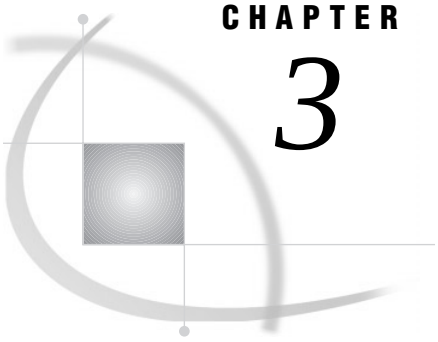

CHAPTER

3

Formats

Definition **51**
Syntax **52**
Using Formats **52**
Ways to Specify Formats **52**
PUT Statement **53**
PUT Function **53**
%SYSFUNC **53**
FORMAT Statement **53**
ATTRIB Statement **54**
Permanent versus Temporary Association **54**
User-Defined Formats **54**
Byte Ordering on Big Endian and Little Endian Platforms **55**
Definitions **55**
How Bytes are Ordered Differently **55**
Writing Data Generated on Big Endian or Little Endian Platforms **55**
Integer Binary Notation and Different Programming Languages **56**
Working with Packed Decimal and Zoned Decimal Data **57**
Definitions **57**
Types of Data **57**
Packed Decimal Data **57**
Zoned Decimal Data **58**
Packed Julian Dates **58**
Platforms Supporting Packed Decimal and Zoned Decimal Data **58**
Languages Supporting Packed Decimal and Zoned Decimal Data **58**
Summary of Packed Decimal and Zoned Decimal Formats and Informats **59**
International Date and Datetime Formats **60**
Formats by Category **66**
Dictionary **71**
$ASCIIw. **71**
$BINARYw. **72**
$CHARw. **72**
$EBCDICw. **73**
$HEXw. **74**
$KANJIw. **75**
$KANJIXw. **76**
$MSGCASEw. **76**
$OCTALw. **77**
$QUOTEw. **78**
$REVERJw. **79**
$REVERSw. **79**
$UPCASEw. **80**

$VARYINGw. **81**
$w. **83**
BESTw. **83**
BINARYw. **84**
COMMAw.d **85**
COMMAXw.d **86**
Dw.s **87**
DATEw. **89**
DATEAMPMw.d **90**
DATETIMEw.d **91**
DAYw. **93**
DDMMYYw. **94**
DDMMYYxw. **95**
DOLLARw.d **97**
DOLLARXw.d **98**
DOWNAMEw. **99**
Ew. **100**
EURDFDDw. **101**
EURDFDEw. **103**
EURDFDNw. **105**
EURDFDTw.d **106**
EURDFDWNw. **108**
EURDFMNw. **110**
EURDFMYw. **111**
EURDFWDXw. **113**
EURDFWKXw. **115**
FLOATw.d **118**
FRACTw. **119**
HEXw. **120**
HHMMw.d **121**
HOURw.d **123**
IBw.d **124**
IBRw.d **125**
IEEEw.d **127**
JULDAYw. **128**
JULIANw. **129**
MINGUOw. **130**
MMDDYYw. **131**
MMDDYYxw. **132**
MMSSw.d **134**
MMYYxw. **135**
MONNAMEw. **136**
MONTHw. **137**
MONYYw. **138**
NEGPARENw.d **139**
NENGOw. **140**
NUMXw.d **141**
OCTALw. **143**
PDw.d **144**
PDJULGw. **145**
PDJULIw. **146**
PERCENTw.d **148**
PIBw.d **149**
PIBRw.d **150**

PKw.d **152**
PVALUEw.d **153**
QTRw. **154**
QTRRw. **155**
RBw.d **155**
ROMANw. **157**
SSNw. **157**
S370FFw.d **158**
S370FIBw.d **159**
S370FIBUw.d **161**
S370FPDw.d **162**
S370FPDUw.d **163**
S370FPIBw.d **164**
S370FRBw.d **166**
S370FZDw.d **167**
S370FZDLw.d **168**
S370FZDSw.d **169**
S370FZDTw.d **170**
S370FZDUw.d **171**
TIMEw.d **172**
TIMEAMPMw.d **174**
TODw.d **175**
w.d **176**
WEEKDATEw. **177**
WEEKDATXw. **179**
WEEKDAYw. **180**
WORDDATEw. **181**
WORDDATXw. **182**
WORDFw. **183**
WORDSw. **184**
YEARw. **185**
YENw.d **186**
YYMMxw. **187**
YYMMDDw. **188**
YYMMDDxw. **190**
YYMONw. **192**
YYQxw. **193**
YYQRxw. **194**
Zw.d **195**
ZDw.d **196**

Definition

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

Syntax

SAS formats have the following form:

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

where

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

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

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

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

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

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

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

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

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

Using Formats

Ways to Specify Formats

You can use formats in the following ways:

- in a PUT statement
- with the PUT, PUTC, or PUTN functions

- with the %SYSFUNC macro function
- in a FORMAT statement in a DATA step or a PROC step
- in an ATTRIB statement in a DATA step or a PROC step.

PUT Statement

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

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

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

```
$1,145.32
```

See “PUT” on page 962 for more information.

PUT Function

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

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

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

The PUT function is useful for converting a numeric value to a character value. See “PUT” on page 503 for more information.

%SYSFUNC

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

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

%tst (1154.23);
```

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

FORMAT Statement

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

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

Because the FORMAT statement permanently associates a format with a variable, any subsequent DATA step or PROC step uses COMMA10.2 to write the values of SALES1, SALES2, and SALES3. See “FORMAT” on page 843 for more information.

Note: Formats that you specify in a PUT statement behave differently from those that you associate with a variable in a FORMAT statement. The major difference is that formats that are specified in the PUT statement preserve leading blanks. If you assign formats with a FORMAT statement prior to a PUT statement, all leading blanks are trimmed. The result is the same as if you used the colon (:) format modifier. For details about using the colon (:) format modifier, see "PUT, List" on page 983. △

ATTRIB Statement

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

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

Because the ATTRIB statement permanently associates a format with a variable, any subsequent DATA step or PROC step uses COMMA10.2 to write the values of SALES1, SALES2, and SALES3. See "ATTRIB" on page 762 for more information.

Permanent versus Temporary Association

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

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

User-Defined Formats

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

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

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

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

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

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

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

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

Byte Ordering on Big Endian and Little Endian Platforms

Definitions

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

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

How Bytes are Ordered Differently

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

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

Writing Data Generated on Big Endian or Little Endian Platforms

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

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

Table 3.1 SAS Formats and Byte Ordering

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

Integer Binary Notation and Different Programming Languages

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

Table 3.2 Integer Binary Notation and Programming Languages

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

Language	2 Bytes	4 Bytes
IBM assembler	H	F
C	short	long

Working with Packed Decimal and Zoned Decimal Data

Definitions

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

Types of Data

Packed Decimal Data

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

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

The following applies to packed decimal data representation:

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

Zoned Decimal Data

The following applies to zoned decimal data representation:

- A zoned decimal representation stores a decimal digit in the low order nibble of each byte. For all but the byte containing the sign, the high-order nibble is the numeric zone nibble (F on EBCDIC and 3 on ASCII).
- The sign can be merged into a byte with a digit, or it can be separate, depending on the representation. But the standard zoned decimal format and informat expects the sign to be merged into the last byte.
- The EBCDIC and ASCII zoned decimal formats produce the same printable representation of numbers. There are two nibbles per byte, each indicated by a hexadecimal digit. For example, the value 15 is stored in two bytes. The first byte contains the hexadecimal value F1 and the second byte contains the hexadecimal value C5.

Packed Julian Dates

The following applies to packed Julian dates:

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

Platforms Supporting Packed Decimal and Zoned Decimal Data

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

Languages Supporting Packed Decimal and Zoned Decimal Data

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

IBM VS COBOL II clauses	Corresponding S370Fxxx formats/informats
PIC S9(X) PACKED-DECIMAL	S370FPDw.
PIC 9(X) PACKED-DECIMAL	S370FPDUw.
PIC S9(W) DISPLAY	S370ZDw.
PIC 9(W) DISPLAY	S370ZDUw.
PIC S9(W) DISPLAY SIGN LEADING	S370FZDLw.

IBM VS COBOL II clauses	Corresponding S370Fxxx formats/informats
PIC S9(W) DISPLAY SIGN LEADING SEPARATE	S370FZDSw.
PIC S9(W) DISPLAY SIGN TRAILING SEPARATE	S370FZDTw.

For the packed decimal representation listed above, X indicates the number of digits represented, and W is the number of bytes. For PIC S9(X) PACKED-DECIMAL, W is **ceil((x+1)/2)**. For PIC 9(X) PACKED-DECIMAL, W is **ceil(x/2)**. For example, PIC S9(5) PACKED-DECIMAL represents five digits. If a sign is included, six nibbles are needed. **ceil((5+1)/2)** has a length of three bytes, and the value of W is 3.

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

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

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

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

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

Summary of Packed Decimal and Zoned Decimal Formats and Informats

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

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

Format	Type of data representation	Corresponding informat	Comments
S370FPDU	Packed decimal	S370FPDU	Last nibble always F (positive)
S370FZD	Zoned decimal	S370FZD	Last byte contains sign in upper nibble: C (positive) or D (negative)
S370FZDU	Zoned decimal	S370FZDU	Unsigned; sign nibble always F
S370FZDL	Zoned decimal	S370FZDL	Sign nibble in first byte in informat; separate leading sign byte of hex C0 (positive) or D0 (negative) in format
S370FZDS	Zoned decimal	S370FZDS	Leading sign of - (hex 60) or + (hex 4E)
S370FZDT	Zoned decimal	S370FZDT	Trailing sign of - (hex 60) or + (hex 4E)
PDJULI	Packed decimal	PDJULI	Julian date in packed representation - IBM computation
PDJULG	Packed decimal	PDJULG	Julian date in packed representation - Gregorian computation
none	Packed decimal	RMFDUR	Input layout is: *mmssttt*F
none	Packed decimal	SHRSTAMP	Input layout is: *yyyyddd*F*hhmmssth*, where *yyyyddd*F is the packed Julian date; *yyyy* is a 0-based year from 1900
none	Packed decimal	SMFSTAMP	Input layout is: *xxxxxxxxyyyyddd*F, where *yyyyddd*F is the packed Julian date; *yyyy* is a 0-based year from 1900
none	Packed decimal	PDTIME	Input layout is: 0*hhmmss*F
none	Packed decimal	RMFSTAMP	Input layout is: 0*hhmmss*F*yyyyddd*F, where *yyyyddd*F is the packed Julian date; yyyy is a 0-based year from 1900

International Date and Datetime Formats

The SAS supports international formats that are equivalent to some of the most commonly used English-language date formats. In each case the format works like the corresponding English-language format. Only the maximum, minimum, and default width are different.

Table 3.3 International Date and Datetime Formats

Language	English Format	International Format	Min	Max	Default
Afrikaans (AFR)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	9
	MONNAME.	EURDFMN.	1	32	9
	MONYY.	EURDFMY.	5	7	5
	WEEKDATX.	EURDFWK.	2	38	28
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFDE.	3	37	29
Catalan (CAT)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	9
	MONNAME.	EURDFMN.	1	32	8
	MONYY.	EURDFMY.	5	32	5
	WEEKDATX.	EURDFWKX.	2	40	27
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EUDFWDX.	3	40	16
Croatian (CRO)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	10
	MONNAME.	EURDFMN.	1	32	8
	MONYY.	EURDFMY.	5	32	5
	WEEKDATX.	EURDFWKX.	3	40	27
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	40	16
Czech (CSY)	DATE.	EURDFDE.	10	14	12
	DATETIME.	EURDFDT.	12	40	21
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFWN.	1	32	7
	MONNAME.	EURDFMN.	1	32	8
	MONYY.	EURDFMY.	10	32	10
	WEEKDATX.	EURDFWKX.	2	40	25

Language	English Format	International Format	Min	Max	Default
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	8	40	16
Danish (DAN)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	7
	MONNAME.	EURDFMN.	1	32	9
	MONYY.	EURDFMY.	5	7	5
	WEEKDATX.	EURDFWKX.	2	31	31
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	18	18
Dutch (NLD)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	9
	MONNAME.	EURDFMN.	1	32	9
	MONYY.	EURDFMY.	5	7	5
	WEEKDATX.	EURDFWKX.	2	38	28
	WORDDATX.	EURDFWDX.	3	37	29
	WEEKDAY.	EURDFDN.	1	32	1
Finnish (FIN)	DATE.	EURDFDE.	9	10	9
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	10
	DOWNAME.	EURDFDWN.	1	32	11
	MONNAME.	EURDFMN.	1	32	11
	MONYY.	EURDFMY.	8	8	8
	WEEKDATX.	EURDFWKX.	2	37	37
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	20	20
French (FRA)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	8
	MONNAME.	EURDFMN.	1	32	9
	MONYY.	EURDFMY.	5	7	5
	WEEKDATX.	EURDFWKX.	3	27	27
	WEEKDAY.	EURDFDN.	1	32	1

Language	English Format	International Format	Min	Max	Default
	WORDDATX.	EURDFWDX.	3	18	18
German (DEU)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	10
	MONNAME.	EURDFMN.	1	32	9
	MONYY.	EURDFMY.	5	7	5
	WEEKDATX.	EURDFWKX.	3	30	30
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	18	18
Hungarian (HUN)	DATE.	EURDFDE.	8	12	10
	DATETIME.	EURDFDT.	0	40	19
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	9
	MONNAME.	EURDFMN.	1	32	10
	MONYY.	EURDFMY.	8	32	8
	WEEKDATX.	EURDFWKX.	3	40	28
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	6	40	18
Italian (ITA)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	9
	MONNAME.	EURDFMN.	1	32	9
	MONYY.	EURDFMY.	5	7	5
	WEEKDATX.	EURDFWKX.	3	28	28
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	17	17
Macedonian (MAC)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	10
	MONNAME.	EURDFMN.	1	32	9
	MONYY.	EURDFMY.	5	32	5
	WEEKDATX.	EURDFWKX.	3	40	29
	WEEKDDATX.	EURDFWDX.	1	32	1
	WORDDATX.	EURDFDN.	3	40	17

Language	English Format	International Format	Min	Max	Default
Norwegian (NOR)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	7
	MONNAME.	EURDFMN.	1	32	9
	MONYY.	EURDFMY.	5	7	5
	WEEKDATX.	EURDFWKX.	3	26	26
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	17	17
Polish (POL)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	12
	MONNAME.	EURDFMN.	1	32	12
	MONYY.	EURDFMY.	5	32	5
	WEEKDATX.	EURDFWKX.	2	40	34
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	40	17
Portuguese (PTG)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	13
	MONNAME.	EURDFMN.	1	32	9
	MONYY.	EURDFMY.	5	7	5
	WEEKDATX.	EURDFWKX.	3	38	38
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	37	23
Russian (RUS)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	11
	MONNAME.	EURDFMN.	1	32	8
	MONYY.	EURDFMY.	5	32	5
	WEEKDATX.	EURDFWKX.	2	40	29
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	40	16
Spanish (ESP)	DATE.	EURDFDE.	5	9	7

Language	English Format	International Format	Min	Max	Default
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	9
	MONNAME.	EURDFMN.	1	32	10
	MONYY.	EURDFMY.	5	7	5
	WEEKDATX.	EURDFWKX.	1	35	35
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	24	24
Slovenian (SLO)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	10
	MONNAME.	EURDFMN.	1	32	9
	MONYY.	EURDFMY.	5	32	5
	WEEKDATX.	EURDFWKX.	3	40	29
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	40	17
Swedish (SVE)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	7
	MONNAME.	EURDFMN.	1	32	9
	MONYY.	EURDFMY.	5	7	5
	WEEKDATX.	EURDFWKX.	3	26	26
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	17	17
Swiss_French (FRS)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	8
	MONNAME.	EURDFMN.	1	32	9
	MONYY.	EURDFMY.	5	7	5
	WEEKDATX.	EURDFWKX.	3	26	26
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	17	17
Swiss_German (DES)	DATE.	EURDFDE.	5	9	7
	DATETIME.	EURDFDT.	7	40	16

Language	English Format	International Format	Min	Max	Default
	DDMMYY.	EURDFDD.	2	10	8
	DOWNAME.	EURDFDWN.	1	32	10
	MONNAME.	EURDFMN.	1	32	9
	MONYY.	EURDFMY.	5	7	5
	WEEKDATX.	EURDFWKX.	3	30	30
	WEEKDAY.	EURDFDN.	1	32	1
	WORDDATX.	EURDFWDX.	3	18	18

Formats by Category

There are four categories of formats in SAS:

Category	Description
CHARACTER	instructs SAS to write character data values from character variables.
DATE and TIME	instructs SAS to write data values from variables that represent dates, times, and datetimes.
DBCS	instructs SAS to handle various Asian languages
NUMERIC	instructs SAS to write numeric data values from numeric variables.
USER-DEFINED	instructs SAS to write data values by using a format that is created with PROC FORMAT.

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

The following table provides brief descriptions of the SAS formats. For more detailed descriptions, see the dictionary entry for each format.

Table 3.4 Categories and Descriptions of Formats

Category	Format	Description
Character	"$ASCII*w*." on page 71	Converts native format character data to ASCII representation
	"$BINARY*w*." on page 72	Converts character data to binary representation
	"$CHAR*w*." on page 72	Writes standard character data
	"$EBCDIC*w*." on page 73	Converts native format character data to EBCDIC representation
	"$HEX*w*." on page 74	Converts character data to hexadecimal representation
	"$MSGCASE*w*." on page 76	Writes character data in uppercase when the MSGCASE system option is in effect
	"$OCTAL*w*." on page 77	Converts character data to octal representation

	“$QUOTE*w*.” on page 78	Writes data values that are enclosed in double quotation marks
	“$REVERJ*w*.” on page 79	Writes character data in reverse order and preserves blanks
	“$REVERS*w*.” on page 79	Writes character data in reverse order and left aligns
	“$UPCASE*w*.” on page 80	Converts character data to uppercase
	“$VARYING*w*.” on page 81	Writes character data of varying length
	“$*w*.” on page 83	Writes standard character data
DBCS	“$KANJI*w*.” on page 75	Adds shift-code data to DBCS data
	“$KANJIX*w*.” on page 76	Removes shift code data from DBCS data
Date and Time	“DATE*w*.” on page 89	Writes date values in the form *ddmmmyy* or *ddmmmyyyy*
	“DATEAMPM*w*.*d* ”on page 90	Writes datetime values in the form *ddmmmyy:hh:mm:ss.ss* with AM or PM
	“DATETIME*w*.*d*” on page 91	Writes datetime values in the form *ddmmmyy:hh:mm:ss.ss*
	“DAY*w*.” on page 93	Writes date values as the day of the month
	“DDMMYY*w*.” on page 94	Writes date values in the form *ddmmyy* or *ddmmyyyy*
	“DDMMYY*xw*.” on page 95	Writes date values in the form *ddmmyy* or *ddmmyyyy* with a specified separator
	“DOWNAME*w*.” on page 99	Writes date values as the name of the day of the week
	“EURDFDD*w*.” on page 101	Writes international date values in the form *dd.mm.yy* or *dd.mm.yyyy*
	“EURDFDE*w*.” on page 103	Writes international date values in the form *ddmmmyy* or *ddmmmyyyy*
	“EURDFDN*w*.” on page 105	Writes international date values as the day of the week
	“EURDFDT*w*.*d*” on page 106	Writes international datetime values in the form *ddmmmyy:hh:mm:ss.ss* or *ddmmmyyyy hh:mm:ss.ss*
	“EURDFDWN*w*.” on page 108	Writes international date values as the name of the day
	“EURDFMN*w*.” on page 110	Writes international date values as the name of the month
	“EURDFMY*w*.” on page 111	Writes international date values in the form *mmmyy* or *mmmyyyy*
	“EURDFWDX*w*.” on page 113	Writes international date values as the name of the month, the day, and the year in the form *dd month-name yy* (or *yyyy*)
	“EURDFWKX*w*.” on page 115	Writes international date values as the name of the day and date in the form *day-of-week, dd month-name yy* (or *yyyy*)
	“HHMM*w*.*d*” on page 121	Writes time values as hours and minutes in the form *hh:mm*

"HOUR*w.d*" on page 123	Writes time values as hours and decimal fractions of hours
"JULDAY*w*." on page 128	Writes date values as the Julian day of the year
"JULIAN*w*." on page 129	Writes date values as Julian dates in the form *yyddd* or *yyyyddd*
"MINGUO*w*." on page 130	Writes date values as Taiwanese dates in the form *yyymmdd*
"MMDDYY*w*." on page 131	Writes date values in the form *mmddyy* or *mmddyyyy*
"MMDDYY*xw*." on page 132	Writes date values in the form *mmddyy* or *mmddyyyy* with a specified separator
"MMSS*w.d*" on page 134	Writes time values as the number of minutes and seconds since midnight
"MMYY*xw*." on page 135	Writes date values as the month and the year and separates them with a character
"MONNAME*w*." on page 136	Writes date values as the name of the month
"MONTH*w*." on page 137	Writes date values as the month of the year
"MONYY*w*." on page 138	Writes date values as the month and the year in the form *mmmyy* or *mmmyyyy*
"NENGO*w*." on page 140	Writes date values as Japanese dates in the form *e.yymmdd*
"PDJULG*w*." on page 145	Writes packed Julian date values in the hexadecimal format *yyyyddd*F for IBM
"PDJULI*w*." on page 146	Writes packed Julian date values in the hexadecimal format *ccyyddd*F for IBM
"QTR*w*." on page 154	Writes date values as the quarter of the year
"QTRR*w*." on page 155	Writes date values as the quarter of the year in Roman numerals
"TIME*w.d*" on page 172	Writes time values as hours, minutes, and seconds in the form *hh:mm:ss.ss*
"TIMEAMPM*w.d* "on page 174	Writes time values as hours, minutes, and seconds in the form *hh:mm:ss.ss* with AM or PM
"TOD*w.d*" on page 175	Writes the time portion of datetime values in the form *hh:mm:ss.ss*
"WEEKDATE*w*." on page 177	Writes date values as the day of the week and the date in the form *day-of-week, month-name dd, yy* (or *yyyy*)
"WEEKDATX*w*." on page 179	Writes date values as day of week and date in the form *day-of-week, dd month-name yy* (or *yyyy*)
"WEEKDAY*w*." on page 180	Writes date values as the day of the week
"WORDDATE*w*. "on page 181	Writes date values as the name of the month, the day, and the year in the form *month-name dd, yyyy*
"WORDDATX*w*." on page 182	Writes date values as the day, the name of the month, and the year in the form *dd month-name yyyy*

	“YEAR*w*.” on page 185	Writes date values as the year
	“YYMM*xw*.” on page 187	Writes date values as the year and month and separates them with a character
	“YYMMDD*w*.” on page 188	Writes date values in the form *yymmdd* or *yyyymmdd*
	“YYMMDD*xw*.” on page 190	Writes date values in the form *yymmdd* or *yyyymmdd* with a specified separator
	“YYMON*w*.” on page 192	Writes date values as the year and the month abbreviation
	“YYQ*xw*.” on page 193	Writes date values as the year and the quarter and separates them with a character
	“YYQR*xw*.” on page 194	Writes date values as the year and the quarter in Roman numerals and separates them with characters
Numeric	“BEST*w*.” on page 83	SAS chooses the best notation
	“BINARY*w*.” on page 84	Converts numeric values to binary representation
	“COMMA*w*.*d* ”on page 85	Writes numeric values with commas and decimal points
	“COMMAX*w*.*d*” on page 86	Writes numeric values with periods and commas
	“D*w*.*s*” on page 87	Prints variables, possibly with a great range of values, lining up decimal places for values of similar magnitude
	“DOLLAR*w*.*d*” on page 97	Writes numeric values with dollar signs, commas, and decimal points
	“DOLLARX*w*.*d*” on page 98	Writes numeric values with dollar signs, periods, and commas
	“E*w*.” on page 100	Writes numeric values in scientific notation
	“FLOAT*w*.*d*” on page 118	Generates a native single-precision, floating-point value by multiplying a number by 10 raised to the *d*th power
	“FRACT*w*.” on page 119	Converts numeric values to fractions
	“HEX*w*.” on page 120	Converts real binary (floating-point) values to hexadecimal representation
	“IB*w*.*d*” on page 124	Writes native integer binary (fixed-point) values, including negative values
	“IBR*w*.*d*” on page 125	Writes integer binary (fixed-point) values in Intel and DEC formats
	“IEEE*w*.*d*” on page 127	Generates an IEEE floating-point value by multiplying a number by 10 raised to the *d*th power
	“NEGPAREN*w*.*d*” on page 139	Writes negative numeric values in parentheses
	“NUMX*w*.*d*” on page 141	Writes numeric values with a comma in place of the decimal point
	“OCTAL*w*.” on page 143	Converts numeric values to octal representation
	“PD*w*.*d*” on page 144	Writes data in packed decimal format
	“PERCENT*w*.*d*” on page 148	Writes numeric values as percentages
	“PIB*w*.*d*” on page 149	Writes positive integer binary (fixed-point) values

“PIBR*w*.*d*” on page 150	Writes positive integer binary (fixed-point) values in Intel and DEC formats
“PK*w*.*d*” on page 152	Writes data in unsigned packed decimal format
“PVALUE*w*.*d*” on page 153	Writes *p*-values
“RB*w*.*d*” on page 155	Writes real binary data (floating-point) in real binary format
“ROMAN*w*.” on page 157	Writes numeric values as Roman numerals
“SSN*w*.” on page 157	Writes Social Security numbers
“S370FF*w*.*d*” on page 158	Writes native standard numeric data in IBM mainframe format
“S370FIB*w*.*d*” on page 159	Writes integer binary (fixed-point) values, including negative values, in IBM mainframe format
“S370FIBU*w*.*d*” on page 161	Writes unsigned integer binary (fixed-point) values in IBM mainframe format
“S370FPD*w*.*d*” on page 162	Writes packed decimal data in IBM mainframe format
“S370FPDU*w*.*d*” on page 163	Writes unsigned packed decimal data in IBM mainframe format
“S370FPIB*w*.*d*” on page 164	Writes positive integer binary (fixed-point) values in IBM mainframe format
“S370FRB*w*.*d*” on page 166	Writes real binary (floating-point) data in IBM mainframe format
“S370FZD*w*.*d*” on page 167	Writes zoned decimal data in IBM mainframe format
“S370FZDL*w*.*d*” on page 168	Writes zoned decimal leading sign data in IBM mainframe format
“S370FZDS*w*.*d*” on page 169	Writes zoned decimal separate leading-sign data in IBM mainframe format
“S370FZDT*w*.*d*” on page 170	Writes zoned decimal separate trailing-sign data in IBM mainframe format
“S370FZDU*w*.*d*” on page 171	Writes unsigned zoned decimal data in IBM mainframe format
“ *w*.*d*” on page 176	Writes standard numeric data one digit per byte
“WORDF*w*.” on page 183	Writes numeric values as words with fractions that are shown numerically
“WORDS*w*.” on page 184	Writes numeric values as words
“YEN*w*.*d*” on page 186	Writes numeric values with yen signs, commas, and decimal points

"Z*w.d*" on page 195	Writes standard numeric data with leading 0s
"ZD*w.d*" on page 196	Writes numeric data in zoned decimal format

Dictionary

$ASCII*w*.

Converts native format character data to ASCII representation

Category: Character
Alignment: left

Syntax

$ASCII*w*.

Syntax Description

w
specifies the width of the output field.
Default: 1
Range: 1–32767

Details

If ASCII is the native format, no conversion occurs.

Comparisons

- On EBCDIC systems, $ASCII*w*. converts EBCDIC character data to ASCII*w*.
- On all other systems, $ASCII*w*. behaves like the $CHAR*w*. format.

Examples

```
put x $ascii3.;
```

Values	Results*
abc	**616263**
ABC	**414243**
();	**28293B**

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

$BINARY*w*.

Converts character data to binary representation

Category: Character
Alignment: left

Syntax

$BINARY*w*.

Syntax Description

w
specifies the width of the output field.
Default: The default width is calculated based on the length of the variable to be printed.
Range: 1–32767

Comparisons

The $BINARY*w*. format converts character values to binary representation. The BINARY*w*. format converts numeric values to binary representation.

Examples

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

Values	Results	
	ASCII	EBCDIC
	`----+----1----+----2`	`----+----1----+----2`
AB	**0100000101000010**	**1100000111000010**

$CHAR*w*.

Writes standard character data

Category: Character
Alignment: left

Syntax

$CHAR*w*.

Syntax Description

w
specifies the width of the output field.

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

Range: 1–32767

Comparisons

- The $CHAR*w*. format is identical to the $*w*. format.
- The $CHAR*w*. and $*w*. formats do not trim leading blanks. To trim leading blanks, use the LEFT function to left align character data prior to output, or use the PUT statement with the colon (:) format modifier and the format of your choice to produce list output.
- Use the following table to compare the SAS format $CHAR8. with notation in other programming languages:

Language	Notation
SAS	$CHAR8.
C	char [8]
COBOL	PIC x(8)
FORTRAN	A8
PL/I	A(8)

Examples

```
put @7 name $char4.;
```

Values	Results
	`----+----1`
`XYZ`	`XYZ`

$EBCDIC*w*.

Converts native format character data to EBCDIC representation

Category: Character
Alignment: left

Syntax

$EBCDIC*w*.

Syntax Description

w
specifies the width of the output field.
Default: 1
Range: 1–32767

Details

If EBCDIC is the native format, no conversion occurs.

Comparisons

- On ASCII systems, $EBCDIC*w*. converts ASCII character data to EBCDIC.
- On all other systems, $EBCDIC*w*. behaves like the $CHAR*w*. format.

Examples

```
put name $ebcdic3.;
```

Values	Results*
qrs	**9899A2**
QRS	**D8D9E2**
+;>	**4E5E6E**

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

$HEX*w.*

Converts character data to hexadecimal representation

Category: Character
Alignment: left

Syntax

$HEX*w.*

Syntax Description

w
specifies the width of the output field.

Default: The default width is calculated based on the length of the variable to be printed.

Range: 1–32767

Tip: To ensure that SAS writes the full hexadecimal equivalent of your data, make *w* twice the length of the variable or field that you want to represent.

Tip: If *w* is greater than twice the length of the variable that you want to represent, $HEX*w*. pads it with blanks.

Details

The $HEX*w*. format converts each character into two hexadecimal digits. Each blank counts as one character, including trailing blanks.

Comparisons

The HEX*w*. format converts real binary numbers to their hexadecimal equivalent.

Examples

```
put @5 name $hex4.;
```

Values	Results	
	EBCDIC	ASCII
	`----+----1`	`----+----1`
AB	**C1C2**	**4142**

$KANJI*w*.

Adds shift-code data to DBCS data

Category: DBCS
Alignment: left

Syntax

$KANJI*w*.

Syntax Description

w
specifies the width of the output field.

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

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

$KANJIX*w.*

Removes shift code data from DBCS data

Category: DBCS

Alignment: left

Syntax

$KANJIX*w.*

Syntax Description

w

specifies the width of the output field.

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

Range: The minimum width of the format is 2.

Details

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

$MSGCASE*w.*

Writes character data in uppercase when the MSGCASE system option is in effect

Category: Character

Alignment: left

Syntax

$MSGCASE*w.*

Syntax Description

w
specifies the width of the output field.

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

Range: 1–32767

Details

When the MSGCASE= system option is in effect, all notes, warnings, and error messages that SAS generates appear in uppercase. Otherwise, all notes, warnings, and error messages appear in mixed case. You specify the MSGCASE= system option in the configuration file or during the SAS invocation.

Operating Environment Information: For more information about the MSGCASE= system option, see the SAS documentation for your operating environment. △

Examples

```
put name $msgcase.;
```

Values	Results
sas	**SAS**

$OCTAL*w*.

Converts character data to octal representation

Category: Character

Alignment: left

Syntax

$OCTAL*w*.

Syntax Description

w
specifies the width of the output field.

Default: The default width is calculated based on the length of the variable to be printed.

Range: 1–32767

Tip: Because each character value generates three octal characters, increase the value of *w* three times the length of the character value.

Comparisons

The $OCTAL*w*. format converts character values to the octal representation of their character codes. The OCTAL*w*. format converts numeric values to octal representation.

Examples

```
put @2 name $octal9.;
```

The character # represents a blank space.

Values	Results	
	EBCDIC	ASCII
	----+----1	----+----1
A##	301100100	101040040
B##	302100100	102040040

$QUOTE*w*.

Writes data values that are enclosed in double quotation marks

Category: Character
Alignment: left

Syntax

$QUOTE*w*.

Syntax Description

w
specifies the width of the output field.

Default: 8 if the length of the variable is undefined; otherwise, the length of the variable + 2

Range: 2–32767

Tip: Make *w* wide enough to include the left and right quotation marks.

Examples

```
put name $quote7.;
```

Values	Results
	`----+----1`
SAS	**"SAS"**
SAS's	**"SAS's"**

$REVERJ*w*.

Writes character data in reverse order and preserves blanks

Category: Character
Alignment: right

Syntax

$REVERJ*w*.

Syntax Description

w
specifies the width of the output field.
Default: 1 if the length of the variable is undefined; otherwise, the length of the variable
Range: 1–32767

Comparisons

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

Examples

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

Values*	Results
	`----+----1`
ABCD###	**DCBA**
###ABCD	**DCBA**

* The character # represents a blank space.

$REVERS*w*.

Writes character data in reverse order and left aligns

Category: Character
Alignment: left

Syntax

$REVERS*w*.

Syntax Description

w
specifies the width of the output field.
Default: 1
Range: 1–32767

Comparisons

The $REVERS*w*. format is similar to the $REVERJ*w*. format except that $REVERJ*w*. does not left align the result.

Examples

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

Values*	Results
	----+----1
ABCD###	**DCBA**
###ABCD	**DCBA**

* The character # represents a blank space.

$UPCASE*w*.

Converts character data to uppercase

Category: Character
Alignment: left

Syntax

$UPCASE*w*.

Syntax Description

w
specifies the width of the output 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 and other symbols, are not altered.

Examples

```
put @1 name $upcase9.;
```

Values	Results
	`----+----1`
`coxe-ryan`	`COXE-RYAN`

$VARYING*w*.

Writes character data of varying length

Valid: in DATA step
Category: Character
Alignment: left

Syntax

$VARYING*w. length-variable*

Syntax Description

w
specifies the maximum width of the output field for any output line or output file record.

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

Range: 1–32767

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

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

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

Tip: If the value of *length-variable* is 1 or missing, SAS writes nothing to the output field. If the value of *length-variable* is greater than 0 but less than *w*, SAS writes the number of characters that are specified by *length-variable*.

Details

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

Examples

Example 1: Obtaining a Variable Length Directly An existing data set variable contains the length of a variable. The data values and the results follow the explanation of this SAS statement:

```
put @10 name $varying12. varlen;
```

NAME is a character variable of length 12 that contains values that vary from 1 to 12 characters in length. VARLEN is a numeric variable in the same data set that contains the actual length of NAME for the current observation.

Values*	Results
	`----+----1----+----2----+`
New York 8	**New York**
Toronto 7	**Toronto**
Buenos Aires 12	**Buenos Aires**
Tokyo 5	**Tokyo**

* The value of NAME appears before the value of VARLEN.

Example 2: Obtaining a Variable Length Indirectly Use the LENGTH function to determine the length of a variable. The data values and the results follow the explanation of these SAS statements:

```
varlen=length(name);
put @10 name $varying12. varlen;
```

The assignment statement determines the length of the varying-length variable. The variable VARLEN contains this length and becomes the *length-variable* argument to the $VARYING12. format.

Values*	Results
	`----+----1----+----2----+`
New York	**New York**
Toronto	**Toronto**
Buenos Aires	**Buenos Aires**
Tokyo	**Tokyo**

* The value of NAME appears before the value of VARLEN.

$*w*.

Writes standard character data

Category: Character
Alignment left
Alias: $F*w*.

Syntax

$*w*.

Syntax Description

w
specifies the width of the output field. You can specify a number or a column range.

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

Range: 1–32767

Comparisons

The $*w*. format and the $CHAR*w*. format are identical, and they do not trim leading blanks. To trim leading blanks, use the LEFT function to left align character data prior to output, or use list output with the colon (:) format modifier and the format of your choice.

Examples

```
put @10 name $5.;
put name $ 10-15;
```

Values*	Results
	`----+----1----+----2`
`#Cary`	`Cary`
`Tokyo`	`Tokyo`

* The character # represents a blank space.

BEST*w*.

SAS chooses the best notation

Category: Numeric

Alignment: right

Syntax

BEST*w*.

Syntax Description

w
specifies the width of the output field.

Default: 12

Tip: If you print numbers between 0 and .01 exclusive, use a field width of at least 7 to avoid excessive rounding. If you print numbers between 0 and -.01 exclusive, use a field width of at least 8.

Range: 1–32

Details

The BEST*w*. format is the default format for writing numeric values. When there is no format specification, SAS chooses the format that provides the most information about the value according to the available field width. BEST*w*. rounds the value, and if SAS can display at least one significant digit in the decimal portion, within the width specified, BEST*w*. produces the result in decimal. Otherwise it produces the result in scientific notation. SAS always stores the complete value regardless of the format that you use to represent it.

Examples

```
put @1 x best6.;
put @1 x best3.;
```

Values	Results
	`----+----1----+----2`
`1257000`	`1.26E6`
	`1E6`

BINARY*w*.

Converts numeric values to binary representation

Category: Numeric

Alignment: left

Syntax

BINARY*w.*

Syntax Description

w
specifies the width of the output field.
Default: 8
Range: 1–64

Details

The BINARY*w*. format writes any negative numbers as all 1s.

Comparisons

BINARY*w*. converts numeric values to binary representation. The $BINARY*w*. format converts character values to binary representation.

Examples

```
put @1 x binary8.;
```

Values	Results
	`----+----1`
`123.45`	`01111011`
`123`	`01111011`
`-123`	`10000101`

COMMA*w.d*

Writes numeric values with commas and decimal points

Category: Numeric
Alignment: right

Syntax

COMMA*w.d*

Syntax Description

w
specifies the width of the output field.
Default: 6
Range: 2–32
Tip: Make *w* wide enough to write the numeric values, the commas, and the optional decimal point.

d
optionally specifies the number of digits to the right of the decimal point in the numeric value.
Range: 0–31
Requirement: must be less than *w*

Details

The COMMA*w.d* format writes numeric values with commas that separate every three digits and a period that separates the decimal fraction.

Comparisons

- The COMMA*w.d* format is similar to the COMMAX*w.d* format, but the COMMAX*w.d* format reverses the roles of the decimal point and the comma. This convention is common in European countries.
- The COMMA*w.d* format is similar to the DOLLAR*w.d* format except that the COMMA*w.d* format does not print a leading dollar sign.

Examples

```
put @10 sales comma10.2;
```

Values	Results
	`----+----1----+----2`
`23451.23`	`23,451.23`
`123451.234`	`123,451.23`

See Also

Formats:
"COMMAX*w.d*" on page 86
"DOLLAR*w.d*" on page 97

COMMAX*w.d*

Writes numeric values with periods and commas

Category: Numeric
Alignment: right

Syntax

COMMAX*w.d*

Syntax Description

w
specifies the width of the output field.
Default: 6
Range: 2–32
Tip: Make *w* wide enough to write the numeric values, the commas, and the optional decimal point.

d
optionally specifies the number of digits to the right of the decimal point in the numeric value.
Range: 0–31
Requirement: must be less than *w*

Details

The COMMAX*w.d* format writes numeric values with periods that separate every three digits and with a comma that separates the decimal fraction.

Comparisons

The COMMA*w.d* format is similar to the COMMAX*w.d* format, but the COMMAX*w.d* format reverses the roles of the decimal point and the comma. This convention is common in European countries.

Examples

```
put @10 sales commax10.2;
```

Values	Results
	----+----1----+----2
23451.23	23.451,23
123451.234	123.451,23

D*w.s*

Prints variables, possibly with a great range of values, lining up decimal places for values of similar magnitude

Category: Numeric
Alignment: right

Syntax

D*w.s*

Syntax Description

w
optionally specifies the width of the output field.

Default: 12

Range: 1–32

s
optionally specifies the significant digits.

Default: 3

Range; 0–16

Requirement: must be less than *w*

Details

The D*w.s* format writes numbers so that the decimal point aligns in groups of values with similar magnitude.

Comparisons

- The BEST*w*. format writes as many significant digits as possible in the output field, but if the numbers vary in magnitude, the decimal points do not line up.
- D*w.s* writes numbers with the desired precision and more alignment than BEST*w*.
- The *w.d* format aligns decimal points, if possible, but does not necessarily show the same precision for all numbers.

Examples

```
put @1 x d10.4;
```

Values	Results
	----+----1----+----2
12345	12345.0
1234.5	1234.5
123.45	123.45000
12.345	12.34500

Values	Results
`1.2345`	`1.23450`
`.12345`	`0.12345`

DATE*w*.

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

Category: Date and Time
Alignment: right

Syntax

DATE*w*.

Syntax Description

w
specifies the width of the output field.
Default: 7
Range: 5–9
Tip: Use a width of 9 to print a 4–digit year.

Details

The DATE*w*. format writes SAS date values in the form *ddmmmyy* or *ddmmmyyyy*, where

dd
is an integer 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.

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1----+`
`put day date5.;`	`16MAR`
`put day date6.;`	` 16MAR`
`put day date7.;`	`16MAR02`

SAS Statements	Results
`put day date8.;`	`16MAR02`
`put day date9.;`	`16MAR2002`

See Also

Function:

"DATE" on page 312

Informat:

"DATE*w*." on page 666

DATEAMPM*w.d*

Writes datetime values in the form *ddmmmyy:hh:mm:ss.ss* with AM or PM

Category: Date and Time

Alignment: right

Syntax

DATEAMPM*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 19

Range: 7–40

Tip: SAS requires a minimum *w* value of 13 to write AM or PM. For widths between 10 and 12, SAS writes a 24-hour clock time.

d

optionally specifies the number of digits to the right of the decimal point in the seconds value.

Requirement: must be less than *w*

Range: 0–39

Note: If *w*–*d*< 17, SAS truncates the decimal values. △

Details

The DATEAMPM*w.d* format writes SAS datetime values in the form *ddmmmyy:hh:mm:ss.ss*, where

dd

is an integer that represents the day of the month.

mmm
is the first three letters of the month name.

yy
is a two-digit integer that represents the year.

hh
is an integer that represents the hour.

mm
is an integer that represents the minutes.

ss.ss
is the number of seconds to two decimal places.

Comparisons

The DATEAMPM*w.d* format is similar to the DATETIMEM*w.d* format except that DATEAMPM*w.d* prints AM or PM at the end of the time.

Examples

The example table uses the input value of 1347453583, which is the SAS datetime value that corresponds to 12:39:43 PM on September 12, 2002.

SAS Statements	Results
	`----+----1----+----2----+`
`put event dateampm.;`	`12SEP02:12:39:43 PM`
`put event dateampm7.;`	`12SEP02`
`put event dateampm10.;`	`12SEP02:12`
`put event dateampm13.;`	`12SEP02:12 PM`
`put event dateampm22.2;`	`12SEP02:12:39:43.00 PM`

See Also

Format:

"DATETIME*w.d*" on page 91

DATETIME*w.d*

Writes datetime values in the form *ddmmmyy:hh:mm:ss.ss*

Category: Date and Time
Alignment: right

Syntax

DATETIME*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 16

Range: 7–40

Tip: SAS requires a minimum *w* value of 16 to write a SAS datetime value with the date, hour, and seconds. Add an additional two places to *w* to return values with optional decimal fractions of seconds.

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

Requirement: must be less than *w*

Range: 0–39

Note: If *w*–*d*< 17, SAS truncates the decimal values. △

Details

The DATETIME*w.d* format writes SAS datetime values in the form *ddmmmyy:hh:mm:ss.ss*, where

dd
is an integer that represents the day of the month.

mmm
is the first three letters of the month name.

yy
is a two-digit integer that represents the year.

hh
is an integer that represents the hour.

mm
is an integer that represents the minutes.

ss.ss
is the number of seconds to two decimal places.

Examples

The example table uses the input value of 1347453583, which is the SAS datetime value that corresponds to September 12, 2002, at 12:39:43 PM.

SAS Statements	Results
	`----+----1----+----2`
`put event datetime.;`	`12SEP02:12:39:43`
`put event datetime7.;`	`12SEP02`
`put event datetime12.;`	`12SEP02:12`
`put event datetime18.;`	`12SEP02:12:39:43`

SAS Statements	Results
`put event datetime18.1;`	`12SEP02:12:39:43.0`
`put event datetime19;`	`12SEP2002:12:39:43`
`put event datetime20.1;`	`12SEP2002:12:39:43.0`
`put event datetime21.2;`	`12SEP2002:12:39:43.00`

See Also

Formats:

"DATE*w*." on page 89

"TIME*w*.*d*" on page 172

Function:

"DATETIME" on page 315

Informats:

"DATE*w*." on page 666

"DATETIME*w*. "on page 667

"TIME*w*." on page 727

DAY*w*.

Writes date values as the day of the month

Category: Date and Time

Alignment: right

Syntax

DAY*w*.

Syntax Description

w

specifies the width of the output field.

Default: 2

Range: 2–32

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1`
`put date day2.;`	`16`

DDMMYY*w*.

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

Category: Date and Time

Alignment: right

Syntax

DDMMYY*w*.

Syntax Description

w

specifies the width of the output field.

Default: 8

Range: 2–10

Tip: When *w* is from 2 to 5, SAS prints as much of the month and day as possible. When *w* is 7, the date appears as a two-digit year without slashes, and the value is right aligned in the output field.

Details

The DDMMYY*w*. format writes SAS date values in the form *ddmmyy* or *ddmmyyyy*, where

dd
is an integer that represents the day of the month.

mm
is an integer that represents the month.

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

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1----+`
`put date ddmmyy5.;`	`16/03`
`put date ddmmyy6.;`	`160302`
`put date ddmmyy7.;`	` 160302`
`put date ddmmyy8.;`	`16/03/02`
`put date ddmmyy10.;`	`16/03/2002`

See Also

Formats:

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

Function:

“MDY” on page 443

Informats:

“DATE*w*.” on page 666
“DDMMYY*w*.” on page 669
“MMDDYY*w*.” on page 687
“YYMMDD*w*.” on page 733

DDMMYY*xw*.

Writes date values in the form *ddmmyy* or *ddmmyyyy* with a specified separator

Category: Date and Time
Alignment: right

Syntax

DDMMYY*xw*.

Syntax Description

x
specifies a separator or no separator, where

B
separates with a blank

C
separates with a colon

D
separates with a dash

N
indicates no separator

P
separates with a period

S
separates with a slash.

w
specifies the width of the output field.

Default: 8

Range: 2–10

Tip: When *w* is from 2 to 5, SAS prints as much of the month and day as possible. When *w* is 7, the date appears as a two-digit year without separators, and the value is right aligned in the output field.

Note: When *x* is N, the width range is 2–8. △

Details

The DDMMYY*xw*. format writes SAS date values in the form *ddmmyy* or *ddmmyyyy*, where

dd
is an integer that represents the day of the month.

mm
is an integer that represents the month.

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

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1----+`
`put date ddmmyyc5.;`	`16:03`
`put date ddmmyyd8.;`	`16-03-02`

SAS Statements	Results
`put date ddmmyyp10.;`	`16.03.2002`
`put date ddmmyyn8.;`	`16032002`

See Also

Formats:
- "DATE*w*." on page 89
- "MMDDYY*xw*." on page 132
- "YYMMDD*xw*." on page 190

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

Informat:
- "DDMMYY*w*." on page 669

DOLLAR*w.d*

Writes numeric values with dollar signs, commas, and decimal points

Category: Numeric
Alignment: right

Syntax

DOLLAR*w.d*

Syntax Description

w
specifies the width of the output field.
Default: 6
Range: 2–32

d
optionally specifies the number of digits to the right of the decimal point in the numeric value.
Range: 0–31
Requirement: must be less than *w*

Details

The DOLLAR*w.d* format writes numeric values with a leading dollar sign, with a comma that separates every three digits, and a period that separates the decimal fraction.

The hexadecimal representation of the code for the dollar sign character ($) is 5B on EBCDIC systems and 24 on ASCII systems. The monetary character that these codes represent may be different in other countries, but DOLLAR*w.d* always produces one of these codes. If you need another monetary character, define your own format with the FORMAT procedure. See "The FORMAT Procedure" in *SAS Procedures Guide* for more details.

Comparisons

- The DOLLAR*w.d* format is similar to the DOLLARX*w.d* format, but the DOLLARX*w.d* format reverses the roles of the decimal point and the comma. This convention is common in European countries.
- The DOLLAR*w.d* format is the same as the COMMA*w.d* format except that the COMMA*w.d* format does not write a leading dollar sign.

Examples

```
put @3 netpay dollar10.2;
```

Values	Results
	`----+----1----+`
`1254.71`	`$1,254.71`

See Also

Formats:

"COMMA*w.d* "on page 85
"DOLLARX*w.d*" on page 98

DOLLARX*w.d*

Writes numeric values with dollar signs, periods, and commas

Category: Numeric
Alignment: right

Syntax

DOLLARX*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 6

Range: 2–32

d

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

Default: 6

Range: 2–32

Details

The DOLLARX*w*.*d* format writes numeric values with a leading dollar sign, with a period that separates every three digits, and with a comma that separates the decimal fraction.

The hexadecimal representation of the code for the dollar sign character ($) is 5B on EBCDIC systems and 24 on ASCII systems. The monetary character that these codes represent may be different in other countries, but DOLLARX*w*.*d* always produces one of these codes. If you need another monetary character, define your own format with the FORMAT procedure. See "The FORMAT Procedure" in *SAS Procedures Guide* for more details.

Comparisons

- The DOLLARX*w*.*d* format is similar to the DOLLAR*w*.*d* format, but the DOLLARX*w*.*d* format reverses the roles of the decimal point and the comma. This convention is common in European countries.
- The DOLLARX*w*.*d* format is the same as the COMMAX*w*.*d* format except that the COMMA*w*.*d* format does not write a leading dollar sign.

Examples

```
put @3 netpay dollarx10.2;
```

Values	Results
	`----+----1----+`
`1254.71`	`$1.254,71`

See Also

Formats:

"COMMAX*w*.*d*" on page 86
"DOLLAR*w*.*d*" on page 97

DOWNAME*w*.

Writes date values as the name of the day of the week

Category: Date and Time
Alignment: right

Syntax

DOWNAME*w.*

Syntax Description

w
specifies the width of the output field.

Default: 9

Range: 1–32

Tip: If you omit *w*, SAS prints the entire name of the day.

Details

If necessary, SAS truncates the name of the day to fit the format width. For example, the DOWNAME2. prints the first two letters of the day name.

Examples

The example table uses the input value of 13589, which is the SAS date value that corresponds to March 16, 1997.

SAS Statements	Results
	`----+----1`
`put date downame.;`	`Sunday`

See Also

Format:

"WEEKDAY*w*." on page 180

E*w.*

Writes numeric values in scientific notation

Category: Numeric
Alignment: right

Syntax

E*w.*

Syntax Description

w
specifies the width of the output field.

Default: 12

Range: 7–32

Details

SAS reserves the first column of the result for a minus sign.

Examples

```
put @1 x e10.;
```

Values	Results
	`----+----1----+`
`1257`	`1.257E+03`
`-1257`	`-1.257E+03`

EURDFDD*w*.

Writes international date values in the form *dd.mm.yy* or *dd.mm.yyyy*

Category: Date and Time

Alignment: right

Syntax

EURDFDD*w*.

Syntax Description

w
specifies the width of the output field.

Default: 8 (except Finnish)

Range: 2–10

Tip: When *w* is from 2 to 5, SAS prints as much of the month and day as possible. When *w* is 7, the date appears as a two-digit year without slashes, and the value is right aligned in the output field.

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

Details

The EURDFDD*w*. format writes SAS date values in the form *dd.mm.yy* or *dd.mm.yyyy*, where

dd
is the two-digit integer that represents the day of the month.

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

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

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

Examples

The example table uses the input value 15342, which is the SAS date value that corresponds to January 2, 2002. The first PUT statement assumes that the DFLANG= system option is set to Spanish.

```
options dflang=spanish;
```

The second PUT statement uses the Spanish language prefix in the format to write the international date value. The third PUT statement uses the French language prefix in the format to write the international date value. The value of the DFLANG= option, therefore, is ignored.

SAS Statements	Results
	`----+----1`
`put date eurdfdd8.;`	`02.01.02`
`put date espdfdd8.;`	`02.01.02`
`put date fradfdd8.;`	`02/01/02`

See Also

Formats:

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

Function:

"MDY" on page 443

Informats:

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

System Option:

"DFLANG=" on page 1085

EURDFDE*w*.

Writes international date values in the form *ddmmmyy* or *ddmmmyyyy*

Category: Date and Time
Alignment: right

Syntax

EURDFDE*w*.

Syntax Description

w
specifies the width of the output field.

Default: 7 (except Finnish)

Range: 5–9 (except Finnish)

Note: If you use the Finnish (FIN) language prefix, the *w* range is 9–10 and the default is 9. △

Details

The EURDFDE*w*. format writes SAS date values in the form *ddmmmyy* or *ddmmmyyyy*, where

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

Examples

The example table uses the input value 15342, which is the SAS date value that corresponds to January 2, 2002. The first PUT statement assumes the DFLANG= system option is set to Spanish.

```
options dflang=spanish;
```

The second PUT statement uses the Spanish language prefix in the format to write the international date value in Spanish. The third PUT statement uses the French language prefix in the format to write the international date value in French. The value of the DFLANG= option, therefore, is ignored.

SAS Statements	Results
	`----+----1`
`put date eurdfde9.;`	`02ene2002`

SAS Statements	Results
`put date espdfde9.;`	`02ene2002`
`put date fradfde9.;`	`02jan2002`

See Also

Format:

"DATE*w*." on page 89

Function:

"DATE" on page 312

Informat:

"EURDFDE*w*. "on page 671

System Option:

"DFLANG=" on page 1085

EURDFDN*w*.

Writes international date values as the day of the week

Category: Date and Time
Alignment: right

Syntax

EURDFDN*w*.

Syntax Description

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

Details

The EURDFDN*w*. format writes SAS date values in the form *day-of-the-week*, where

day-of-the-week
is represented as 1=Monday, 2=Tuesday, and so on.

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

Examples

The example table uses the input value 15342, which is the SAS date value that corresponds to January 2, 2002. The first PUT statement assumes that the DFLANG= system option is set to Spanish.

```
options dflang=spanish;
```

The second PUT statement uses the Spanish language prefix in the format to write the day of the week in Spanish. The third PUT statement uses the Italian lanaguage prefix in the format to write the day of the week in Italian. The value of the DFLANG= option, therefore, is ignored.

SAS Statements	Results
	`----+----1`
`put day eurdfdn.;`	`3`
`put day espdfdn.;`	`3`
`put day itadfdn.;`	`3`

See Also

Formats:

"DOWNAME*w*." on page 99

"WEEKDAY*w*." on page 180

System Option:

"DFLANG=" on page 1085

EURDFDT*w.d*

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

Category: Date and Time

Alignment: right

Syntax

EURDFDT*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 16

Range: 7–40

Tip: If you want to write a SAS datetime value with the date, hour, and seconds, the width of *w* must be at least 16. Add an additional two places to the width if you want to return values with optional decimal fractions of seconds.

d

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

Range: 1–39

Restriction: must be less than *w*

Restriction: If $w - d < 17$, SAS truncates the decimal values.

Details

The EURDFDT*w*. format writes SAS datetime values in the form *ddmmmyy:hh:mm:ss.ss*, where

dd
is an integer 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.

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 that range from 00 through 59 with the fraction of a second following the decimal point.

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

Examples

The example table uses the input value of 1347453583, which is the SAS datetime value that corresponds to September 12, 2002, at 12:39:43 PM. The first PUT statement assumes the DFLANG= system option is set to German.

```
options dflang=german;
```

The second PUT statement uses the German language prefix in the format to write the international datetime value in German. The third PUT statement uses the Italian language prefix in the format to write the international datetime value in Italian. The value of the DFLANG= option, therefore, is ignored.

SAS Statements	Results
	`----+----1----+----2`
`put date eurdfdt20.;`	`12Sep2002:12:39:43`
`put date deudfdt20.;`	`12Sep2002:12:39:43`
`put date itadfdt20.;`	`12Set2002:12:39:43`

See Also

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

Function:
- "DATETIME" on page 315

Informats:
- "DATE*w*." on page 666
- "DATETIME*w*. "on page 667
- "EURDFDT*w*." on page 673
- "TIME*w*." on page 727

System Option:
- "DFLANG=" on page 1085

EURDFDWN*w*.

Writes international date values as the name of the day

Category: Date and Time
Alignment: right

Syntax

EURDFDWN*w*.

Syntax Description

w
specifies the width of the output field.

Default: depends on the language prefix you use. The following table shows the default for each language:

Language	Default
Afrikaans (AFR)	9
Catalan (CAT)	9
Croatian (CRO)	10

Language	Default
Czech (CSY)	7
Danish (DAN)	7
Dutch (NLD)	9
Finnish (FIN)	11
French (FRA)	8
German (DEU)	10
Hungarian (HUN)	9
Italian (ITA)	9
Macedonian (MAC)	10
Norwegian (NOR)	7
Polish (POL)	12
Portuguese (PTG)	13
Russian (RUS)	11
Slovenian (SLO)	10
Spanish (ESP)	9
Swedish (SVE)	7
Swiss-French (FRS)	8
Swiss-German (DES)	10

Range: 1–32

Tip: If you omit *w*, SAS prints the entire name of the day.

Details

If necessary, SAS truncates the name of the day to fit the format width. The EURDFDWN*w*. format writes SAS date values in the form *day-name*, where

day-name
is the name of the day.

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

Examples

The following example table uses the input value 15344, which is the SAS date value that corresponds to January 4, 2002. The first PUT statement assumes the DFLANG= system option is set to French.

```
options dflang=french;

put day eurdfdwn8.;
```

The second PUT statement uses the French language prefix in the format to write the day of the week in French. The third PUT statement uses the Spanish language prefix in the format to write the day of the week in Spanish. The value of the DFLANG= option, therefore, is ignored.

SAS Statements	Results
	`----+----1`
`put day eurdfdwn8.;`	`Vendredi`
`put day fradfdwn8.;`	`Vendredi`
`put day espdfdwn8.;`	`viernes`

See Also

Formats:

"DOWNAME*w*." on page 99
"WEEKDAY*w*." on page 180

Informats:

"DATE*w*." on page 666
"DATETIME*w*. "on page 667
"EURDFDT*w*." on page 673
"TIME*w*." on page 727

System Option:

"DFLANG=" on page 1085

EURDFMN*w*.

Writes international date values as the name of the month

Category: Date and Time
Alignment: right

Syntax

EURDFMN*w*.

Syntax Description

w
specifies the width of the output field.
Default: 9 (except for Finnish and Spanish)
Range: 1–32

Note: If you use the Finnish (FIN) language prefix, the default *w* is 11. If you use the Spanish (ESP) language prefix, the default *w* is 10. △

Details

If necessary, SAS truncates the name of the month to fit the format width. The EURDFMNw. format writes SAS date values in the form *month-name*, where

month-name
is the name of the month.

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

Examples

The example table uses the input value 15344, which is the SAS date value that corresponds to January 4, 2002. The first PUT statement assumes the DFLANG= system option is set to Italian.

```
options dflang=ita;
```

The second PUT statement uses the Italian language prefix in the format to write the name of the month in Italian. The third PUT statement uses German language prefix in the format to write the name of the month in German. The value of the DFLANG= option, therefore, is ignored.

SAS Statements	Results
	`----+----1`
`put date eurdfmn10.;`	`janvier`
`put date itadfmn10.;`	`Gennaio`
`put date deudfmn10.;`	`Januar`

See Also

Format:
"MONNAME*w*." on page 136
Function:
"DATE" on page 312
Informat:
"EURDFDE*w*. "on page 671
System Option:
"DFLANG=" on page 1085

EURDFMY*w*.

Writes international date values in the form *mmmyy* or *mmmyyyy*

Category: Date and Time
Alignment: right

Syntax

EURDFMY*w*.

Syntax Description

w
specifies the width of the output field.

Default: 5 (except for Finnish)

Range: 5–7

Note: If you use the Finnish (FIN) language prefix, *w* must be 8, which is the default value. △

Details

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

Examples

The example table uses the input value 15342, which is the SAS date value that corresponds to January 2, 2002. The first PUT statement assumes the DFLANG= system option is set to Spanish.

```
options dflang=spanish;
```

The second PUT statement uses the Spanish language prefix in the format to write the name of the month in Spanish. The third PUT statement uses the French language prefix in the format to write the name of the month in French. The value of the DFLANG= option, therefore, is ignored.

SAS Statements	Results
	`----+----1`
`put date eurdfmy7.;`	`ene2002`
`put date espdfmy7.;`	`ene2002`
`put date fradfmy7.;`	`jan2002`

See Also

Formats:

“DDMMYY*w*.” on page 94

“MMDDYY*w*.” on page 131

“MONYY*w*.” on page 138

“YYMMDD*w*.” on page 188

Functions:

“MONTH” on page 451

“YEAR” on page 618

Informats:

“EURDFMY*w*. ”on page 675

“MONYY*w*.” on page 689

System Option:

“DFLANG=” on page 1085

EURDFWDX*w*.

Writes international date values as the name of the month, the day, and the year in the form *dd month-name yy* (or *yyyy*)

Category: Date and Time

Alignment: right

Syntax

EURDFWDX*w*.

Syntax Description

w

specifies the width of the output field.

Default: depends on the language prefix you use. The following table shows the default for each language:

Language	Maximum	Default
Afrikaans (AFR)	37	29
Catalan (CAT)	40	16
Croatian (CRO)	40	16
Czech (CSY)	40	16
Danish (DAN)	18	18
Dutch (NLD)	37	29
Finnish (FIN)	20	20
French (FRA)	18	18
German (DEU)	18	18
Hungarian (HUN)	40	18
Italian (ITA)	17	17
Macedonian (MAC)	40	17
Norwegian (NOR)	17	17
Polish (POL)	40	20
Portuguese (PTG)	37	23
Russian (RUS)	40	16
Slovenian (SLO)	40	17
Spanish (ESP)	24	24
Swedish (SVE)	17	17
Swiss-French (FRS)	17	17
Swiss-German (DES)	18	18

Range: 3 – maximum width

Tip: If *w* is too small to write the complete day of the week and the month, SAS abbreviates as necessary.

Details

The EURDFWDX*w*. format writes SAS date values in the form *dd month-name yy* or *dd month-name yyyy*, where

dd
is an integer that represents the day of the month.

month-name
is the name of the month.

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

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

Comparisons

The EURDFWKX*w*. format is the same as the EURDFWDX*w*. format except that EURDFWKX *w*. prints *dd* after the month's name.

Examples

The example table uses the input value 15342, which is the SAS date value that corresponds to January 2, 2002. The first PUT statement assumes the DFLANG= system option is set to Dutch.

```
options dflang=dutch;
```

The second PUT statement uses the Dutch language prefix in the format to write the name of the month in Dutch. The third PUT statement uses the Italian language prefix in the format to write the name of the month in Italian. The value of the DFLANG= option, therefore, is ignored.

SAS Statements	Results
	`----+----1----+----2----+----3`
`put day eurdfwdx29.;`	`2 januari 2002`
`put day nlddfwdx29.;`	`2 januari 2002`
`put day itadfwdx17.;`	`02 Gennaio 1998`

See Also

Format:

"WORDDATX*w*." on page 182

System Option:

"DFLANG=" on page 1085

EURDFWKX*w*.

Writes international date values as the name of the day and date in the form *day-of-week*, *dd month-name yy* (or *yyyy*)

Category: Date and Time

Alignment: right

Syntax

EURDFWKX*w*.

Syntax Description

w
specifies the width of the output field.

Default: depends on the language prefix you use. The following table shows the default for each language:

Language	Minimum	Maximum	Default
Afrikaans (AFR)	2	38	28
Catalan (CAT)	2	40	27
Croatian (CRO)	3	40	27
Czech (CSY)	2	40	25
Danish (DAN)	2	31	31
Dutch (NLD)	2	38	28
Finnish (FIN)	2	37	37
French (FRA)	3	27	27
German (DEU)	3	30	30
Hungarian (HUN)	3	40	28
Italian (ITA)	3	28	28
Macedonian (MAC)	3	40	29
Norwegian (NOR)	3	26	26
Polish (POL)	2	40	34
Portuguese (PTG)	3	38	38
Russian (RUS)	2	40	29
Slovenian (SLO)	3	40	29
Spanish (ESP)	1	35	35
Swedish (SVE)	3	26	26
Swiss-French (FRS)	3	26	26
Swiss-German (DES)	3	30	30

Tip: If *w* is too small to write the complete day of the week and the month, SAS abbreviates as necessary.

Details

The EURDFWKX*w*. format writes SAS date values in the form *day-of-week*, *dd month-name yy* (or *yyyy*) where

day-of-week
is the name of day.

dd
is an integer that represents the day of the month.

month-name
is the name of the month.

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

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 for the list of language prefixes. When you specify the language prefix in the format, SAS ignores the DFLANG= option.

Comparisons

The EURDFWKX*w*. format is the same as the EURDFWDX*w*. format except that EURDFWKX*w*. prints *dd* after the month's name.

Examples

The example table uses the input value 15344, which is the SAS date value that corresponds to January 4, 2002. The first PUT statement assumes the DFLANG= system option is set to German.

```
options dflang=German;
```

The second PUT statement uses the German language prefix in the format to write the name of the month in German. The third PUT statement uses the Italian language prefix in the format to write the name of the month in Italian. The value of the DFLANG= option, therefore, is ignored.

Values	Results
	`----+----1----+----2----+----3`
`put date eurdfwkx30.;`	`Freitag, 4. Januar 2002`

Values	Results
`put date deudfwkx30.;`	`Freitag, 4. Januar 2002`
`put date itadfwkx17.;`	`Ven, 04 Gen 2002`

See Also

Formats:

"DATE*w*." on page 89
"DDMMYY*w*." on page 94
"MMDDYY*w*." on page 131
"TOD*w.d*" on page 175
"WEEKDATX*w*." on page 179
"YYMMDD*w*." on page 188

Functions:

"JULDATE" on page 416
"MDY" on page 443
"WEEKDAY" on page 617

Informats:

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

System Option:

"DFLANG=" on page 1085

FLOAT*w.d*

Generates a native single-precision, floating-point value by multiplying a number by 10 raised to the *d*th power

Category: Numeric
Alignment: left

Syntax

FLOAT*w.d*

Syntax Description

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

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

Details

This format is useful in operating environments where a float value is not the same as a truncated double. Values that are written by FLOAT4. typically are those meant to be read by some other external program that runs in your operating environment and that expects these single-precision values.

Note: If the value that is to be formatted is a missing value, or if it is out-of-range for a native single-precision, floating-point value, a single-precision value of zero is generated. △

On IBM mainframe systems, a four-byte floating point number is the same as a truncated eight-byte floating point number. However, in operating environments using the IEEE floating-point standard, such as IBM PC-based operating environments and most UNIX operating environments, a four-byte floating-point number is not the same as a truncated double. Hence, the RB4. format does not produce the same results as the FLOAT4. format. Floating-point representations other than IEEE may have this same characteristic.

Comparisons

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

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

Examples

```
put x float4.;
```

Values	Results*
1	**3F800000**

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

FRACT*w*.

Converts numeric values to fractions

Category: Numeric

Alignment: right

Syntax

FRACT*w*.

Syntax Description

w
specifies the width of the output field.

Default: 10

Range: 4–32

Details

Dividing the number 1 by 3 produces the value 0.33333333. To write this value as 1/3, use the FRACT*w*. format. FRACT*w*. writes fractions in reduced form, that is, 1/2 instead of 50/100.

Examples

```
put x fract8.;
```

Values	Results
	----+----1
0.6666666667	2/3
0.2784	174/625

HEX*w*.

Converts real binary (floating-point) values to hexadecimal representation

Category: Numeric

Alignment: left

Syntax

HEX*w*.

Syntax Description

w
specifies the width of the output field.
Default: 8
Range: 1–16
Tip: If *w*< 16, the HEX*w*. format converts real binary numbers to fixed-point integers before writing them as hexadecimal digits. It also writes negative numbers in two's complement notation, and right aligns digits. If *w* is 16, HEX*w*. displays floating-point values in their hexadecimal form.

Details

In any operating environment, the least significant byte written by HEX*w*. is the rightmost byte. Some operating environments store integers with the least significant digit as the first byte. The HEX*w*. format produces consistent results in any operating environment regardless of the order of significance by byte.

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

Comparisons

The HEX*w*. numeric format and the $HEX*w*. character format both generate the hexadecimal equivalent of values.

Examples

```
put @8 x hex8.;
```

Values	Results
	----+----1----+----2
35.4	00000023
88	00000058
2.33	00000002
-150	FFFFFF6A

HHMM*w.d*

Writes time values as hours and minutes in the form *hh:mm*

Category: Date and Time
Alignment: right

Syntax

HHMM*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 5

Range: 2–20

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

Requirement: must be less than *w*

Range: 1–19

Details

The HHMM*w.d* format writes SAS datetime values in the form *hh:mm*, 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.

SAS rounds hours and minutes that are based on the value of seconds in a SAS time value.

Comparisons

The HHMM*w.d* format is similar to the TIME*w.d* format except that the HHMM*w.d* format does not print seconds.

Examples

The example table uses the input value of 46796, which is the SAS time value that corresponds to 12:59:56 PM.

SAS Statements	Results
	`----+----1`
`put time hhmm.;`	`13:00`

SAS rounds up the time value four seconds based on the value of seconds in the SAS time value.

See Also

Formats:

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

Functions:

"HMS" on page 394
"HOUR" on page 395
"MINUTE" on page 445
"SECOND" on page 542
"TIME" on page 563

Informat:

"TIME*w*." on page 727

HOUR*w.d*

Writes time values as hours and decimal fractions of hours

Category: Date and Time
Alignment: right

Syntax

HOUR*w.d*

Syntax Description

w
specifies the width of the output field.
Default: 2
Range: 2–20

d
optionally specifies the number of digits to the right of the decimal point in the hour value. Therefore, SAS prints decimal fractions of the hour.
Requirement: must be less than *w*
Range: 0-19

Details

SAS rounds hours based on the value of minutes in the SAS time value.

Examples

The example table uses the input value of 41400, which is the SAS time value that corresponds to 11:30 AM.

SAS Statements	Results
	`----+----1`
`put time hour4.1;`	`11.5`

See Also

Formats:

"HHMM*w.d*" on page 121
"MMSS*w.d*" on page 134
"TIME*w.d*" on page 172
"TOD*w.d*" on page 175

Functions:

"HMS" on page 394
"HOUR" on page 395
"MINUTE" on page 445
"SECOND" on page 542
"TIME" on page 563

Informat:

"TIME*w*." on page 727

IB*w.d*

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

Category: Numeric
Alignment: left

Syntax

IB*w.d*

Syntax Description

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

d
optionally specifies to multiply the number by 10^{d}.

Details

The IB*w.d* format writes integer binary (fixed-point) values, including negative values that are represented in two's complement notation. IB*w.d* writes 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 55. △

Comparisons

The IB*w.d* and PIB*w.d* formats are used to write native format integers. (Native format allows you to read and write values created in the same operating environment.) The IBR*w.d* and PIBR*w.d* formats are used to write little 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 3.1 on page 56.

To view a table that compares integer binary notation in several programming languages, see Table 3.2 on page 56.

Examples

```
y=put(x,ib4.);
put y $hex8.;
```

Values	Results on Big Endian Platforms*	Results on Little Endian Platforms*
	`----+----1`	`----+----1`
`128`	`00000080`	`80000000`

* The result is a hexadecimal representation of a four-byte integer binary number. Each byte occupies one column of the output field.

See Also

Format:

"IBR*w.d*" on page 125

IBR*w.d*

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

Category: Numeric

Alignment: left

Syntax

IBR*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 4

Range: 1–8

d
optionally specifies to multiply the number by 10^{d}.

Details

The IBR*w.d* format writes integer binary (fixed-point) values, including negative values that are represented in two's complement notation. IBR*w.d* writes integer binary values that are generated by and for Intel and DEC operating environments. Use IBR*w.d* to write integer binary data from Intel or DEC environments on other operating environments. The IBR*w.d* format in SAS code allows for a portable implementation for writing 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 55. △

Comparisons

- The IB*w.d* and PIB*w.d* formats are used to write native format integers. (Native format allows you to read and write values that are created in the same operating environment.)
- The IBR*w.d* and PIBR*w.d* formats are used to write little endian integers, regardless of the operating environment you are writing on.
- In Intel and DEC operating environments, the IB*w.d* and IBR*w.d* formats are equivalent.

To view a table that shows the type of format to use with big endian and little endian integers, see Table 3.1 on page 56.

To view a table that compares integer binary notation in several programming languages, see Table 3.2 on page 56.

Examples

```
y=put(x,ibr4.);
put y $hex8.;
```

Values	Results*
	`----+----1`
`128`	`80000000`

* The result is a hexadecimal representation of a 4-byte integer binary number. Each byte occupies one column of the output field.

See Also

Format:

"IB*w.d*" on page 124

IEEE*w.d*

Generates an IEEE floating-point value by multiplying a number by 10 raised to the *d*th power

Category: Numeric

Alignment: left

Syntax

IEEE*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 8

Range: 3–8

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

d

optionally specifies to multiply the number by 10^d.

Details

This format is useful in operating environments where IEEE*w.d* is the floating-point representation that is used. In addition, you can use the IEEE*w.d* format to create files that are used by programs in operating environments that use the IEEE floating-point representation.

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

Examples

```
test1=put(x,ieee4.);
put test1 $hex8.;

test2=put(x,ieee5.);
put test2 $hex10.;
```

Values	Results*
1	**3F800000**
	3FF0000000

* The result contains hexadecimal representations of binary numbers stored in IEEE form.

JULDAY*w*.

Writes date values as the Julian day of the year

Category: Date and Time
Alignment: right

Syntax

JULDAY*w*.

Syntax Description

w
specifies the width of the output field.
Default: 3
Range: 3–32

Details

The JULDAY*w*. format writes SAS date values in the form *ddd*, where

ddd
is the number of the day, 1–365 (or 1–366 for leap years).

Examples

The example table uses the input values of 13515, which is the SAS date value that corresponds to January 1, 1997, and 13589, which is the SAS date value that corresponds to March 16, 1997.

SAS Statements	Results
	`----+----1`
`put date julday3.;`	`1`
`put date julday3.;`	`75`

JULIAN*w*.

Writes date values as Julian dates in the form *yyddd* or *yyyyddd*

Category: Date and Time
Alignment: left

Syntax

JULIAN*w*.

Syntax Description

w
specifies the width of the output field.

Default: 5

Range: 5–7

Tip: If *w* is 5, the JULIAN*w*. format writes the date with a two-digit year. If *w* is 7, the JULIAN*w*. format writes the date with a four-digit year.

Details

The JULIAN*w*. format writes SAS date values in the form *yyddd* or *yyyyddd*, where

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

ddd
is the number of the day, 1–365 (or 1–366 for leap years), in that year.

Examples

The example table uses the input value of 15342, which is the SAS date value that corresponds to January 2, 2002 (the 2nd day of the year).

SAS Statements	Results
	`----+----1`
`put date julian5.;`	`02002`
`put date julian7.;`	`2002002`

See Also

Functions:

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

Informat:

"JULIAN*w*." on page 685

MINGUO*w*.

Writes date values as Taiwanese dates in the form *yyymmdd*

Category: Date and Time
Alignment: left

Syntax

MINGUO*w*.

Syntax Description

w
specifies the width of the output field.
Default: 8
Range: 1–10

Details

The MINGUO*w*. format writes SAS date values in the form *yyyymmdd*, where

yyyy
is an integer that represents the year.

mm
is an integer that represents the month.

dd
is an integer that represents the day of the month.

The Taiwanese calendar uses 1912 as the base year (01/01/01 is January 1, 1912). Dates prior to 1912 appear as a series of asterisks. Year values do not roll around after 100 years; instead, they continue to increase.

Examples

The example table uses the following input values:

1 12054 is the SAS date value that corresponds to January 1, 1993.
2 18993 is the SAS date value that corresponds to January 1, 2012.
3 -20088 is the SAS date value that corresponds to January 1, 1905.

SAS Statements	Results
	`----+----1`
`put date minguo10.;`	`0082/01/01`
	`0101/01/01`
	`**********`

MMDDYY*w*.

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

Category: Date and Time
Alignment: right

Syntax

MMDDYY*w*.

Syntax Description

w
specifies the width of the output field.
Default: 8
Range: 2–10

Details

The MMDDYY*w*. format writes SAS date values in the form *mmddyy* or *mmddyyyy*, where

mm
is an integer that represents the month.

dd
is an integer that represents the day of the month.

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

Examples

The example table uses the input value of 15595, which is the SAS date value that corresponds to September 12, 2002.

SAS Statements	Results
	`----+----1----+`
`put day mmddyy2.;`	`09`
`put day mmddyy3.;`	`09`
`put day mmddyy4.;`	`0912`
`put day mmddyy5.;`	`09/12`
`put day mmddyy6.;`	`091202`
`put day mmddyy7.;`	`091202`
`put day mmddyy8.;`	`09/12/02`
`put day mmddyy10.;`	`09/12/2002`

See Also

Formats:

"DATE*w*." on page 89
"DDMMYY*w*." on page 94
"YYMMDD*w*." on page 188

Functions:

"DAY" on page 315
"MDY" on page 443
"MONTH" on page 451
"YEAR" on page 618

Informats:

"DATE*w*." on page 666
"DDMMYY*w*." on page 669
"YYMMDD*w*." on page 733

MMDDYY*xw.*

Writes date values in the form *mmddyy* or *mmddyyyy* with a specified separator

Category: Date and Time
Alignment: right

Syntax

MMDDYY*xw.*

Syntax Description

x
specifies a separator or no separator, where

B
separates with a blank

C
separates with a colon

D
separates with a dash

N
indicates no separator

P
separates with a period

S
separates with a slash.

w
specifies the width of the output field.

Default: 8

Range: 2–10

Tip: When *w* is from 2 to 5, SAS prints as much of the month and day as possible. When *w* is 7, the date appears as a two-digit year without separators, and the value is right aligned in the output field.

Note: When *x* is N, the width range is 2–8. △

Details

The MMDDYY*xw*. format writes SAS date values in the form *mmddyy* or *mmddyyyy*, where

mm
is an integer that represents the month.

dd
is an integer that represents the day of the month.

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

Examples

The example table uses the input value of 15595, which is the SAS date value that corresponds to September 12, 2002.

SAS Statements	Results
	`----+----1----+`
`put day mmddyyc5.;`	`09:12`
`put day mmddyyd8.;`	`09-12-02`
`put day mmddyyp10.;`	`09.12.2002`
`put day mmddyyn8.;`	`09122002`

See Also

Formats:

"DATE*w*." on page 89
"DDMMYY*xw*." on page 95
"YYMMDD*xw*." on page 190

Functions:

"DAY" on page 315
"MDY" on page 443
"MONTH" on page 451
"YEAR" on page 618

Informat:

"MMDDYY*w*." on page 687

MMSS*w.d*

Writes time values as the number of minutes and seconds since midnight

Category: Date and Time
Alignment: right

Syntax

MMSS*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 5

Range: 2–20

Tip: Make *w* at least 5 to write a value that represents minutes and seconds.

d
optionally specifies the number of digits to the right of the decimal point in the seconds value. Therefore, the SAS time value includes fractional seconds.

Range: 0–19
Restriction: must be less than *w*

Examples

The example table uses the input value of 4530.

SAS Statements	Results
	`----+----1`
`put time mmss.;`	`75:30`

See Also

Formats:
- "HHMM*w.d*" on page 121
- "TIME*w.d*" on page 172

Functions:
- "HMS" on page 394
- "MINUTE" on page 445
- "SECOND" on page 542

Informat:
- "TIME*w*." on page 727

MMYY*xw*.

Writes date values as the month and the year and separates them with a character

Category: Date and Time
Alignment: right

Syntax

MMYY*xw*.

Syntax Description

x
specifies the separators, which can be colons, dashes, periods, or other specific characters. For a list of the possible values of *x*, see the table in "Details" on page 136.

w
specifies the width of the output field.
Default: 7

Range: 5–32

Interaction: when no separator is specified by setting *x* to *N*, the width range is 4–32 and the default changes to 6.

Tip: If *w* is too small to print a four-digit year, only the last two digits of the year print.

Details

The following table shows the separators that correspond to the possible *x* values:

Syntax	Separator
MMYY*w*.	uppercase M
MMYYC*w*.	colon
MMYYD*w*.	dash
MMYYN*w*.	no separator
MMYYP*w*.	period
MMYYS*w*.	forward slash

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1----+`
`put date mmyy5.;`	`03M02`
`put date mmyyc7.;`	`03:2002`
`put date mmyyd7.;`	`03-2002`
`put date mmyyn6.;`	`032002`
`put date mmyyp6.;`	`03.02`
`put date mmyys7.;`	`03/2002`

See Also

Format:

"YYMM*x*w." on page 187

MONNAME*w*.

Writes date values as the name of the month

Category: Date and Time

Alignment: right

Syntax

MONNAME*w*.

Syntax Description

w
specifies the width of the output field.

Default: 9

Range: 1–32

Tip: Use MONNAME3. to print the first three letters of the month name.

Details

If necessary, SAS truncates the name of the month to fit the format width.

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1`
`put date monname1.;`	`M`
`put date monname3.;`	`Mar`
`put date monname5.;`	`March`

See Also

Format:

"MONTH*w*." on page 137

MONTH*w*.

Writes date values as the month of the year

Category: Date and Time

Alignment: right

Syntax

MONTH*w*.

Syntax Description

w
specifies the width of the output field.
Default: 2
Range: 1–21
Tip: Use MONTH1. to obtain a hex value.

Details

The MONTH*w*. format writes the month (01 through 12) of the year from a SAS date value.

Examples

The example table uses the input value of 15656, which is the SAS date value that corresponds to November 12, 2002.

SAS Statements	Results
	`----+----1`
`put date month.;`	`11`

See Also

Format:
"MONNAME*w*." on page 136

MONYY*w*.

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

Category: Date and Time
Alignment: right

Syntax

MONYY*w*.

Syntax Description

w
specifies the width of the output field.
Default: 5
Range: 5–7

Details

The MONYY*w*. format writes SAS date values 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.

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1`
`put date monyy5.;`	`MAR02`
`put date monyy7.;`	`MAR2002`

See Also

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

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

Informat:
- "MONYY*w*." on page 689

NEGPAREN*w.d*

Writes negative numeric values in parentheses

Category: Numeric
Alignment: right

Syntax

NEGPAREN*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 6

Range: 1–32

d

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

Range: 0–2

Details

The NEGPAREN*w.d* format displays nonnegative numbers with blanks instead of parentheses for proper column alignment. That is, NEGPAREN*w.d* reserves the last column for a right parenthesis, even when the value is positive. If the field is wide enough, NEGPAREN*w.d* places parentheses around a number to represent a negative value. Otherwise, it uses a minus sign.

Comparisons

The NEGPAREN*w.d* format is similar to the COMMA*w.d* format in that it separates every three digits of the value with a comma.

Examples

```
put @1 sales negparen5.;
```

Values	Results
	----+----1----+
100	100
1000	1,000
-2000	(200)
-2000	-2,000

NENGO*w*.

Writes date values as Japanese dates in the form *e.yymmdd*

Category: Date and Time

Alignment: left

Syntax

NENGO*w*.

Syntax Description

w
specifies the width of the output field.
Default: 10
Range: 2–10

Details

The NENGO*w*. format writes SAS date values in the form *e.yymmdd*, 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 that represents the month.

dd
is an integer that represents the day of the month.

If the width is too small, SAS omits the period.

Examples

The example table uses the input value of 15342, which is the SAS date value that corresponds to January 2, 2002.

SAS Statements	Results
	`----+----1`
`put date nengo3.;`	`H14`
`put date nengo6.;`	`H14/01`
`put date nengo8.;`	`H.140102`
`put date nengo9.;`	`H14/01/02`
`put date nengo10.;`	`H.14/01/02`

See Also

Informat:
"NENGO*w*. "on page 691

NUMX*w.d*

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

Category: Numeric
Alignment: right

Syntax

NUMX*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 12

Range: 1–32

d
optionally specifies the number of digits to the right of the decimal point (comma) in the numeric value.

Details

The NUMX*w.d* format writes numeric values with a comma in place of the decimal point.

Comparisons

The NUMX*w.d* format is similar to the *w.d* format except that NUMX*w.d* writes numeric values with a comma in place of the decimal point.

Examples

```
put x numx10.2;
```

Values	Results
	----+----1----+
896.48	896,48
64.89	64,89
3064.10	3064,10

See Also

Format:

" *w.d*" on page 176

Informat:

"NUMX*w.d*" on page 693

OCTAL*w*.

Converts numeric values to octal representation

Category: Numeric
Alignment: left

Syntax

OCTAL*w*.

Syntax Description

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

Details

If necessary, the OCTAL*w*. format converts numeric values to integers before displaying them in octal representation. The decimal portion is truncated.

Comparisons

OCTAL*w*. converts numeric values to octal representation. The $OCTAL*w*. format converts character values to octal representation.

Examples

```
put x octal6.;
```

Values	Results
	`----+----1`
`3592`	`007010`

PD*w.d*

Writes data in packed decimal format

Category: Numeric
Alignment: left

Syntax

PD*w.d*

Syntax Description

w
specifies the width of the output field. The *w* value specifies the number of bytes, not the number of digits. (In packed decimal data, each byte contains two digits.)

Default: 1

Range: 1–16

d
optionally specifies to multiply the number by 10^{d}.

Details

Different operating environments store packed decimal values in different ways. However, the PD*w.d* format writes packed decimal values with consistent results if the values are created in the same kind of operating environment that you use to run SAS.

Comparisons

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

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

Examples

```
y=put(x,pd4.);
put y $hex8.;
```

Values	Results*
	----+----1
128	**0000128**

* The result is a hexadecimal representation of a binary number written in packed decimal format. Each byte occupies one column of the output field.

PDJULG*w*.

Writes packed Julian date values in the hexadecimal format *yyyyddd*F for IBM

Category: Date and Time

Syntax

PDJULG*w*.

Syntax Description

w
specifies the width of the output field.
Default: 4
Range: 4

Details

The PDJULG*w*. format writes SAS date values in the form *yyyyddd*F, 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

```
juldate = put(date,pdjulg4.);
put juldate $hex8.;
```

SAS date value	Results*
`----+----1`	
`15342`	`2002002F`

* This value represents January 2, 2002.

See Also

Formats:

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

Functions:

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

Informats:

"PDJULI*w*." on page 697
"PDJULG*w*." on page 696
"JULIAN*w*." on page 685

System Option:

"YEARCUTOFF=" on page 1177

PDJULI*w*.

Writes packed Julian date values in the hexadecimal format *ccyyddd*F for IBM

Category: Date and Time

Syntax

PDJULI*w*.

Syntax Description

w
specifies the width of the output field.
Default: 4
Range: 4

Details

The PDJULI*w*. format writes SAS date values in the form *ccyyddd*F, where

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

yy
is the one-byte representation of a two-digit integer that represents the year. The PDJULI*w*. format makes an adjustment for the century byte by subtracting 1900 from the 4–digit Gregorian year to produce the correct packed decimal *ccyy* representation. A year value of 1998 is stored in *ccyy* as 0098, and a year value of 2011 is stored as 0111.

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

```
juldate = put(date, pdjuli4.);
put juldate $hex8.;
```

SAS date value	Results*
13881	**0098002F**
18630	**0111003F**

* The result is a hexadecimal four-byte packed decimal Julian date in the format of *ccyyddd*F. The first SAS date value represents the date of January 2, 1998. The second SAS date value represents the date of January 3, 2011.

See Also

Formats:

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

Functions:

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

Informats:

"PDJULG*w*." on page 696
"PDJULI*w*." on page 697
"JULIAN*w*." on page 685

System Option:

"YEARCUTOFF=" on page 1177

PERCENT*w.d*

Writes numeric values as percentages

Category: Numeric
Alignment: right

Syntax

PERCENT*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 6

Range: 4–32

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

Range: 0–2

Details

The PERCENT*w.d* format multiplies values by 100, formats them the same as the BEST*w.d* format, and adds a percent sign (%) to the end of the formatted value, while it

encloses negative values in parentheses. The PERCENT*w.d* format allows room for a percent sign and parentheses, even if the value is not negative.

Examples

```
put @10 gain percent10.;
```

Values	Results
	----+----1----+----2
0.1	10%
1.2	120%
-0.05	(5%)

PIB*w.d*

Writes positive integer binary (fixed-point) values

Category: Numeric
Alignment: left

Syntax

PIB*w.d*

Syntax Description

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

d
optionally specifies to multiply the number by 10^{d}.

Details

All values are treated as positive. PIB*w.d* writes 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 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 55. △

Comparisons

- Positive integer binary values are the same as integer binary values except that the sign bit is part of the value, which is always a positive integer. The PIB*w.d* format treats all values as positive and includes the sign bit as part of the value.
- The PIB*w.d* format 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 PIB*w.d* format is the same as the IB*w.d* format except that PIB*w.d* treats all values as positive values.
- The IB*w.d* and PIB*w.d* formats are used to write native format integers. (Native format allows you to read and write values that are created in the same operating environment.) The IBR*w.d* and PIBR*w.d* formats are used to write little 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 3.1 on page 56.

  To view a table that compares integer binary notation in several programming languages, see Table 3.2 on page 56.

Examples

```
y=put(x,pib1.);
put y $hex2.;
```

Values	Results*
	`----+----1`
`12`	`0C`

* The result is a hexadecimal representation of a one-byte binary number written in positive integer binary format, which occupies one column of the output field.

See Also

Format:

"PIBR*w.d*" on page 150

PIBR*w.d*

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

Category: Numeric

Syntax

PIBR*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 1

Range: 1–8

d
optionally specifies to multiply the number by 10^{d}.

Details

All values are treated as positive. PIBR*w.d* writes positive integer binary values that have been generated by and for Intel and DEC operating environments. Use PIBR*w.d* to write positive integer binary data from Intel or DEC environments on other operating environments. The PIBR*w.d* format in SAS code allows for a portable implementation for writing 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 55. △

Comparisons

- Positive integer binary values are the same as integer binary values except that the sign bit is part of the value, which is always a positive integer. The PIBR*w.d* format treats all values as positive and includes the sign bit as part of the value.
- The PIBR*w.d* format 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 operating environments, the PIB*w.d* and PIBR*w.d* formats are equivalent.
- The IB*w.d* and PIB*w.d* formats are used to write native format integers. (Native format allows you to read and write values that are created in the same operating environment.) The IBR*w.d* and PIBR*w.d* formats are used to write little 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 3.1 on page 56.

  To view a table that compares integer binary notation in several programming languages, see Table 3.2 on page 56.

Examples

```
y=put(x,pibr2.);
put y $hex4.;
```

Values	Results*
	`----+----1`
`128`	`8000`

* The result is a hexadecimal representation of a two-byte binary number written in positive integer binary format, which occupies one column of the output field.

See Also

Informat:

"PIB*w.d*" on page 701

PK*w.d*

Writes data in unsigned packed decimal format

Category: Numeric
Alignment: left

Syntax

PK*w.d*

Syntax Description

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

d
optionally specifies to multiply the number by 10^{d}.

Details

Each byte of unsigned packed decimal data contains two digits.

Comparisons

The PK*w.d* format is similar to the PD*w.d* format except that PK*w.d* does not write the sign in the low-order byte.

Examples

```
y=put(x,pk4.);
put y $hex8.;
```

Values	Results*
	`----+----1`
`128`	`00000128`

* The result is a hexadecimal representation of a four-byte number written in packed decimal format. Each byte occupies one column of the output field.

PVALUE*w.d*

Writes *p*-values

Category: Numeric

Alignment: right

Syntax

PVALUE*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 6

Range: 3–32

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

Default: the minimum of 4 and *w*–2

Restriction: must be less than *w*

Details

The PVALUE*w.d* format writes *p*-values for the GENMOD procedure, the MIXED procedure, and SAS/INSIGHT software.

Comparisons

The PVALUE*w.d* format follows the rules for the *w.d* format, except that

- all missing values print as "."
- negative values represent exact zero and print as "0.0"
- if the value x is such that 0 <= x < 10^{-d}, x prints as "<.0...01" with *d*-1 zeros

□ if you specify the PROBSIG= option with a value of 1 or 2, and the field width is at least 5, the PVALUE*w*.*d* format uses the BEST*w*. format.

Examples

```
put x pvalue5.3;
```

Values	Results
	----+----1
-1	0.0
0	<.001
1e-8	<.001
.01232456	0.012
.5	0.5000
1	1.000

QTR*w*.

Writes date values as the quarter of the year

Category: Date and Time

Alignment: right

Syntax

QTR*w*.

Syntax Description

w
specifies the width of the output field.

Default: 1

Range: 1–32

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1`
`put date qtr.;`	`1`

See Also

Format:
"QTRR*w*." on page 155

QTRR*w.*

Writes date values as the quarter of the year in Roman numerals

Category: Date and Time
Alignment: right

Syntax

QTRR*w.*

Syntax Description

w
specifies the width of the output field.
Default: 3
Range: 3–32

Examples

The example table uses the input value of 15595, which is the SAS date value that corresponds to September 12, 2002.

SAS Statements	Results
	`----+----1`
`put date qtrr.;`	`III`

See Also

Format:
"QTR*w*." on page 154

RB*w.d*

Writes real binary data (floating-point) in real binary format

Category: Numeric
Alignment: left

Syntax

RB*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 4

Range: 2–8

d
optionally specifies to multiply the number by 10^{d}.

Details

The RB*w.d* format writes numeric data in the same way that SAS stores them. Because it requires no data conversion, RB*w.d* is the most efficient method for writing data with SAS.

Note: Different operating environments store real binary values in different ways. However, RB*w.d* writes real binary values with consistent results in the same kind of operating environment that you use to run SAS. △

CAUTION:
Using RB4. to write real binary data on equipment that conforms to the IEEE standard for floating-point numbers results in a truncated eight-byte (double-precision) number rather than a true four-byte (single-precision) floating-point number. △

Comparisons

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

Language	4 Bytes	8 Bytes
SAS	RB4.	RB8.
FORTRAN	REAL*4	REAL*8
C	float	double
COBOL	COMP-1	COMP-2
IBM 370 assembler	E	D

Examples

```
y=put(x,rb8.);
put y $hex16.;
```

Values	Results*
	`----+---1----+----2`
`128`	`4280000000000000`

* The result is a hexadecimal representation of an eight-byte real binary number as it looks on an IBM mainframe. Each byte occupies one column of the output field.

ROMAN*w*.

Writes numeric values as Roman numerals

Category: Numeric
Alignment: left

Syntax

ROMAN*w*.

Syntax Description

w
specifies the width of the output field.
Default: 6
Range: 2–32

Details

The ROMAN*w*. format truncates a floating-point value to its integer component before the value is written.

Examples

```
put @5 year roman10.;
```

Values	Results
`1998`	`MCMXCVIII`

SSN*w*.

Writes Social Security numbers

Category: Numeric
Alignment: none

Syntax

SSN*w.*

Syntax Description

w
specifies the width of the output field.
Default: 11
Restriction: *w* must be 11

Details

If the value is missing, SAS writes nine single periods with dashes between the third and fourth periods and between the fifth and sixth periods. If the value contains fewer than nine digits, SAS right aligns the value and pads it with zeros on the left. If the value has more than nine digits, SAS writes it as a missing value.

Examples

```
put id ssn11.;
```

Values	Results
	`----+----1----+`
`263878439`	`263-87-8439`

S370FF*w.d*

Writes native standard numeric data in IBM mainframe format

Category: Numeric

Syntax

S370FF*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 12

Range: 1–32

d

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

Range: 0–31

Details

The S370FF*w.d* format writes numeric data in IBM mainframe format (EBCDIC). The EBCDIC numeric values are represented with one byte per digit. If EBCDIC is the native format, S370FF*w.d* performs no conversion.

If a value is negative, an EBCDIC minus sign precedes the value. A missing value is represented as a single EBCDIC period.

Comparisons

On an EBCDIC system, S370FF*w.d* behaves like the *w.d* format.

On all other systems, S370FF*w.d* performs the same role for numeric data that the $EBCDIC*w*. format does for character data.

Examples

```
x=put(x,s370ff3.)
put y $hex10.;
```

Values	Results*
`----+----1`	
`12345`	`F1F2F3F4F5`

* The result is the hexadecimal representation for the integer.

See Also

Formats:

"$EBCDIC*w*." on page 73

" *w.d*" on page 176

S370FIB*w.d*

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

Category: Numeric

Alignment: left

Syntax

S370FIB*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 4

Range: 1–8

d
optionally specifies to multiply the number by 10^{d}.

Details

The S370FIB*w.d* format writes integer binary (fixed-point) values that are stored in IBM mainframe format, including negative values that are represented in two's complement notation. S370FIB*w.d* writes integer binary values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

Use S370FIB*w.d* to write integer binary data in IBM mainframe format from data that are created 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 55. △

Comparisons

- If you use SAS on an IBM mainframe, S370FIB*w.d* and IB*w.d* are identical.
- S370FPIB*w.d*, S370FIBU*w.d*, and S370FIB*w.d* are used to write 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 3.1 on page 56.

  To view a table that compares integer binary notation in several programming languages, see Table 3.2 on page 56.

Examples

```
y=put(x,s370fib4.);
put y $hex8.;
```

Values	Results*
	`----+----1`
`128`	`00000080`

* The result is a hexadecimal representation of a 4-byte integer binary number. Each byte occupies one column of the output field.

See Also

Formats:

"S370FIBU*w.d*" on page 161
"S370FPIB*w.d*" on page 164

S370FIBU*w.d*

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

Category: Numeric
Alignment: left

Syntax

S370FIBU*w.d*

Syntax Description

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

d
optionally specifies to multiply the number by 10^{d}.

Details

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

Use S370FIBU*w.d* to write unsigned integer binary data in IBM mainframe format from data that are created 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 55. △

Comparisons

- The S370FIBU*w.d* format is equivalent to the COBOL notation PIC 9(*n*) BINARY, where *n* is the number of digits.
- The S370FIBU*w.d* format is the same as the S370FIB*w.d* format except that the S370FIBU*w.d* format always uses the absolute value instead of the signed value.

- The S370FPIB*w.d* format writes all negative numbers as FFs, while the S370FIBU*w.d* format writes the absolute value.
- S370FPIB*w.d*, S370FIBU*w.d*, and S370FIB*w.d* are used to write 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 3.1 on page 56.

  To view a table that compares integer binary notation in several programming languages, see Table 3.2 on page 56.

Examples

```
y=put(x,s370fibu1.);
put y $hex2.;
```

Values	Results*
245	**F5**
-245	**F5**

* The result is a hexadecimal representation of a one-byte integer binary number. Each byte occupies one column of the output field.

See Also

Formats:

"S370FIB*w.d*" on page 159

"S370FPIB*w.d*" on page 164

S370FPD*w.d*

Writes packed decimal data in IBM mainframe format

Category: Numeric

Alignment: left

Syntax

S370FPD*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 1

Range: 1–16

d
optionally specifies to multiply the number by 10^d.

Details

Use S370FPD*w.d* in other operating environments to write packed decimal data in the same format as on an IBM mainframe computer.

Comparisons

The following table shows the notation for equivalent packed decimal formats in several programming languages:

Language	Packed Decimal Notation
SAS	S370FPD4.
PL/I	FIXED DEC(7,0)
COBOL	COMP-3 PIC S9(7)
IBM 370 assembler	PL4

Examples

```
y=put(x,s370fpd4.);
put y $hex8.;
```

Values	Results*
	`----+----1`
`128`	`0000128C`

* The result is a hexadecimal representation of a binary number written in packed decimal format. Each byte occupies one column of the output field.

S370FPDU*w.d*

Writes unsigned packed decimal data in IBM mainframe format

Category: Numeric
Alignment: left

Syntax

S370FPDU*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 1

Range: 1–16

d
optionally specifies to multiply the number by 10^{d}.

Details

Use S370FPDU*w.d* in other operating environments to write unsigned packed decimal data in the same format as on an IBM mainframe computer.

Comparisons

- The S370FPDU*w.d* format is similar to the S370FPD*w.d* format except that the S370FPD*w.d* format always uses the absolute value instead of the signed value.
- The S370FPDU*w.d* format is equivalent to the COBOL notation PIC 9(*n*) PACKED-DECIMAL, where the *n* value is the number of digits.

Examples

```
y=put(x,s370fpdu2.);
put y $hex4.;
```

Values	Results*
123	**123F**
-123	**123F**

* The result is a hexadecimal representation of a binary number written in packed decimal format. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one column of the output field.

S370FPIB*w.d*

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

Category: Numeric

Alignment: left

Syntax

S370FPIB*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 4

Range: 1–8

d
optionally specifies to multiply the number by 10^{d}.

Details

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

Use S370FPIB*w.d* to write positive integer binary data in IBM mainframe format from data that are created 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 55. △

Comparisons

- If you use SAS on an IBM mainframe, S370FPIB*w.d* and PIB*w.d* are identical.
- The S370FPIB*w.d* format is the same as the S370FIB*w.d* format except that the S370FPIB*w.d* format treats all values as positive values.
- S370FPIB*w.d*, S370FIBU*w.d*, and S370FIB*w.d* are used to write 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 3.1 on page 56.

  To view a table that compares integer binary notation in several programming languages, see Table 3.2 on page 56.

Examples

```
y=put(x,s370fpib1.);
put y $hex2.;
```

Values	Results*
	`----+----1`
`12`	`0C`

* The result is a hexadecimal representation of a one-byte binary number written in positive integer binary format, which occupies one column of the output field.

See Also

Formats:

"S370FIB*w.d*" on page 159
"S370FIBU*w.d*" on page 161

S370FRB*w.d*

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

Category: Numeric
Alignment: left

Syntax

S370FRB*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 6

Range: 2–8

d
optionally specifies to multiply the number by 10^{d}.

Details

A floating-point value consists of two parts: a mantissa that gives the value and an exponent that gives the value's magnitude.

Use S370FRB*w.d* in other operating environments to write floating-point binary data in the same format as on an IBM mainframe computer.

Comparisons

The following table shows the notation for equivalent floating-point formats in several programming languages:

Language	4 Bytes	8 Bytes
SAS	S370FRB4.	S370FRB8.
PL/I	FLOAT BIN(21)	FLOAT BIN(53)
FORTRAN	REAL*4	REAL*8
COBOL	COMP-1	COMP-2

Language	4 Bytes	8 Bytes
IBM 370 assembler	E	D
C	float	double

Examples

```
y=put(x,s370frb6.);
put y $hex8.;
```

Values	Results*
128	**42800000**
-123	**C2800000**

* The result is a hexadecimal representation of a binary number in zoned decimal format on an IBM mainframe computer. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one column of the output field.

S370FZD*w.d*

Writes zoned decimal data in IBM mainframe format

Category: Numeric

Alignment: left

Syntax

S370FZD*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 8

Range: 1–32

d

optionally specifies to multiply the number by 10^{d}.

Details

Use S370FZD*w.d* in other operating environments to write zoned decimal data in the same format as on an IBM mainframe computer.

Comparisons

The following table shows the notation for equivalent zoned decimal formats in several programming languages:

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

Examples

```
y=put(x,s370fzd3.);
put y $hex6.;
```

Values	Results*
123	**F1F2C3**
-123	**F1F2D3**

* The result is a hexadecimal representation of a binary number in zoned decimal format on an IBM mainframe computer. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one column of the output field.

S370FZDL*w.d*

Writes zoned decimal leading sign data in IBM mainframe format

Category: Numeric
Alignment: left

Syntax

S370FZDL*w.d*

Syntax Description

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

d
optionally specifies to multiply the number by 10^d.

Details

Use S370FZDL*w.d* in other operating environments to write zoned decimal leading-sign data in the same format as on an IBM mainframe computer.

Comparisons

- The S370FZDL*w.d* format is similar to the S370FZD*w.d* format except that the S370FZDL*w.d* format displays the sign of the number in the first byte of the formatted output.
- The S370FZDL*w.d* format is equivalent to the COBOL notation PIC S9(*n*) DISPLAY SIGN LEADING, where the *n* value is the number of digits.

Examples

```
y=put(x,s370fzdl3.);
put y $hex6.;
```

Values	Results*
123	**C1F2F3**
-123	**D1F2F3**

* The result is a hexadecimal representation of a binary number in zoned decimal format on an IBM mainframe computer. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one column of the output field.

S370FZDS*w.d*

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

Category: Numeric
Alignment: left

Syntax

S370FZDS*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 8

Range: 2–32

d

optionally specifies to multiply the number by 10^{d}.

Details

Use S370FZDS*w.d* in other operating environments to write zoned decimal separate leading-sign data in the same format as on an IBM mainframe computer.

Comparisons

- The S370FZDS*w.d* format is similar to the S370FZDL*w.d* format except that the S370FZDS*w.d* format does not embed the sign of the number in the zoned output.
- The S370FZDS*w.d* format is equivalent to the COBOL notation PIC S9(*n*) DISPLAY SIGN LEADING SEPARATE, where the *n* value is the number of digits.

Examples

```
y=put (x,s370fzds4.);
put y $hex8.;
```

Values	Results*
123	**4EF1F2F3**
-123	**60F1F2F3**

* The result is a hexadecimal representation of a binary number in zoned decimal format on an IBM mainframe computer. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one column of the output field.

S370FZDT*w.d*

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

Category: Numeric

Alignment: left

Syntax

S370FZDT*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 8

Range: 2–32

d

optionally specifies to multiply the number by 10^{d}.

Details

Use S370FZDT*w.d* in other operating environments to write zoned decimal separate trailing-sign data in the same format as on an IBM mainframe computer.

Comparisons

- The S370FZDT*w.d* format is similar to the S370FZDS*w.d* format except that the S370FZDT*w.d* format displays the sign of the number at the end of the formatted output.
- The S370FZDT*w.d* format is equivalent to the COBOL notation PIC S9(*n*) DISPLAY SIGN TRAILING SEPARATE, where the *n* value is the number of digits.

Examples

```
y=put (x,s370fzdt4.); ;
put y $hex8.;
```

Values	Results*
123	**F1F2F34E**
-123	**F1F2F360**

* The result is a hexadecimal representation of a binary number in zoned decimal format on an IBM mainframe computer. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one column of the output field.

S370FZDU*w.d*

Writes unsigned zoned decimal data in IBM mainframe format

Category: Numeric

Alignment: left

Syntax

S370FZDU*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 8

Range: 1–32

d
optionally specifies to multiply the number by 10^{d}.

Details

Use S370FZDU*w.d* in other operating environments to write unsigned zoned decimal data in the same format as on an IBM mainframe computer.

Comparisons

- The S370FZDU*w.d* format is similar to the S370FZD*w.d* format except that the S370FZDU*w.d* format always uses the absolute value of the number.
- The S370FZDU*w.d* format is equivalent to the COBOL notation PIC 9(*n*) DISPLAY, where the *n* value is the number of digits.

Examples

```
y=put (x,s370fzdu3.);
put y $hex6.;
```

Values	Results*
123	**F1F2F3**
-123	**F1F2F3**

* The result is a hexadecimal representation of a binary number in zoned decimal format on an IBM mainframe computer. Each two hexadecimal digits correspond to one byte of binary data, and each byte corresponds to one column of the output field.

TIME*w.d*

Writes time values as hours, minutes, and seconds in the form *hh:mm:ss.ss*

Category: Date and Time

Alignment: right

Syntax

TIME*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 8

Range: 2–20

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

Requirement: must be less than *w*

Range: 1–19

Details

The TIME*w.d* format writes SAS time values 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 that range from 00 through 59 with the fraction of a second following the decimal point.

Make *w* large enough to produce the desired results. To obtain a complete time value with three decimal places, you must allow at least 12 spaces: 8 spaces to the left of the decimal point, 1 space for the decimal point itself, and 3 spaces for the decimal fraction of seconds.

Comparisons

The TIME*w.d* format is similar to the HHMM*w.d* format except that TIME*w.d* prints seconds.

Examples

The example table uses the input value of 59083, which is the SAS time value that corresponds to 4:24:43 PM.

SAS Statements	Results
	`----+----1`
`put begin time.;`	`16:24:43`

See Also

Formats:

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

Functions:

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

Informat:

"TIME*w*." on page 727

TIMEAMPM*w.d*

Writes time values as hours, minutes, and seconds in the form *hh:mm:ss.ss* with AM or PM

Category: Date and Time
Alignment: right

Syntax

TIMEAMPM*w.d*

Syntax Description

w
specifies the width of the output field.
Default: 11
Range: 2–20

d
optionally specifies the number of digits to the right of the decimal point in the seconds value.
Requirement: must be less than *w*

Details

The TIMEAMPM*w.d* format writes SAS time values in the form *hh:mm:ss.ss* with AM or PM, where

hh
is an integer that represents the hour.

mm
is an integer that represents the minutes.

ss.ss
is the number of seconds to two decimal places.

Times greater than 23:59:59 PM appear as the next day.

Make *w* large enough to produce the desired results. To obtain a complete time value with three decimal places and AM or PM, you must allow at least 11 spaces (*hh:mm:ss* PM). If *w* is less than 5, SAS writes AM or PM only.

Comparisons

- The TIMEAMPMM*w.d* format is similar to the TIMEM*w.d* format except, that TIMEAMPMM*w.d* prints AM or PM at the end of the time.
- TIME*w.d* writes hours greater than 23:59:59 PM, and TIMEAMPM*w.d* does not.

Examples

The example table uses the input value of 59083, which is the SAS time value that corresponds to 4:24:43 PM.

SAS Statements	Results
	`----+----1----+`
`put begin timeampm3.;`	`PM`
`put begin timeampm5.;`	`4 PM`
`put begin timeampm7.;`	`4:24 PM`
`put begin timeampm11.;`	`4:24:43 PM`

See Also

Format:

"TIME*w.d*" on page 172

TOD*w.d*

Writes the time portion of datetime values in the form *hh:mm:ss.ss*

Category: Date and Time
Alignment: right

Syntax

TOD*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 8

Range: 2–20

d

optionally specifies the number of digits to the right of the decimal point in the seconds value.

Requirement: must be less than *w*

Details

The TOD*w*.*d* format writes SAS datetime values in the form *hh:mm:ss.ss*, where

hh
is an integer that represents the hour.

mm
is an integer that represents the minutes.

ss.ss
is the number of seconds to two decimal places.

Examples

The example table uses the input value of 1347453583, which is the SAS datetime value that corresponds to September 12, 2002, at 12:39:43 PM.

SAS Statements	Results
	`----+----1`
`put begin tod9.2;`	`12:39:43`

See Also

Formats:

"TIME*w*.*d*" on page 172

"TIMEAMPM*w*.*d* "on page 174

Function:

"TIMEPART" on page 564

Informat:

"TIME*w*." on page 727

w.d

Writes standard numeric data one digit per byte

Category: Numeric

Alignment: right

Syntax

w.*d*

Syntax Description

w
specifies the width of the output field.

Range: 1–32

Tip: Allow enough space to write the value, the decimal point, and a minus sign, if necessary.

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

Range: 0–31

Tip: If *d* is 0 or you omit *d*, *w*.*d* writes the value without a decimal point.

Details

The *w*.*d* format rounds to the nearest number that fits in the output field. If *w*.*d* is too small, SAS may shift the decimal to the BEST*w*. format. The *w*.*d* format writes negative numbers with leading minus signs. In addition, *w*.*d* right aligns before writing and pads the output with leading blanks.

Comparisons

The Z*w*.*d* format is similar to the *w*.*d* format except that Z*w*.*d* pads right-aligned output with 0s instead of blanks.

Examples

```
put @7 x 6.3;
```

Values	Results
	`----+----1----+`
`23.45`	`23.450`

WEEKDATE*w*.

Writes date values as the day of the week and the date in the form *day-of-week*, *month-name dd*, *yy* (or *yyyy*)

Category: Date and Time

Alignment: right

Syntax

WEEKDATE*w*.

Syntax Description

w
specifies the width of the output field.

Default: 29

Range: 3–37

Details

The WEEKDATE*w*. format writes SAS date values in the form *day-of-week*, *month-name dd*, *yy* (or *yyyy*), where

dd
is an integer that represents the day of the month.

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

If *w* is too small to write the complete day of the week and month, SAS abbreviates as needed.

Comparisons

The WEEKDATE*w*. format is the same as the WEEKDATX*w*. format except that WEEKDATEX*w*. prints *dd* before the month's name.

Examples

The example table uses the input value of 15415 which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1----+----2`
`put date weekdate3.;`	`Sat`
`put date weekdate9.;`	`Saturday`

SAS Statements	Results
`put date weekdate15.;`	`Sat, Mar 16, 02`
`put date weekdate17.;`	`Sat, Mar 16, 2002`

See Also

Formats:
- "DATE*w*." on page 89
- "DDMMYY*w*." on page 94
- "MMDDYY*w*." on page 131
- "TOD*w.d*" on page 175
- "WEEKDATX*w*." on page 179
- "YYMMDD*w*." on page 188

Functions:
- "JULDATE" on page 416
- "MDY" on page 443
- "WEEKDAY" on page 617

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

WEEKDATX*w*.

Writes date values as day of week and date in the form *day-of-week*, *dd month-name yy* (or *yyyy*)

Category: Date and Time
Alignment: right

Syntax

WEEKDATX*w*.

Syntax Description

w
specifies the width of the output field.
Default: 29
Range: 3–37

Details

The WEEKDATX*w*. format writes SAS date values in the form *day-of-week*, *dd month-name* , *yy* (or *yyyy*), where

dd
is an integer that represents the day of the month.

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

If *w* is too small to write the complete day of the week and month, SAS abbreviates as needed.

Comparisons

The WEEKDATE*w*. format is the same as the WEEKDATX*w*. format except that WEEKDATE*w*. prints *dd* after the month's name.

Examples

The example table uses the input value of 15415 which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1----+----2----+----3`
`put date weekdatex.;`	`Saturday, March 16, 2002`

See Also

Formats:
- "DATE*w*." on page 89
- "DDMMYY*w*." on page 94
- "MMDDYY*w*." on page 131
- "TOD*w.d*" on page 175
- "WEEKDATE*w*." on page 177
- "YYMMDD*w*." on page 188

Functions:
- "JULDATE" on page 416
- "MDY" on page 443
- "WEEKDAY" on page 617

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

WEEKDAY*w*.

Writes date values as the day of the week

Category: Date and Time

Alignment: right

Syntax

WEEKDAY*w*.

Syntax Description

w
specifies the width of the output field.

Default: 1

Range: 1–32

Details

The WEEKDAY*w*. format writes a SAS date value as the day of the week (where 1=Sunday, 2=Monday, and so on).

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statement	Results
	`----+----1`
`put date weekday.;`	`7`

See Also

Format:

"DOWNAME*w*." on page 99

WORDDATE*w*.

Writes date values as the name of the month, the day, and the year in the form *month-name dd, yyyy*

Category: Date and Time

Alignment: right

Syntax

WORDDATE*w*.

Syntax Description

w
specifies the width of the output field.

Default: 18

Range: 3–32

Details

The WORDDATE*w*. format writes SAS date values in the form *month-name dd*, *yyyy*, where

dd
is an integer that represents the day of the month.

yyyy
is a four-digit integer that represents the year.

If the width is too small to write the complete month, SAS abbreviates as necessary.

Comparisons

The WORDDATE*w*. format is the same as the WORDDATX*w*. format except that WORDDATX*w*. prints *dd* before the month's name.

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1----+----2`
`put term worddate3.;`	`Mar`
`put term worddate9.;`	`March`
`put term worddate12.;`	`Mar 16, 2002`
`put term worddate20.;`	`March 16, 2002`

See Also

Format:

"WORDDATX*w*." on page 182

WORDDATX*w*.

Writes date values as the day, the name of the month, and the year in the form *dd month-name yyyy*

Category: Date and Time

Alignment: right

Syntax

WORDDATX*w*.

Syntax Description

w
specifies the width of the output field.
Default: 18
Range: 3–32

Details

The WORDDATX*w*. format writes SAS date values in the form *dd month-name*, *yyyy*, where

dd
is an integer that represents the day of the month.

yyyy
is a four-digit integer that represents the year.

If the width is too small to write the complete month, SAS abbreviates as necessary.

Comparisons

The WORDDATX*w*. format is the same as the WORDDATE*w*. format except that WORDDATE*w*. prints *dd* after the month's name.

Examples

The example table uses the input value of 15595, which is the SAS date value that corresponds to September 12, 2002.

SAS Statement	Results
	`----+----1----+----2`
`put term worddatx.;`	`12 September 2002`

See Also

Format:
"WORDDATE*w*. "on page 181

WORDF*w*.

Writes numeric values as words with fractions that are shown numerically

Category: Numeric
Alignment: left

Syntax

WORDF*w*.

Syntax Description

w
specifies the width of the output field.
Default: 10
Range: 5–32767

Details

The WORDF*w*. format converts numeric values to their equivalent in English words, with fractions that are represented numerically in hundredths. For example, 8.2 prints as eight and 20/100.

Negative numbers are preceded by the word minus. When the value's equivalent in words does not fit into the specified field, it is truncated on the right and the last character prints as an asterisk.

Comparisons

The WORDF*w*. format is similar to the WORDS*w*. format except that WORDF*w*. prints fractions as numbers instead of words.

Examples

```
put price wordf15.;
```

Values	Results
	`----+----1----+`
`2.5`	`two and 50/100`

See Also

Format:

"WORDS*w*." on page 184

WORDS*w*.

Writes numeric values as words

Category: Numeric
Alignment: left

Syntax

WORDS*w*.

Syntax Description

w
specifies the width of the output field.
Default: 10
Range: 5–32767

Details

You can use the WORDS*w*. format to print checks with the amount written out below the payee line.

Negative numbers are preceded by the word minus. If the number is not an integer, the fractional portion is represented as hundredths. For example, 5.3 prints as five and thirty hundredths. When the value's equivalent in words does not fit into the specified field, it is truncated on the right and the last character prints as an asterisk.

Comparisons

The WORDS*w*. format is similar to the WORDF*w*. format except that WORDS*w*. prints fractions as words instead of numbers.

Examples

```
put price words23.;
```

Values	Results
	`----+----1----+----2----+`
`2.1`	`two and ten hundredths`

See Also

Format:
"WORDF*w*." on page 183

YEAR*w*.

Writes date values as the year

Category: Date and Time
Alignment: right

Syntax

YEAR*w.*

Syntax Description

w
specifies the width of the output field.

Default: 4

Range: 2–32

Tip: If *w* is less than 4, the last two digits of the year print; otherwise, the year value prints as four digits.

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1`
`put date year2.;`	`02`
`put date year4.;`	`2002`

YEN*w.d*

Writes numeric values with yen signs, commas, and decimal points

Category: Numeric
Default width: right

Syntax

YEN*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 1

Range: 1–32

d

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

Restriction: must be either 0 or 2

Tip: If *d* is 2, YEN*w*.*d* writes a decimal point and two decimal digits. If *d* is 0, YEN*w*.*d* does not write a decimal point or decimal digits.

Details

The YEN*w*.*d* format writes numeric values with a leading yen sign and with a comma that separates every three digits of each value.

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

Examples

```
put cost yen10.2;
```

Value	Results
	`----+----1`
`1254.71`	`¥1,254.71`

YYMM*xw*.

Writes date values as the year and month and separates them with a character

Category: Date and Time

Alignment: right

Syntax

YYMM*xw*.

Syntax Description

x

specifies the separators, which can be colons, dashes, periods, or other specific characters. For a list of the possible values of *x*, see the table in “Details” on page 188.

w

specifies the width of the output field.

Default: 7

Range: 5–32

Interaction: when no separator is specified by setting *x* to *N*, the width range is 4–32 and the default changes to 6.

Tip: If *w* is too small to print a four-digit year, only the last two digits of the year print.

Details

The following table shows the separators that correspond to the possible *x* values:

Syntax	Separator
YYMM*w*.	uppercase M
YYMMC*w*.	colon
YYMMD*w*.	dash
YYMMN*w*.	no separator
YYMMP*w*.	period
YYMMS*w*.	forward slash

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1----+`
`put date yymm7.;`	`2002M03`
`put date yymmc7.;`	`2002:03`
`put date yymmd7.;`	`2002-03`
`put date yymmn6.;`	`200203`
`put date yymmp7.;`	`2002.03`
`put date yymms7.;`	`2002/03`

See Also

Format:

"MMYY*xw*." on page 135

YYMMDD*w*.

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

Category: Date and Time

Alignment: right

Syntax

YYMMDD*w*.

Syntax Description

w
specifies the width of the output field.

Default: 8

Range: 2–10

Details

The YYMMDD*w*. format writes SAS date values in the form *yymmdd* or *yyyymmdd*, where

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

mm
is an integer that represents the month.

dd
is an integer that represents the day of the month.

Examples

The example table uses the input value of 15595, which is the SAS date value that corresponds to September 12, 2002.

SAS Statements	Results
	`----+----1----+`
`put day yymmdd2.;`	`02`
`put day yymmdd3.;`	`02`
`put day yymmdd4.;`	`0209`
`put day yymmdd5.;`	`02-09`
`put day yymmdd6.;`	`020912`
`put day yymmdd7.;`	`020912`

SAS Statements	Results
`put day yymmdd8.;`	`02-09-12`
`put day yymmdd10.;`	`2002-09-12`

See Also

Formats:

"DATE*w*." on page 89
"DDMMYY*w*." on page 94
"MMDDYY*w*." on page 131

Functions:

"DAY" on page 315
"MDY" on page 443
"MONTH" on page 451
"YEAR" on page 618

Informats:

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

YYMMDD*xw*.

Writes date values in the form *yymmdd* or *yyyymmdd* with a specified separator

Category: Date and Time
Alignment: right

Syntax

YYMMDD*xw.*

Syntax Description

x
specifies a separator or no separator, where

B
separates with a blank

C
separates with a colon

D
separates with a dash

N
indicates no separator

P
separates with a period

S
separates with a slash.

w
specifies the width of the output field.

Default: 8

Range: 2–10

Tip: When *w* is from 2 to 5, SAS prints as much of the month and day as possible. When *w* is 7, the date appears as a two-digit year without separators, and the value is right aligned in the output field.

Note: When *x* is N, the width range is 2–8. △

Details

The YYMMDD*xw*. format writes SAS date values in the form *yymmdd* or *yyyymmdd*, where

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

mm
is an integer that represents the month.

dd
is an integer that represents the day of the month.

Examples

The example table uses the input value of 15595, which is the SAS date value that corresponds to September 12, 2002.

SAS Statements	Results
	`----+----1----+`
`put day yymmddc5.;`	`02:09`
`put day yymmddd8.;`	`02-09-12`

SAS Statements	Results
`put day yymmddp10.;`	`2002.09.12`
`put day yymmddn8.;`	`20020912`

See Also

Formats:
- "DATE*w*." on page 89
- "DDMMYY*xw*." on page 95
- "MMDDYY*xw*." on page 132

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

Informat:
- "YYMMDD*w*." on page 733

YYMON*w*.

Writes date values as the year and the month abbreviation

Category: Date and Time
Alignment: right

Syntax

YYMON*w*.

Syntax Description

w
specifies the width of the output field. If the format width is too small to print a four-digit year, only the last two digits of the year are printed.

Default: 7
Range: 5–32

Details

The YYMON*w*. format abbreviates the month's name to three characters.

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statement	Results
	`----+----1`
`put date yymon6.;`	`02MAR`
`put date yymon7.;`	`2002MAR`

See Also

Format:

"MMYY*xw*." on page 135

YYQ*xw*.

Writes date values as the year and the quarter and separates them with a character

Category: Date and Time
Alignment: right

Syntax

YYQ*xw*.

Syntax Description

x
specifies the separators, which can be colons, dashes, periods, or other specific characters. For a list of the possible values of *x*, see the table in "Details" on page 193.

w
specifies the width of the output field.

Default: 7

Range: 5–32

Interaction: when no separator is specified by setting *x* to *N*, the width range is 4–32 and the default changes to 6.

Tip: If *w* is too small to print a four-digit year, only the last two digits of the year print.

Details

The following table shows the separators that correspond to the possible *x* values:

Syntax	Separator
YYQ*w*. *	uppercase Q
YYQC*w*.	colon
YYQD*w*.	dash

Syntax	Separator
YYQN*w*.	no separator
YYQP*w*.	period
YYQS*w*.	forward slash

* For compatibility with previous versions, the width range of YYQ*w*. without *x* is 4-6.

Examples

The example table uses the input value of 15415, which is the SAS date value that corresponds to March 16, 2002.

SAS Statements	Results
	`----+----1----+`
`put date yyq6.;`	`2002Q1`
`put date yyqc6.;`	`2002:1`
`put date yyqd6.;`	`2002-1`
`put date yyqn5.;`	`20021`
`put date yyqp6.;`	`2002.1`
`put date yyqs6.;`	`2002/1`

See Also

Format:

"YYQR*xw*." on page 194

YYQR*xw*.

Writes date values as the year and the quarter in Roman numerals and separates them with characters

Category: Date and Time

Alignment: right

Syntax

YYQR*xw*.

Syntax Description

x

specifies the separators, which can be colons, dashes, periods, or other specific characters. For a list of the possible values of *x*, see the table in "Details" on page 195.

w
specifies the width of the output field.

Default: 5

Range: 3–32

Tip: If *w* is too small to print a four-digit year, only the last two digits of the year print.

Details

The following table shows the separators that correspond to the possible *x* values:

Syntax	Separator
YYQR*w*.	uppercase Q
YYQRC*w*.	colon
YYQRD*w*.	dash
YYQRN*w*.	no separator
YYQRP*w*.	period
YYQRS*w*.	forward slash

Examples

The example table uses the input value of 15431, which is the SAS date value that corresponds to April 1, 2002.

SAS Statements	Results
	`----+----1----+`
`put date yyqr8.;`	`2002QII`
`put date yyqrc8.;`	`2002:II`
`put date yyqrd8.;`	`2002-II`
`put date yyqrn7.;`	`2002II`
`put date yyqrp8.;`	`2002.II`
`put date yyqrs8.;`	`2002/II`

See Also

Format:

"YYQ*x*w." on page 193

Zw.d

Writes standard numeric data with leading 0s

Category: Numeric

Alignment: right

Syntax

Z*w.d*

Syntax Description

w
specifies the width of the output field.

Default: 1

Range: 1–32

Tip: Allow enough space to write the value, the decimal point, and a minus sign, if necessary.

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

Range: 0–31

Tip: If *d* is 0 or you omit *d*, Z*w.d* writes the value without a decimal point.

Details

The Z*w.d* format writes standard numeric values one digit per byte and fills in 0s to the left of the data value.

The Z*w.d* format rounds to the nearest number that will fit in the output field. If *w.d* is too large to fit, SAS may shift the decimal to the BEST*w*. format. The Z*w.d* format writes negative numbers with leading minus signs. In addition, it right aligns before writing and pads the output with leading zeros.

Comparisons

The Z*w.d* format is similar to the *w.d* format except that Z*w.d* pads right-aligned output with 0s instead of blanks.

Examples

```
put @5 seqnum z8.;
```

Values	Results
	`----+----1`
`1350`	`00001350`

ZD*w.d*

Writes numeric data in zoned decimal format

Category: Numeric
Alignment: left

Syntax

ZD*w.d*

Syntax Description

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

d
optionally specifies to multiply the number by 10^{d}.

Details

The zoned decimal format is similar to standard numeric format in that every digit requires one byte. However, the value's sign is in the last byte, along with the last digit.

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

Comparisons

The following table compares the zoned decimal format with notation in several programming languages:

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

Examples

```
y=put(x,zd4.);
put y $hex8.;
```

Values	Results*
120	**F0F1F2C0**

* The result is a hexadecimal representation of a binary number in zoned decimal format on an IBM mainframe computer. Each byte occupies one column of the output field.

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

SAS® Language Reference, Version 8

ISBN 1–58025–369–5

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

1st printing, October 1999