEEw.d} informat is useful in operating environments where IEEE is the floating-point representation that is used. In addition, you can use the \texttt{IEEEw.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 IEEEw.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.;
```

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>3F800000</td>
<td>1</td>
</tr>
<tr>
<td>3FF00000000</td>
<td>1</td>
</tr>
</tbody>
</table>

* 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.

**JULIANw.**

Reads Julian dates in the form yyddd or yyyyddd

Category: Date and time

**Syntax**

```
JULIANw.
```

**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.;
```

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results*</th>
</tr>
</thead>
<tbody>
<tr>
<td>97075</td>
<td>13589</td>
</tr>
<tr>
<td>1997075</td>
<td>13589</td>
</tr>
</tbody>
</table>

* The input values correspond to the seventy-fifth day of 1997, which is March 16.

See Also

Format:

“JULIANw.” on page 123

Functions:

“DATEJUL” on page 306

“JULDATE” on page 406

System Option:

“YEARCUTOFF=” on page 1116

MINGUOW.

Reads dates in Taiwanese form

Category: Date and time

Syntax

MINGUOW.

Syntax Description
w

specifies the width of the input field.

_default_: 6
_range_: 6–10

**Details**

The general form of a Taiwanese date is `yyyyymmdd`, 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 `YYMMDDw.` 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**

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

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>49/01/01</td>
<td>01JAN1960</td>
</tr>
<tr>
<td>891215</td>
<td>15DEC2000</td>
</tr>
<tr>
<td>103-01-01</td>
<td>01JAN2014</td>
</tr>
</tbody>
</table>

**MMDDYYw.**

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

**Category:** Date and time

**Syntax**

`MMDDYYw.`
### 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.;
```

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>031697</td>
<td>13589</td>
</tr>
</tbody>
</table>
### MONYYw.

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

**Category:** Date and time

**Syntax**

```
MONYYw.
```

**Syntax Description**

`w`

specifies the width of the input field.

**Default:** 5

**Range:** 5–32

---

### See Also

**Formats:**

- “DATEw.” on page 83
- “DDMMYYw.” on page 88
- “MMDDYYw.” on page 125
- “YYMMDDw.” on page 182

**Functions:**

- “DAY” on page 308
- “MDY” on page 420
- “MONTHw.” on page 131
- “YEAR” on page 592

**Informats:**

- “DATEw.” on page 633
- “DDMMYYw.” on page 636
- “YYMMDDw.” on page 701

**System Option:**

- “YEARCUTOFF=” on page 1116
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 MONYYw. 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 YEARCUTOFF = system option.

Examples

    input month_and_year monyy7.;
Data Lines Results

---+---1---+

mar 97 13574
mar1997 13574

See Also

Formats:
“DDMMYYw.” on page 88
“MMDDYYw.” on page 653
“MONYYw.” on page 132
“YYMMDDw.” on page 182

Functions:
“MONTH” on page 428
“YEAR” on page 592

Informats:
“DDMMYYw.” on page 636
“MMDDYYw.” on page 653
“YYMMDDw.” on page 701

System Option:
“YEARCUTOFF=” on page 1116

MSECw.

Reads TIME MIC values
Category: Date and time

Syntax
MSECw.

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 MSECw. informat reads time values that are produced by IBM mainframe operating environments and converts the time values to SAS time values.

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

Comparisons

The MSECw. and TODSTAMPw. informats both read IBM time-of-day clock values, but the MSECw. informat assigns a time value to a variable, and the TODSTAMPw. 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:

“TODSTAMPw.” on page 697

NENGOw.

Reads Japanese date values in the form eyymmd

Category:  Date and time

```
NENGOw.
```

Syntax Description

```
w
```

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

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

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>s640107</td>
<td>07JAN1989</td>
</tr>
<tr>
<td>h.10108</td>
<td>08JAN1998</td>
</tr>
<tr>
<td>10/01/08</td>
<td>08JAN1998</td>
</tr>
</tbody>
</table>

See Also

Format:

```
NENGOw.
```

NUMXw.d

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

Category: Numeric

Syntax

```
NUMXw.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 \( \text{NUMX}w.d \) informat reads numeric values and interprets a comma as a decimal point.

Comparisons

The \( \text{NUMX}w.d \) 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.;
```

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>896,48</td>
<td>896.48</td>
</tr>
<tr>
<td>3064,1</td>
<td>3064.1</td>
</tr>
<tr>
<td>6489</td>
<td>6489</td>
</tr>
</tbody>
</table>

See Also

Formats:

"NUMXw.d" on page 135  
"w.d" on page 170

\( \text{OCTAL}w.d \)

Converts positive octal values to integers  
**Category:** Numeric
Syntax

OCTAL \texttt{w.d}

Syntax Description

\texttt{w}

specifies the width of the input field.

\textbf{Default:} 3

\textbf{Range:} 1–24

\texttt{d}

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

\textbf{Range:} 1–31

\textbf{Restriction:} must be greater than or equal to the \texttt{w} value.

Details

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

OCTAL \texttt{w.d} cannot read negative values. It treats all input values as positive (unsigned).

Examples

\begin{verbatim}
input @1 value octal3.1;
\end{verbatim}

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>0177</td>
<td>12.7</td>
</tr>
</tbody>
</table>

\subsection*{PD\texttt{w.d}}

Reads data that are stored in IBM packed decimal format

\textbf{Category:} Numeric

Syntax

PD\texttt{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 PDw.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, PDw.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:

<table>
<thead>
<tr>
<th>Language</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>PD4.</td>
</tr>
<tr>
<td>COBOL</td>
<td>COMP-3 PIC S9(7)</td>
</tr>
<tr>
<td>IBM 370 Assembler</td>
<td>PL4</td>
</tr>
<tr>
<td>PL/I</td>
<td>FIXED DEC</td>
</tr>
</tbody>
</table>

Examples

**Example 1: Reading Packed Decimal Data**

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

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000128C</td>
<td>128</td>
</tr>
</tbody>
</table>

* 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.);
```
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, 1997.

**PDJULGw.**

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

Category: Date and time

**Syntax**

**PDJ ULGw.**

**Syntax Description**

*w* specifies the width of the input field.

**Default:** 4

**Range:** 4

**Details**

The PDJ ULGw. informat reads IBM packed Julian date values in the form of *yyyydddf*, 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**

```sas
input date pdjulg4.;
```
Data Line in Hexadecimal Results*

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998003F</td>
<td>13882</td>
</tr>
</tbody>
</table>

* SAS date value 13882 represents January 3, 1998.

See Also

Functions:

"DATEJUL" on page 306
"JULDATE" on page 406

Informats:

"JULIANw." on page 651
"PDJULIw." on page 664

Formats:

"JULDAYw." on page 122
"JULIANw." on page 123
"PDJULGw." on page 139
"PDJULIw." on page 140

System Option:

"YEARCUTOFF=" on page 1116

PDJULIw.

Reads packed Julian dates in the hexadecimal format ccyydddF for IBM

Category: Date and time

Syntax

PDJ ULIw.

Syntax Description

w

specifies the width of the input field.

Default: 4

Range: 4

Details

The PDJ ULIw. informat reads IBM packed Julian date values in the form ccyydddF, 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 PDJULIw 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.

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

**Examples**

```plaintext
input date pdjuli4.;
```

<table>
<thead>
<tr>
<th>Data Lines in Hexadecimal</th>
<th>Results*</th>
</tr>
</thead>
<tbody>
<tr>
<td>----+----1</td>
<td></td>
</tr>
<tr>
<td>0097001F</td>
<td>13515</td>
</tr>
<tr>
<td>0110015F</td>
<td>18277</td>
</tr>
</tbody>
</table>

* SAS date value 13515 is January 1, 1997. SAS date value 18277 is January 15, 2010.
See Also

Functions:

“DATEJ U L” on page 306
“J U LDATE” on page 406

Informats:

“J U LIANw.” on page 651
“PDJ U LGw.” on page 663

Formats:

“J U LDAYw.” on page 122
“J U LIANw.” on page 123
“PDJ U LGw.” on page 139
“PDJ U LIw.” on page 140

System Option:

“YE ARCUTOFF =” on page 1116

PDTIMEw.

Reads packed decimal time of SMF and RMF records

Category: Date and time

Syntax

PDTIMEw.

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 PDTIMEw. 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 0hhmmssF, 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, PDTIMEw. treats it as a missing value.

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

**Examples**

```sas
input begin pdtime4.;
```

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>0142225F</td>
<td>51745</td>
</tr>
</tbody>
</table>

* 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.

---

**PERCENTw.d**

Reads percentages as numeric values

**Category:** Numeric

**Syntax**

```sas
PERCENTw.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 PERCENTw.d informat converts the numeric portion of the input data to a number using the same method as the COMMAw.d informat. If a percent sign (%) follows the number in the input field, PERCENTw.d divides the number by 100.

Examples

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

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1% (20%)</td>
<td>0.01 -0.2</td>
</tr>
</tbody>
</table>

PIBw.d

Reads positive integer binary (fixed-point) values

Category: Numeric

Syntax

PIBw.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. PIBw.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 604.

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 PIBw.d informat treats all values as positive and includes the sign bit as part of the value.
- The PIBw.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 IBw.d and PIBw.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 IBw.d and PIBw.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 605.

To view a table that compares integer binary notation in several programming languages, see Table 5.2 on page 606.

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.

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

<table>
<thead>
<tr>
<th>SAS Statements</th>
<th>Results on Big Endian Platforms</th>
<th>Results on Little Endian Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>put y=;</td>
<td>1</td>
<td>256</td>
</tr>
</tbody>
</table>

See Also

Informat:

“PIBRw.d” on page 669

PIBRw.d

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

Category: Numeric

Syntax

PIBRw.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. PIBRw.d reads positive integer binary values that have been generated by and for Intel and DEC operating environments. Use PIBRw.d to read positive integer binary data from Intel or DEC environments on other operating environments. The PIBRw.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 604.

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 PIBRw.d informat treats all values as positive and includes the sign bit as part of the value.
- The PIBRw.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 PIBw.d and PIBRw.d informats are equivalent.
- The IBw.d and PIBw.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 IBRw.d and PIBRw.d informats are used to read little endian integers in any operating environment.

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.

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

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>001234</td>
<td>1234</td>
</tr>
</tbody>
</table>

* 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**

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>SAS Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-7-8</td>
<td>input x punch.12</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>input x punch.11</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>input x punch0.7</td>
<td>1</td>
</tr>
</tbody>
</table>

* 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:
“$CBw.” on page 614
“CBw.d” on page 630
“ROWw.d” on page 677

RBw.d

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

Category: Numeric

Syntax

RBw.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 RBw.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:

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>RB4.</td>
<td>RB8.</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>REAL*4</td>
<td>REAL*8</td>
</tr>
<tr>
<td>C</td>
<td>float</td>
<td>double</td>
</tr>
</tbody>
</table>
Real Binary Notation

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM 370 assembler</td>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td>PL/I</td>
<td>FLOAT BIN(21)</td>
<td>FLOAT BIN(53)</td>
</tr>
</tbody>
</table>

**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.;
```

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>4280000000000000</td>
<td>128</td>
</tr>
</tbody>
</table>

* 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:

“IEEEw.d” on page 650

---

**RMFDURw.**

Reads duration intervals of RMF records

**Category:** Date and time

**Syntax**

`RMFDURw.`

**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 RMFDURw. 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 mmsstttF, 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, RMFDURw. results in a missing value.

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

Examples

```
input dura rmfdur4.;
```

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>3552226F</td>
<td>2152.266</td>
</tr>
</tbody>
</table>

* 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:

“RMFSTAMPw.” on page 676
“SMFSTAMPw. “on page 680

RMFSTAMPw.
Reads time and date fields of RMF records
Category: Date and time

Syntax
RMFSTAMPw.

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 RMFSTAMPw. 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 0hhmmssFccyydddF, 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.

RMFSTAMPw. 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 RMFSTAMPw. informat and the PDTIMEw. informat read packed decimal values from RMF records. The RMFSTAMPw. informat reads both time and date values and results in a SAS datetime value. The PDTIMEw. 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.

ROWw.d
Reads a column-binary field down a card column
Category:  Column binary

Syntax
ROWw.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, \( \text{ROWw}.d \) assigns the variable a missing value and sets the automatic variable _ERROR_ to 1. If the field has no punches, \( \text{ROWw}.d \) assigns the variable a missing value.

\( \text{ROWw}.d \) can read fields across columns, continuing with row 12 of the new column and going down through the rest of the rows. After \( \text{ROWw}.d \) reads a field, the pointer moves to the next row.

**Examples**

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

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>----+----1</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

* 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:

"$CBw." on page 614
"CBw.d" on page 630
"PUNCH.d" on page 672

**SHRSTAMPw.**

Reads date and time values of SHR records

Category: Date and time

**Syntax**

**SHRSTAMPw.**

**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 SHRSTAMPw. 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 yyyydddFhmmsssth, 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 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**

```sas
input begin shrstamp8.;
```

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>0097239F12403576</td>
<td>1188304835.8</td>
</tr>
</tbody>
</table>

* 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 ccyydddF, 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 SMFSTAMPw. 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.;

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>0058DC0C0098200F</td>
<td>1216483835</td>
</tr>
</tbody>
</table>

* 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.

S370FFw.d

Reads EBCDIC numeric data
Category: Numeric

Syntax
S370FFw.d

Syntax Description

w
specifies the width of the input field.
**Examples**

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

### 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

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

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1F2F3</td>
<td>123</td>
</tr>
<tr>
<td>F2F4F0</td>
<td>240</td>
</tr>
</tbody>
</table>

* 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

S370FIBw.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 S370FIBw.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. S370FIBw.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 S370FIBw.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 604.

Comparisons

- If you use SAS on an IBM mainframe, S370FIBw.d and IBw.d are identical.
- S370FPIBw.d, S370FIBUw.d, and S370FIBw.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 605.

  To view a table that compares integer binary notation in several programming languages, see Table 5.2 on page 606.

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>128</td>
</tr>
</tbody>
</table>

See Also

Informats:

“S370FIBUw.d” on page 684, “S370FPIBw.d” on page 687

S370FIBUw.d

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

Category: Numeric

Syntax

S370FIBUw.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 S370FIBUw.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. S370FIBUw.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 S370FIBUw.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 604.
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 format to use with big endian and little endian integers, see Table 5.1 on page 605.
  
  To view a table that compares integer binary notation in several programming languages, see Table 5.2 on page 606.

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.

```sas
x=input('7F'x,s370fibul.);
y=input('F6'x,s370fibul.);
```

<table>
<thead>
<tr>
<th>SAS Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>127</td>
</tr>
<tr>
<td>put y=;</td>
<td>246</td>
</tr>
</tbody>
</table>

See Also

Informs:

- “S370FIB.w.d” on page 682, “S370FPIB.w.d” on page 687

S370FPD.w.d

Reads packed data in IBM mainframe format

Category: Numeric

Syntax

S370FPDw.d

Syntax Description

w

specifies the width of the input field.
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 S370FPDw.d to read packed decimal data from IBM mainframe files on other operating environments.

Comparisons
- If you use SAS on an IBM mainframe, the S370FPDw.d and the PDw.d informats are identical.
- The following table compares the equivalent packed decimal notation by programming language:

<table>
<thead>
<tr>
<th>Language</th>
<th>Packed Decimal Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>S370FIB4.</td>
</tr>
<tr>
<td>PL/I</td>
<td>FIXED DEC(7,0)</td>
</tr>
<tr>
<td>COBOL</td>
<td>COMP-3 PIC 9(7)</td>
</tr>
<tr>
<td>assembler</td>
<td>PL4</td>
</tr>
</tbody>
</table>

S370FPDw.d
Reads unsigned packed decimal data in IBM mainframe format

Category: Numeric

Syntax

S370FPDw.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 S370FPDUw.d on other operating environments to read unsigned packed decimal data from IBM mainframe files.

Comparisons

- The S370FPDUw.d informat is similar to the S370FPDw.d informat except that the S370FPDUw.d informat rejects all sign digits except F.
- The S370FPDUw.d informat is equivalent to the COBOL notation PIC 9(n) PACKED-DECIMAL, where the n value is the number of digits.

Examples

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

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345F</td>
<td>12345</td>
</tr>
</tbody>
</table>

* 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.

S370FPIBw.d

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

Category: Numeric

Syntax

S370FPIBw.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. S370FPIBw.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 S370FPIBw.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 604.

Comparisons

- If you use SAS on an IBM mainframe, S370FPIBw.d and PIBw.d are identical.
- S370FPIBw.d, S370FIBUw.d, and S370FIBw.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 605.

  To view a table that compares integer binary notation in several programming languages, see Table 5.2 on page 606.

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.

\[
x=\text{input}(\text{‘0100’x}, \text{s370fpib2.});
\]

SAS Statement Results

<table>
<thead>
<tr>
<th>put x=;</th>
<th>256</th>
</tr>
</thead>
</table>

See Also

Informats:

“S370FIBw.d” on page 682, “S370FIBUw.d” on page 684

S370FRBw.d

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

Syntax
S370FRBw.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 S370FRBw.d to read real binary data from IBM mainframe files on other operating environments.

Comparisons
- If you use SAS on an IBM mainframe, S370FRBw.d and RBw.d are identical.
- The following table shows the equivalent real binary notation for several programming languages:

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I</td>
<td>FLOAT BIN(21)</td>
<td>FLOAT BIN(53)</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>REAL*4</td>
<td>REAL*8</td>
</tr>
<tr>
<td>COBOL</td>
<td>COMP-1</td>
<td>COMP-2</td>
</tr>
</tbody>
</table>
Real Binary Notation

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>assembler</td>
<td>E</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>float</td>
<td>double</td>
</tr>
</tbody>
</table>

**See Also**

Informat:
“RBw.d” on page 673

**S370FZDw.d**

Reads zoned decimal data in IBM mainframe format

**Category:** Numeric

**Syntax**

S370FZDw.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 S370FZDw.d on other operating environments to read zoned decimal data from IBM mainframe files.

**Comparisons**

- If you use SAS on an IBM mainframe, S370FZDw.d and ZDw.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**

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

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1F2C3</td>
<td>123</td>
</tr>
<tr>
<td>F1F2D3</td>
<td>-123</td>
</tr>
</tbody>
</table>

* 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:

“ZDw.d “ on page 706

---

**S370FZDLw.d**

Reads zoned decimal leading-sign data in IBM mainframe format

**Category:** Numeric

**Syntax**

S370FZDLw.d

**Syntax Description**

```plaintext
w
```

specifies the width of the input field.

**Default:** 8

**Range:** 1–32

```plaintext
d
```

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

Details
Use S370FZDLw.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 S370FZDLw.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 s370fzd13.;

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1F2F3</td>
<td>123</td>
</tr>
<tr>
<td>D1F2F3</td>
<td>-123</td>
</tr>
</tbody>
</table>

* 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.

S370FZDSw.d

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

Category: Numeric

Syntax

S370FZDSw.d

Syntax Description

w

specifies the width of the input field.

Default: 8
Informats S370FZDTw.d

Range: 2–32

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

Details
Use S370FZDSw.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 S370FZDSw.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.;

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>4EFP2F3</td>
<td>123</td>
</tr>
<tr>
<td>60FP2F3</td>
<td>-123</td>
</tr>
</tbody>
</table>

* 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.

S370FZDTw.d

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

Category: Numeric

Syntax
S370FZDTw.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 S370FZDTw.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 S370FZDTw.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.;

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1F2F34E</td>
<td>123</td>
</tr>
<tr>
<td>F1F2F360</td>
<td>-123</td>
</tr>
</tbody>
</table>

* 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.

S370FZDUw.d

Reads unsigned zoned decimal data in IBM mainframe format

Category: Numeric

Syntax

S370FZDUw.d

Syntax Description
**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:** 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.;
```

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1F2F3</td>
<td>123</td>
</tr>
</tbody>
</table>

* 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.
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:
“HHMMw.d” on page 115
“HOURw.d” on page 117
“MMSSw.d” on page 128
“TIMEw.d” on page 166

Functions:
“HOUR” on page 388
“MINUTE” on page 422
“SECOND” on page 518
“TIME” on page 540

TODSTAMPw.

Reads an eight-byte time-of-day stamp

Category: Date and time

Syntax
TODSTAMPw.

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 TODSTAMPw. 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, TODSTAMPw. results in a missing value.

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

Examples

input btime todstamp8.;
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.

**TUw.**

Reads timer units

**Category:** Date and time

**Syntax**

TUw.

**Syntax Description**

\textit{w}

specifies the width of the input field.

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

**Details**

The TUw. 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 TUw. informat to read timer unit values that are produced by an IBM mainframe on other operating environments.

**Examples**

```sas
input btime tu4.;
```

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.
**VAXRBw.d**

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

Category: Numeric

**Syntax**

`VAXRBw.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 VAXRBw.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 VAXRBw.d and the RBw.d informats are identical.

**See Also**

Informat:

“RBw.d” on page 673

---

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

Examples

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

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>23 2300</td>
<td>23 23</td>
</tr>
<tr>
<td>23 2300</td>
<td>23 23</td>
</tr>
<tr>
<td>23 -2300</td>
<td>23 -23</td>
</tr>
<tr>
<td>23.0 23.</td>
<td>23 23</td>
</tr>
<tr>
<td>2.3E1 2.3</td>
<td>23 2.3</td>
</tr>
<tr>
<td>-23 0</td>
<td>-23 0</td>
</tr>
</tbody>
</table>

YENw.d

Removes embedded yen signs, commas, and decimal points
Category: Numeric

Syntax
YENw.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 YENw.d reads a decimal point and two decimal digits. If d is 0, YENw.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;

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>¥1254.71</td>
<td>1254.71</td>
</tr>
</tbody>
</table>

YYMMDDw.

Reads date values in the form yymmd or yyyyymmdd

Category: Date and time
Syntax Description

w

specifies the width of the input field.

Default: 6
Range: 6–32

Details

The date values must be in the form yymmd or yyyyymmd, 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 yymmd10.;

<table>
<thead>
<tr>
<th>Data Lines</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>970316</td>
<td>13589</td>
</tr>
<tr>
<td>97/03/16</td>
<td>13589</td>
</tr>
</tbody>
</table>
**YYMMN\textit{w}.

Reads date values in the form \textit{yyyymm} or \textit{yymm}

Category: Date and time

**Syntax**

\texttt{YYMMN\textit{w}.}

**Syntax Description**

\textit{\textit{w}}

specifies the width of the input field.

- **Default:** 4
- **Range:** 4–6

**Details**

The date values must be in the form \textit{yyyymm} or \textit{yymm}, where

---

**See Also**

Formats:

- “DATE\textit{w}.” on page 83
- “DDMMYY\textit{w}.” on page 88
- “MMDDYY\textit{w}.” on page 125
- “YYMMDD\textit{w}.” on page 182

Functions:

- “DAY” on page 308
- “MDY” on page 420
- “MONTH\textit{w}.” on page 131
- “YEAR” on page 592

Informats:

- “DATE\textit{w}.” on page 633
- “DDMMYY\textit{w}.” on page 636
- “MMDDYY\textit{w}.” on page 653

System Option:

- “YEARCUTOFF =” on page 1116
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 $\text{YEARCUTOFF}=$ system option.

**Examples**

```
input date1 yyymm6.;
```

---

**Data Lines**

| 199808 | 01AUG1998 |

---

**See Also**

Functions:

- “DAY” on page 308
- “MONTH” on page 428
- “YEAR” on page 592
- “MDY” on page 420

Informats:

- “DATEw.” on page 633
- “DDMMYYw.” on page 636
- “MMDDYYw.” on page 653
- “YYMMDDw.” on page 701

System Option:

- “YEARCUTOFF=” on page 1116

Formats:

- “DATEw.” on page 83
- “DDMMYYw.” on page 88
- “YYMDDw.” on page 182
- “YYMMxw.” on page 181
- “YYMONEw.” on page 186

**YYQw.**

Reads quarters of the year
Category: Date and time

Syntax
YYQw.

Syntax Description

w
specifies the width of the input field.

Default: 4
Range: 4–32

Details
The quarter must be in the form yyQq or yyyyQq, 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 YYQw. 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 yyq7.;

Data Lines Results

| ----+----1----+ | 13605 |
| 97Q2 | 13605 |
Data Lines | Results
---|---
97Q02 | 13605
1997Q02 | 13605

See Also

Functions:
- “QTR” on page 485
- “YEAR” on page 592
- “YYQ” on page 594

System Option:
- “YEARCUTOFF=” on page 1116

**ZDw.d**

Reads zoned decimal data

Category: Numeric

**Syntax**

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

<table>
<thead>
<tr>
<th>Language</th>
<th>Zoned Decimal Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>ZD3.</td>
</tr>
<tr>
<td>PL/I</td>
<td>PICTURE’99T’</td>
</tr>
<tr>
<td>COBOL</td>
<td>DISPLAY PIC S 999</td>
</tr>
<tr>
<td>IBM 370 assembler</td>
<td>ZL3</td>
</tr>
</tbody>
</table>

Examples

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

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>00F1F2C8</td>
<td>128</td>
</tr>
</tbody>
</table>

* 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 699
- “ZDVw.d” on page 708

ZDBw.d

Reads zoned decimal data in which zeros have been left blank
Category: Numeric

Syntax

ZDBw.d

Syntax Description
specifies the width of the input field.

**Default:** 1

**Range:** 1–32

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

**Range:** 0–31

**Details**

The ZDBw.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**

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

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>F140C2</td>
<td>102</td>
</tr>
</tbody>
</table>

* 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.

**ZDVw.d**

Reads and validates zoned decimal data

**Category:** Numeric

**Syntax**

```
ZDVw.d
```

**Syntax Description**

specifies the width of the input field.

**Default:** 1

**Range:** 1–32

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

Details
The ZDVw.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.

ZDVw.d is dependent on the operating environment. For example, on IBM mainframes, ZDVw.d requires an F for all high-order nibbles except the last. (In contrast, the ZDw.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 ZDVw.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 ZDVw.d informat functions like the ZDw.d informat with one exception: ZDVw.d validates the input string and disallows invalid data.

Examples

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

<table>
<thead>
<tr>
<th>Data Lines*</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0F1F2C8</td>
<td>128</td>
</tr>
</tbody>
</table>

* 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 699
“ZDw.d” on page 706